

CA-ADS[®]

Reference
15.0



Computer Associates™

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

This documentation may not be copied, transferred, reproduced, disclosed or duplicated, in whole or in part, without the prior written consent of CA. This documentation is proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of this documentation for their own internal use, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software are permitted to have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to return to CA the reproduced copies or to certify to CA that same have been destroyed.

To the extent permitted by applicable law, CA provides this documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will CA be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of this documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised of such loss or damage.

The use of any product referenced in this documentation and this documentation is governed by the end user's applicable license agreement.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013(c)(1)(ii) or applicable successor provisions.

Second Edition, October 2001

© 2001 Computer Associates International, Inc.
All rights reserved.

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contents

How to Use This Manual	xiii
------------------------	------

Volume 1. CA-ADS Reference

Chapter 1. Introduction to CA-ADS	1-1
1.1 What is CA-ADS?	1-3
1.2 What CA-ADS does	1-4
1.3 Creating a CA-ADS application	1-5
1.4 Tools used to develop an application	1-7
1.4.1 The CA-ADS application compiler (ADSA)	1-7
1.4.2 Mapping facilities (MAPC and the Batch Compiler/Utility)	1-9
1.4.3 CA-ADS dialog compilers (ADSC and ADSOBCOM)	1-10
1.4.4 IDD menu facility and online IDD	1-12
1.4.5 The CA-ADS runtime system	1-13
1.5 CA-ADS screens	1-14
1.5.1 Action bar	1-15
1.5.2 Action bar actions	1-17
1.6 Checkout and release procedures	1-28
1.6.1 How to check out or release an entity	1-28
1.6.2 Listing checkouts (ADSL)	1-30
1.6.3 Modifying checkouts (ADSM)	1-31
1.7 CA-ADS help facility	1-32
Chapter 2. CA-ADS Application Compiler (ADSA)	2-1
2.1 Overview	2-3
2.2 Application compiler session	2-4
2.2.1 Invoking the application compiler	2-4
2.2.2 Sequencing through application compiler screens	2-7
2.2.3 Suspending a session	2-10
2.2.4 Terminating a session	2-10
2.3 Application compiler screens	2-11
2.3.1 Main menu	2-11
2.3.2 General Options screen—Page 1	2-14
2.3.3 General Options screen—Page 2	2-16
2.3.4 Response/Function List screen	2-19
2.3.5 Response Definition screen	2-23
2.3.6 Function Definition (Dialog) screen	2-27
2.3.7 Function Definition (Program) screen	2-30
2.3.8 Function Definition (Menu) screen	2-32
2.3.9 Global Records screen	2-37
2.3.10 Task Codes screen	2-39
Chapter 3. CA-ADS Dialog Compiler (ADSC)	3-1
3.1 Overview	3-3
3.2 Dialog compiler session	3-4
3.2.1 Invoking the dialog compiler	3-4

3.2.2	Sequencing through dialog compiler screens	3-5
3.2.3	Suspending a session	3-8
3.2.4	Terminating a session	3-9
3.3	Dialog compiler screens	3-10
3.3.1	Main menu	3-10
3.3.2	Options and Directives screen	3-13
3.3.3	Map Specifications screen	3-17
3.3.4	Database Specifications screen	3-20
3.3.5	Records and Tables screen	3-23
3.3.6	Process Modules screen	3-25
Chapter 4.	CA-ADS Runtime System	4-1
4.1	Initiating the CA-ADS runtime system	4-3
4.1.1	How to define runtime tasks	4-3
4.1.2	How to start a CA-ADS application	4-4
4.2	Runtime menu and help screens	4-8
4.2.1	Menu screens	4-8
4.2.2	Site-defined menu maps	4-9
4.2.3	System-defined menu maps	4-10
4.2.4	Application help screen	4-16
4.3	Runtime flow of control	4-19
4.3.1	Effects of automatic editing on flow of control	4-22
4.4	Message prefixes	4-23
4.5	CA-ADS tasks, run units, and transactions	4-24
4.5.1	Run units and database access	4-25
4.5.2	Extended run units	4-25
4.6	Dialog Abort Information screen	4-28
4.7	Debugging a dialog	4-32
4.8	Linking From CA-ADS to CA-OLQ	4-33
4.8.1	Linking to CA-OLQ	4-33
4.8.2	Passing syntax to CA-OLQ	4-33
4.9	Linking built-in functions with the runtime system	4-34
4.9.1	Linking system-supplied built-in functions	4-34
4.9.2	Linking user-written built-in functions	4-39
4.10	Managing storage	4-40
4.10.1	Adjusting record compression	4-40
4.10.2	Calculating RBB storage	4-40
4.10.3	Writing resources to scratch records	4-41
4.10.4	Using XA storage	4-42
Chapter 5.	Introduction to Process Language	5-1
5.1	Overview	5-3
5.2	Process modules	5-5
5.2.1	Creating process modules	5-5
5.2.2	Adding process modules to dialogs	5-5
5.2.3	Executing process modules	5-5
5.3	Process commands	5-7
5.3.1	Constructing commands	5-7
5.3.2	Coding considerations	5-8
5.4	Data types	5-10
5.4.1	Conversion between data types	5-16

Chapter 6. Arithmetic Expressions	6-1
6.1 Overview	6-3
6.2 Syntax	6-4
6.3 Evaluation of arithmetic expressions	6-5
6.4 Coding considerations	6-6
Chapter 7. Built-in Functions	7-1
7.1 Overview	7-3
7.1.1 Invocation names	7-3
7.1.2 Built-in function values	7-4
7.1.3 Coding parameters	7-4
7.2 User-defined built-in functions	7-5
7.3 System-supplied functions	7-6
7.3.1 Arithmetic functions	7-6
7.3.2 Date functions	7-6
7.3.3 String functions	7-7
7.3.4 Trailing-sign functions	7-8
7.3.5 Trigonometric functions	7-9
7.4 ABSOLUTE-VALUE	7-11
7.5 ARC COSINE	7-12
7.6 ARC SINE	7-13
7.7 ARC TANGENT	7-14
7.8 CONCATENATE	7-15
7.9 COSINE	7-16
7.10 DATECHG	7-17
7.11 DATEDIF	7-20
7.12 DATEOFF	7-21
7.13 EXTRACT	7-23
7.14 FIX	7-24
7.15 GOODDATE	7-25
7.16 GOODTRAILING	7-26
7.17 INITCAP	7-27
7.18 INSERT	7-28
7.19 INVERT-SIGN	7-30
7.20 LEFT-JUSTIFY	7-31
7.21 LIKE	7-32
7.22 LOGARITHM	7-34
7.23 MODULO	7-35
7.24 NEXT-INT-EQHI	7-36
7.25 NEXT-INT-EQLO	7-37
7.26 NUMERIC	7-38
7.27 RANDOM-NUMBER	7-40
7.28 REPLACE	7-42
7.29 RIGHT-JUSTIFY	7-44
7.30 SIGN-VALUE	7-45
7.31 SINE	7-46
7.32 SQUARE-ROOT	7-47
7.33 STRING-INDEX	7-48
7.34 STRING-LENGTH	7-49
7.35 STRING-REPEAT	7-50

7.36	SUBSTRING	7-51
7.37	TANGENT	7-53
7.38	TODAY	7-54
7.39	TOLOWER	7-55
7.40	TOMORROW	7-56
7.41	TOUPPER	7-57
7.42	TRAILING-TO-ZONED	7-58
7.43	TRANSLATE	7-59
7.44	VERIFY	7-61
7.45	WEEKDAY	7-62
7.46	WORDCAP	7-65
7.47	YESTERDAY	7-66
7.48	ZONED-TO-TRAILING	7-67
Chapter 8. Conditional Expressions		8-1
8.1	Overview	8-3
8.2	General considerations	8-4
8.2.1	Syntax for conditional expressions	8-4
8.3	Batch-control event condition	8-6
8.4	Command status condition	8-7
8.5	Comparison condition	8-10
8.6	Cursor position condition	8-12
8.7	Dialog execution status condition	8-14
8.8	Environment status condition	8-16
8.9	Level-88 condition	8-17
8.10	Map field status condition	8-18
8.11	Map paging status conditions	8-22
8.12	Set status condition	8-25
8.13	Arithmetic and assignment command status condition	8-27
Chapter 9. Constants		9-1
9.1	Overview	9-3
9.2	Figurative constants	9-4
9.3	Graphic literals	9-6
9.4	Multibit binary constants	9-7
9.5	Nonnumeric literals	9-8
9.6	Numeric literals	9-9
Chapter 10. Error Handling		10-1
10.1	Overview	10-3
10.2	The autostatus facility	10-4
10.3	Error expressions	10-6
10.4	The ALLOWING clause	10-7
10.5	Status definition records	10-9
Chapter 11. Variable Data Fields		11-1
11.1	Overview	11-3
11.2	User-defined data field names	11-4
11.3	System-supplied data field names	11-6
11.4	Entity names	11-12

Chapter 12. Introduction to Process Commands	12-1
12.1 Overview	12-3
12.2 Summary of process commands	12-4
12.3 INCLUDE	12-8
Chapter 13. Arithmetic and Assignment Commands	13-1
13.1 Overview	13-3
13.2 General considerations	13-4
13.2.1 Numeric fields	13-4
13.2.2 EBCDIC and DBCS fields	13-4
13.2.3 Arithmetic and assignment command status condition	13-5
13.3 Arithmetic commands	13-6
13.3.1 ADD	13-6
13.3.2 COMPUTE	13-7
13.3.3 DIVIDE	13-8
13.3.4 MULTIPLY	13-10
13.3.5 SUBTRACT	13-11
13.4 Assignment command	13-13
13.4.1 MOVE	13-14
Chapter 14. Conditional Commands	14-1
14.1 Overview	14-3
14.2 EXIT	14-4
14.3 IF	14-5
14.4 NEXT	14-8
14.5 WHILE	14-10

Volume 2. CA-ADS Reference

Chapter 15. Control Commands	15-1
15.1 Overview	15-3
15.2 General considerations	15-5
15.2.1 Application thread	15-5
15.2.2 Operative and nonoperative dialogs	15-6
15.2.3 Application levels	15-6
15.2.4 Mainline dialog	15-6
15.2.5 The menu stack	15-7
15.2.6 Database currencies	15-7
15.3 CONTINUE	15-10
15.4 DISPLAY	15-12
15.5 EXECUTE NEXT FUNCTION	15-17
15.6 INVOKE	15-19
15.7 LEAVE	15-22
15.8 LINK	15-24
15.9 READ TRANSACTION	15-30
15.10 RETURN	15-31
15.11 TRANSFER	15-34
15.12 WRITE TRANSACTION	15-36

Chapter 16. Database Access Commands	16-1
16.1 Overview	16-3
16.2 Navigational DML	16-5
16.2.1 Overview of navigational database access	16-5
16.2.2 Use of native VSAM data sets	16-7
16.2.3 Record locking	16-9
16.2.4 Suppression of record retrieval locks	16-10
16.2.5 Overview of ACCEPT	16-12
16.2.6 ACCEPT DB-KEY FROM CURRENCY	16-12
16.2.7 ACCEPT DB-KEY RELATIVE TO CURRENCY	16-14
16.2.8 ACCEPT PAGE-INFO	16-16
16.2.9 ACCEPT STATISTICS	16-17
16.2.10 BIND PROCEDURE	16-19
16.2.11 COMMIT	16-20
16.2.12 CONNECT	16-22
16.2.13 DISCONNECT	16-25
16.2.14 ERASE	16-27
16.2.15 Overview of FIND/OBTAIN	16-30
16.2.16 FIND/OBTAIN CALC	16-31
16.2.17 FIND/OBTAIN CURRENT	16-33
16.2.18 FIND/OBTAIN DB-KEY	16-34
16.2.19 FIND/OBTAIN OWNER	16-37
16.2.20 FIND/OBTAIN WITHIN SET/AREA	16-38
16.2.21 FIND/OBTAIN WITHIN SET USING SORT KEY	16-42
16.2.22 GET	16-44
16.2.23 KEEP	16-46
16.2.24 KEEP LONGTERM	16-47
16.2.25 MODIFY	16-53
16.2.26 READY	16-55
16.2.27 RETURN DB-KEY	16-57
16.2.28 ROLLBACK	16-59
16.2.29 STORE	16-60
16.3 Logical Record Facility commands	16-64
16.3.1 Overview of LRF database access	16-64
16.3.2 WHERE clause	16-65
16.3.3 Conditional expression	16-65
16.3.4 Comparison expression	16-66
16.3.5 ERASE	16-68
16.3.6 MODIFY	16-69
16.3.7 OBTAIN	16-70
16.3.8 ON command	16-71
16.3.9 STORE	16-75
Chapter 17. Map Commands	17-1
17.1 Overview	17-3
17.2 Map modification commands	17-4
17.3 Attributes Command	17-5
17.4 CLOSE	17-10
17.5 MODIFY MAP	17-12
17.6 Pageable maps	17-21
17.6.1 Areas of a pageable map	17-21

17.6.2	Map paging session	17-22
17.6.3	Map paging dialog options	17-27
17.6.4	GET DETAIL	17-28
17.6.5	PUT DETAIL	17-30
17.6.6	Creating or modifying a detail occurrence of a pageable map	17-32
17.6.7	Specifying a numeric value associated with an occurrence	17-32
17.6.8	Specifying a message to appear in the message field of an occurrence	17-32
Chapter 18. Queue and Scratch Management Commands		18-1
18.1	Overview	18-3
18.2	Queue records	18-5
18.3	DELETE QUEUE	18-7
18.4	GET QUEUE	18-9
18.5	PUT QUEUE	18-12
18.6	Scratch records	18-15
18.6.1	CA-ADS usage	18-15
18.6.2	CA-ADS/Batch considerations	18-16
18.7	DELETE SCRATCH	18-17
18.8	GET SCRATCH	18-19
18.9	PUT SCRATCH	18-22
Chapter 19. Subroutine Control Commands		19-1
19.1	Overview	19-3
19.2	CALL	19-4
19.3	DEFINE	19-5
19.4	GOBACK	19-6
Chapter 20. Utility Commands		20-1
20.1	Overview	20-3
20.2	ABORT	20-4
20.3	ACCEPT	20-8
20.4	INITIALIZE RECORDS	20-10
20.5	SNAP	20-11
20.6	TRACE	20-13
20.7	WRITE PRINTER	20-14
20.8	WRITE TO LOG/OPERATOR	20-18
Chapter 21. Cooperative Processing Commands		21-1
21.1	Using SEND/RECEIVE commands	21-3
21.1.1	How cooperative processing works	21-3
21.2	Sample cooperative application	21-4
21.2.1	Program A: Client listing (PC)	21-5
21.2.2	Dialog B: Server listing (Mainframe)	21-7
21.3	SEND/RECEIVE commands	21-9
21.4	ALLOCATE	21-10
21.5	CONFIRM	21-13
21.6	CONFIRMED	21-14
21.7	CONTROL SESSION	21-15
21.8	DEALLOCATE	21-17

21.9	PREPARE-TO-RECEIVE	21-19
21.10	RECEIVE-AND-WAIT	21-20
21.11	REQUEST-TO-SEND	21-21
21.12	SEND-DATA	21-22
21.13	SEND-ERROR	21-24
21.14	Design guidelines	21-25
21.15	Understanding conversation states	21-26
21.15.1	Conversation states in a successful data transfer	21-28
21.16	Testing APPC status codes and system fields	21-30
21.16.1	Status codes	21-30
21.16.2	System fields	21-30
21.16.3	When APPC status codes and system field values are returned	21-30
21.16.4	APPCODE and APPCERC	21-31
21.16.5	System fields	21-34
Chapter 22.	OSCaR Commands	22-1
22.1	OSCaR command syntax	22-4
22.1.1	OPEN	22-4
22.1.2	SEND	22-5
22.1.3	CLOSE	22-6
22.1.4	RECEIVE	22-6
22.2	Sample OSCaR application	22-7
22.3	OSCaR to APPC Mapping	22-9
Appendix A.	System Records	A-1
A.1	Overview	A-3
A.2	ADSO-APPLICATION-GLOBAL-RECORD	A-4
A.3	ADSO-APPLICATION-MENU-RECORD	A-15
Appendix B.	CA-ADS Dialog and Application Reporter	B-1
B.1	Overview	B-3
B.2	Dialog reports	B-4
B.3	Application reports	B-15
B.4	Control statements	B-16
B.4.1	APPLICATIONS	B-16
B.4.2	DIALOGS	B-18
B.4.3	LIST	B-21
B.4.4	SEARCH	B-22
B.5	SYSIDMS parameter file	B-24
B.6	JCL and commands to run reports	B-25
Appendix C.	Dialog Statistics	C-1
C.1	Overview	C-3
C.2	Collecting selected statistics	C-4
C.3	Enabling dialog statistics	C-8
C.4	Selecting dialogs	C-9
C.5	Setting a checkpoint interval	C-10
C.6	Collecting and writing statistics	C-11
C.7	Statistics reporting	C-12
Appendix D.	Application and Dialog Utilities	D-1

D.1	Overview	D-3
D.2	ADSOBCOM	D-4
D.2.1	Standard control statements	D-4
D.2.2	Special control statements	D-5
D.2.3	SIGNON	D-5
D.2.4	COMPILE	D-6
D.2.5	DECOMPILE	D-8
D.2.6	Dialog-expression	D-10
D.2.7	JCL and commands	D-30
D.2.7.1	OS/390 JCL	D-30
D.2.7.2	VSE/ESA JCL	D-31
D.2.7.3	VM/ESA commands	D-33
D.2.7.4	BS2000/OSD JCL	D-35
D.3	ADSOBSYS	D-37
D.3.1	Control statements	D-37
D.3.2	SYSTEM statement	D-38
D.3.3	JCL and commands	D-39
D.3.3.1	OS/390 JCL	D-39
D.3.3.2	VSE/ESA JCL	D-42
D.3.3.3	VM/ESA commands	D-44
D.3.3.4	BS2000/OSD JCL	D-46
D.4	ADSOBTAT	D-48
D.4.1	Control statements	D-49
D.4.2	JCL and commands	D-51
D.4.2.1	OS/390 JCL	D-51
D.4.2.2	VSE/ESA JCL	D-52
D.4.2.3	VM/ESA commands	D-54
D.4.2.4	BS2000/OSD JCL	D-55
D.5	ADSOTATU	D-57
D.5.1	TAT update utility screen	D-58
 Appendix E. Activity Logging for a CA-ADS Dialog		E-1
E.1	Overview	E-3
E.2	Data dictionary organization	E-4
E.3	Activity logging record formats	E-5
 Appendix F. Built-in Function Support		F-1
F.1	Overview	F-3
F.2	Internal structure of built-in functions	F-4
F.2.1	Master function table	F-5
F.2.2	Model XDE module	F-6
F.2.3	XDEs and VXDEs	F-8
F.2.4	Processing program modules	F-17
F.2.5	Runtime processing of built-in functions	F-24
F.3	Assembler macros	F-27
F.3.1	#EFUNMST	F-27
F.3.2	RHDCEVBF	F-28
F.3.3	#EFUNMOD	F-31
F.4	Changing invocation names	F-40
F.5	Creating user-defined built-in functions	F-41

F.5.1	Steps for generating a user-defined built-in function	F-41
F.5.2	LRF considerations for user-defined built-in functions	F-42
F.5.3	Calling a user-defined built-in function	F-42
Appendix G. Security Features		G-1
G.1	Overview	G-3
G.2	CA-ADS compiler security	G-4
G.3	CA-ADS application security	G-5
G.3.1	Response security	G-5
G.3.2	Signon security	G-6
Appendix H. Debugging a CA-ADS Dialog		H-1
H.1	Creating a symbol table	H-4
H.2	Trace facility	H-5
H.3	Online debugger	H-7
Index		X-1

How to Use This Manual

What this manual is about

This manual is a reference for Application Development System (CA-ADS®) development tools and facilities. It provides reference information appropriate for application developers defining online and batch applications.

The manual is divided into two volumes.

Volume 1:

- Introduces CA-ADS and provides information about tools used to develop applications
- Introduces the process language and error-handling facility used in developing CA-ADS applications

Volume 2:

- Presents syntax and examples of process language command statements used to construct processing routines
- Presents detailed information about facilities and utilities available to the CA-ADS application developer

New users may find it helpful to familiarize themselves with the *CA-ADS User Guide* before relying solely on this reference manual. Experienced users can use this reference as needed. Developers using CA-ADS/Batch® extensions to CA-ADS to define batch applications should refer to *CA-ADS Batch User Guide*. Syntax for developing batch applications is included in this reference manual.

Related documentation

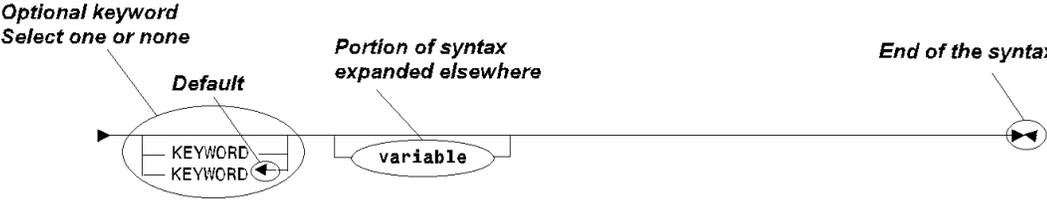
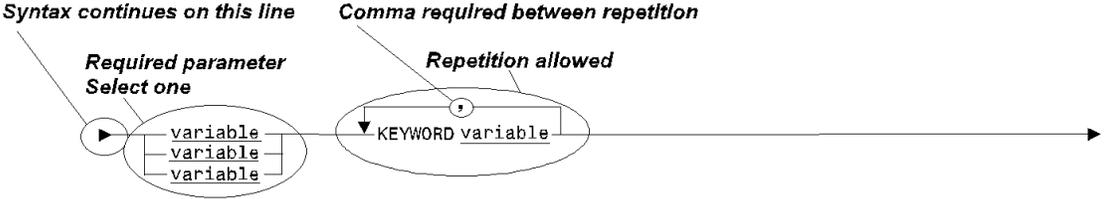
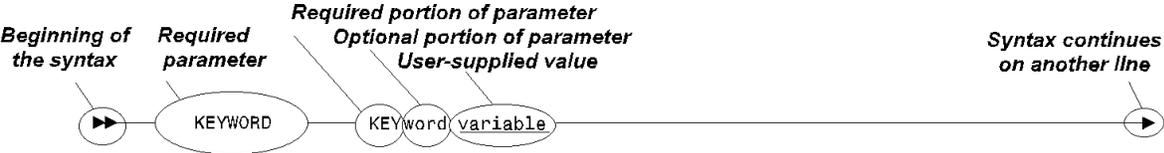
- *CA-ADS Quick Reference*
- *CA-ADS User Guide*
- *CA-IDMS Mapping Facility*
- *CA-IDMS Mapping Facility quick Reference*
- *IDD DDDL Reference*
- *CA-ADS DSECT Reference*
- *CA-IDMS SQL Programming*
- *CA-ADS Batch User Guide*
- *CA-IDMS Security Administration*
- *CA-IDMS System Generation*

Understanding Syntax Diagrams

Look at the list of notation conventions below to see how syntax is presented in this manual. The example following the list shows how the conventions are used.

UPPERCASE OR SPECIAL CHARACTERS	Represents a required keyword, partial keyword, character, or symbol that must be entered completely as shown.
lowercase	Represents an optional keyword or partial keyword that, if used, must be entered completely as shown.
<u>underlined lowercase</u>	Represents a value that you supply.
←	Points to the default in a list of choices.
lowercase bold	Represents a portion of the syntax shown in greater detail at the end of the syntax or elsewhere in the document.
▶▶	Shows the beginning of a complete piece of syntax.
◀◀	Shows the end of a complete piece of syntax.
▶	Shows that the syntax continues on the next line.
▶	Shows that the syntax continues on this line.
▶	Shows that the parameter continues on the next line.
▶	Shows that a parameter continues on this line.
▶ parameter ▶	Shows a required parameter.
▶ parameter parameter ▶	Shows a choice of required parameters. You must select one.
▶ parameter ▶	Shows an optional parameter.
▶ parameter parameter ▶	Shows a choice of optional parameters. Select one or none.
▶ parameter ▶	Shows that you can repeat the parameter or specify more than one parameter.
▶ parameter , parameter ▶	Shows that you must enter a comma between repetitions of the parameter.

Sample Syntax Diagram



Volume 1. CA-ADS Reference

Chapter 1. Introduction to CA-ADS

- 1.1 What is CA-ADS? 1-3
- 1.2 What CA-ADS does 1-4
- 1.3 Creating a CA-ADS application 1-5
- 1.4 Tools used to develop an application 1-7
 - 1.4.1 The CA-ADS application compiler (ADSA) 1-7
 - 1.4.2 Mapping facilities (MAPC and the Batch Compiler/Utility) 1-9
 - 1.4.3 CA-ADS dialog compilers (ADSC and ADSOBCOM) 1-10
 - 1.4.4 IDD menu facility and online IDD 1-12
 - 1.4.5 The CA-ADS runtime system 1-13
- 1.5 CA-ADS screens 1-14
 - 1.5.1 Action bar 1-15
 - 1.5.2 Action bar actions 1-17
- 1.6 Checkout and release procedures 1-28
 - 1.6.1 How to check out or release an entity 1-28
 - 1.6.2 Listing checkouts (ADSL) 1-30
 - 1.6.3 Modifying checkouts (ADSM) 1-31
- 1.7 CA-ADS help facility 1-32

1.1 What is CA-ADS?

The Application Development System (CA-ADS) is a tool used to expedite the writing and testing of modular applications. Activities such as flow-of-control processing, data storage definition, data verification, editing, error handling, terminal input and output, menu creation, and menu display are specified by using a series of screens instead of conventional detailed code.

CA-ADS can be used to develop online or batch applications. The following overview provides general information about each environment. Detailed information about CA-ADS facilities is contained in the subsequent sections of this manual.

1.2 What CA-ADS does

Develop a prototype Using a series of CA-ADS online development tools, you can create an early version of an application without writing any code. In this way, the structure of the online interactions and screen displays are available for review and modification before coding occurs.

Process logic and other enhancements can be added to the application prototype at any time. Process logic includes:

- Modules written in traditional programming languages
- Modules developed by using the Automatic System Facility (ASF)
- Modules already created with CA-ADS

Process and retrieve data: You can manipulate data from:

- A CA-IDMS/DB database
- Online entries
- VSAM data sets defined to the subschema
- External sequential files (for ADS/Batch only)

Edit input records: Input records can be automatically edited and verified using the editing and error-handling facilities available to CA-ADS applications.

Batch applications also use suspense files to store erroneous input records found at runtime. Suspense file records can be corrected and resubmitted at a later time.

Define and update multiple application components: Using the batch facilities of CA-ADS, updates to multiple application components, such as record definitions, can be accomplished at one time.

System utilities and facilities: System utilities and facilities allow application developers to:

- Transfer between CA-ADS development tools
- Debug applications
- Monitor runtime performance and resource usage

Options include:

- Archiving or printing log file information
- Obtaining reports that document CA-ADS applications and their components

1.3 Creating a CA-ADS application

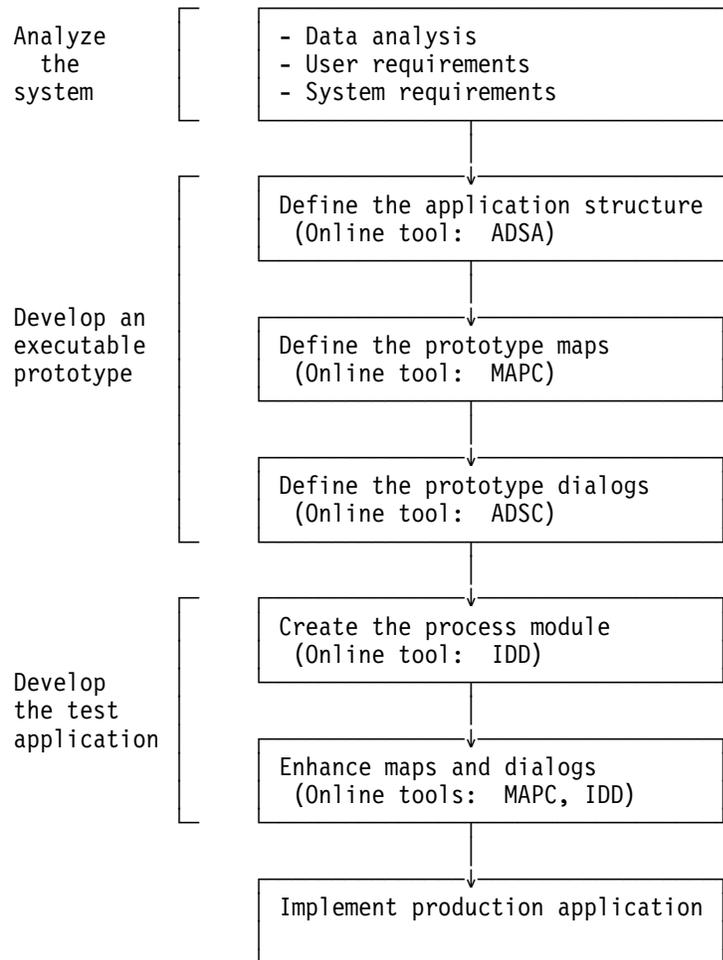
A CA-ADS application is based on an analysis of data and user requirements. This analysis forms the basis for determining the processing and the flow of control between processing activities required by the application. Once the blueprint, or design, of the application is created, the components of the application are defined and created using screen-driven development tools.

Procedure: Online application components can be developed in any order. However, the following sequence is typically used:

1. Develop an application structure diagram based on user responses and the paths between those responses.
2. Develop an application prototype by:
 - Defining the flow of control between processing activities
 - Defining the screens that the application uses to communicate with the end user
 - Defining the dialogs that represent application transactions and relate the screens to the application structure
3. Execute the application prototype.
4. Modify the application prototype, as needed.
5. Add process logic that performs the custom processing required by each dialog in the application.
6. Execute and test the application.
7. Put the approved application into production use.

The diagram below shows the steps and the online tools used for creating an online application. The application can be executed throughout the application development cycle. The online tools are discussed later in this section.

Typical steps when creating a CA-ADS application



1.4 Tools used to develop an application

The following online tools are used to develop CA-ADS applications:

- **The CA-ADS application compiler (ADSA)** — Defines the executable application structure
- **The CA-IDMS mapping facility (MAPC)** — Defines maps that establish preformatted screens for online processing
- **The CA-ADS dialog compiler (ADSC)** — Defines dialogs that consist of map, subschema, and process-module definitions
- **The Integrated Data Dictionary (IDD)** — Creates data definitions, edit and code tables, modules of process code, and declaration modules
- **The runtime system** — Executes CA-ADS applications at any stage in the applications' life cycle
- **The transfer control facility (TCF)** — Allows the application developer to transfer control between the online tools at definition time

Each development tool, except for the transfer control facility, is briefly discussed below. Detailed information about the development tools is presented later in this manual.

►► Details about the transfer control facility are contained in *CA-IDMS Transfer Control Facility*.

1.4.1 The CA-ADS application compiler (ADSA)

ADSA screens prompt for information that defines the application structure and runtime flow of control. When the definition is completed and compiled, CA-ADS stores the resulting load module in the data dictionary for use at runtime.

Functions: Runtime flow of control is based on the analysis of the interactions (**functions**) necessary to conduct the work of the application. In a CA-ADS application, a function can be any one of the function types listed in the table below. Functions are the structural units of an application. They are defined by using ADSA screens.

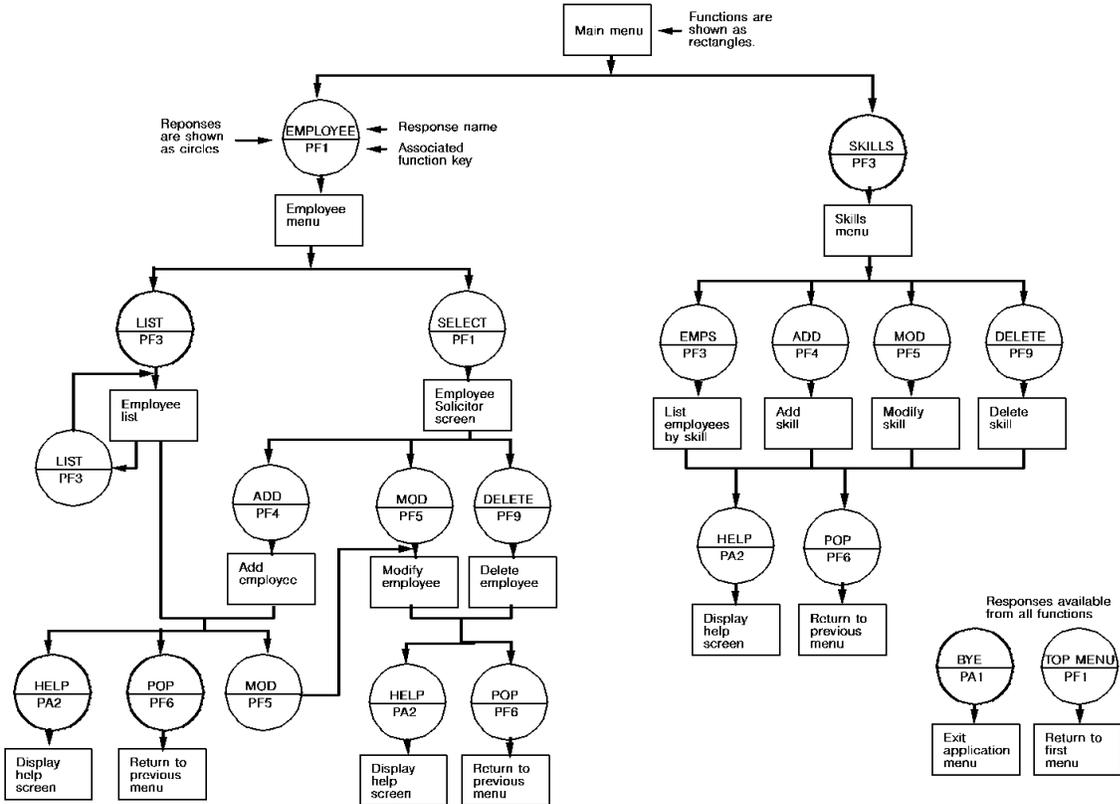
Typically, an online application contains menu functions, menu/dialog functions, dialog functions, and many of the system functions. Program functions are less often used.

Functions in a CA-ADS application

Function Type	What it Does
Dialog	Performs a variety of processing activities, such as data retrieval and update
Program	Performs processing specified in user-written COBOL, PL/I, or Assembler programs
Menu	Displays a system-defined menu screen Performs standard menu processing activities at runtime
Menu/dialog	Displays either a system-defined or a site-defined menu screen Performs standard processing and any additional site-defined processing supplied by an associated dialog
System Functions	Perform predefined activities
ESCAPE	Bypasses a function even though the current screen contains errors
FORWARD/BACKWARD	Pages forward or backward on menu maps
HELP	Displays the runtime Application Help screen
POP	Returns to the last menu or menu/dialog function
POPTOP	Returns to the first menu or the menu/dialog function
QUIT	Terminates application processing
RETURN	Returns to the next higher level function in the sequence of operative functions
SIGNON/SIGNOFF	Signs on to or off from CA-IDMS/DC or CA-IDMS/UCF from within the application
TOP	Returns to the highest level function in the sequence of operative functions

Responses: The path between two functions is called a **response**. Responses define all possible flow of control in the application. The following diagram shows the functions and responses of a sample employee information application that stores and displays employee information.

Functions and responses in a sample CA-ADS application



1.4.2 Mapping facilities (MAPC and the Batch Compiler/Utility)

Online: CA-IDMS mapping facility (MAPC) screens prompt for specifications that define the screen format (**map**) for a CA-ADS application. Data editing, data conversion, and error-handling criteria can also be specified. The specified criteria are automatically applied to data processed by the map at runtime.

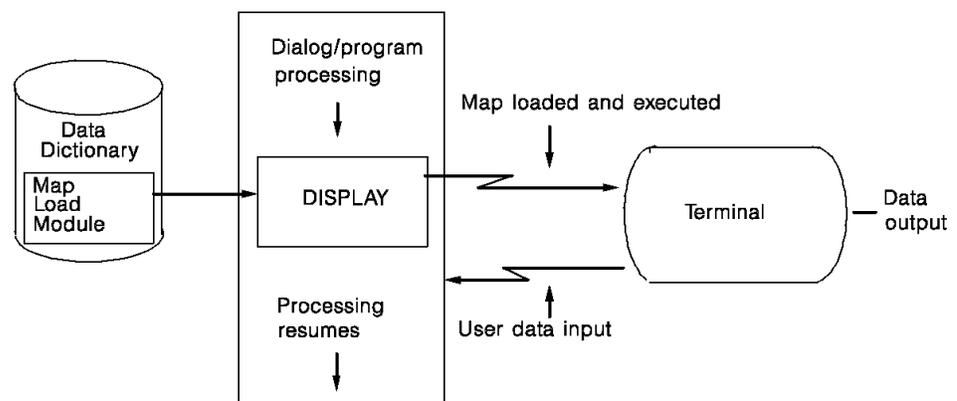
Batch: Alternatively, the batch compiler and utility allow the developer to define and compile maps in batch definition mode. These batch tools are particularly useful when several maps require modification and recompilation.

►► More information about the online mapping facility and the batch compiler and utility can be found in *CA-IDMS Mapping Facility*.

Defining the screen format: A map in an online application defines the screen format displayed to an end user at runtime. The fields displayed on the screen allow the end user to enter or modify data. The data is then processed according to the instructions contained in the processing logic of the dialog.

The diagram below shows the sequence followed when a map is displayed at runtime. The DISPLAY statement in the processing logic accesses the map load module that is stored in the data dictionary, causing the map to be displayed on the screen. Data entered on the screen is then processed according to the instructions contained in the dialog processing code.

Runtime display screen defined by online map



1.4.3 CA-ADS dialog compilers (ADSC and ADSOBCOM)

Dialog: ADSC brings various application components together into a modular entity (**dialog**) that is executed at runtime. The table below lists the components of a dialog and describes what each component does. ADSC screens prompt for names of dialog components and other information needed to define the dialog for an online or batch application.

ADSOBCOM is a utility that can be used to define and recompile several dialogs in batch mode. This capability is particularly useful when dialogs need to be recompiled because maps, processes, subschemas, or records associated with several dialogs are modified.

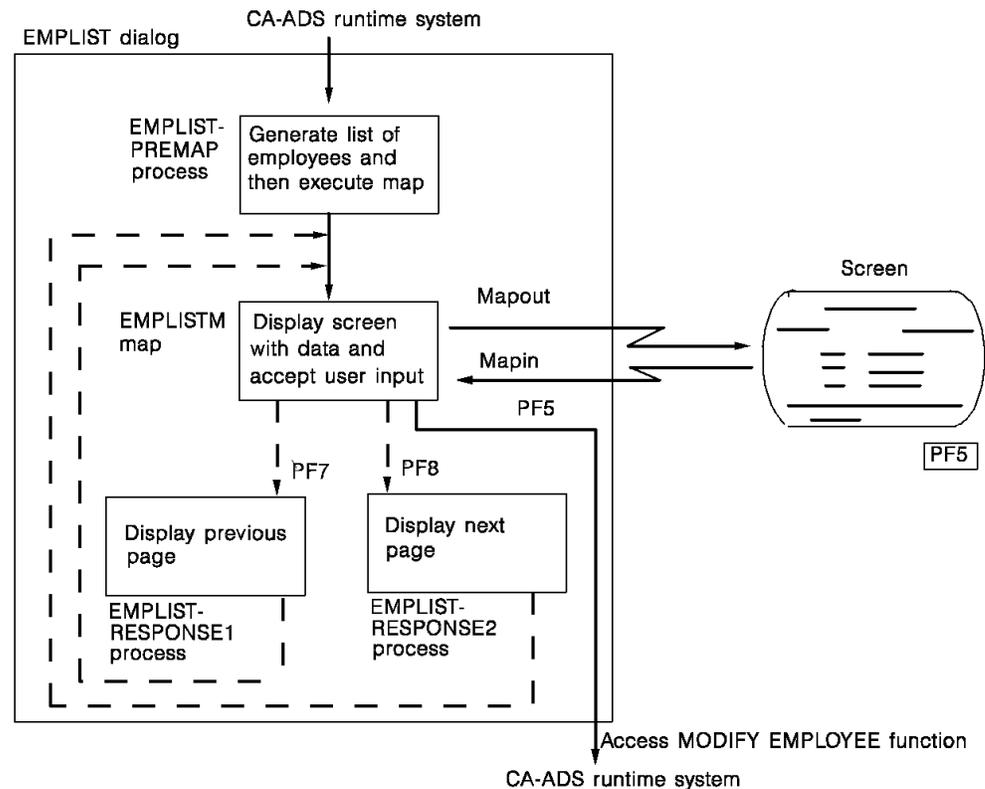
Components of a dialog

Dialog component	What it does
Map	Provides the means of communication between one data source and the application
Subschema	Provides the dialog's view of the database
Access module	Provides optimized access to an SQL-defined database
Records	Describe the data used by the dialog and map
Subschema records	Allow the dialog to read and write information to the database
Dialog work records	Provide temporary storage to be used by dialogs and maps
Process modules	Define the processing the dialog performs at runtime
Premap process (optional, maximum of one per dialog)	Defines processing that prepares the screen for display
Response process (optional, any number per dialog)	Defines processing that occurs after the end user presses a control key (such as Enter or PF1) in response to the dialog's map
Declaration module (optional, maximum of one per dialog)	Specifies SQL cursor and WHENEVER declarations (for SQL error processing) ▶▶ See <i>CA-IDMS SQL Programming</i> .

In an online application, a dialog interacts with the end user by displaying a screen and allowing the user to view and input information.

Interaction between dialog and end user at runtime: The diagram below shows the interaction between the dialog and the end user at runtime. In the EMPLIST dialog, the premap process generates a list of employee names. The screen defined by the EMPLISTM map displays a page of names. The user can respond by paging backward or forward by pressing PF7 or PF8. Pressing PF5 accesses the MODIFY EMPLOYEE function.

CA-ADS dialog at run time



1.4.4 IDD menu facility and online IDD

The CA Integrated Data Dictionary (IDD) consists of two related online tools, the IDD menu facility and online IDD. These tools are used to define data and various CA-ADS application components to the data dictionary.

IDD menu facility screens prompt for all required specifications. Online IDD allows developers to use Data Dictionary Definition Language (DDDL) statements to define and modify data dictionary entities.

- ▶▶ For information about how to use the IDD menu facility and online IDD screens, see *CA-IDMS Online Compiler Text Editor*.
- ▶▶ For details about online IDD and DDDL statements, see the *IDD DDDL Reference*.

1.4.5 The CA-ADS runtime system

CA-IDMS/DC and CA-IDMS/UCF: The CA-ADS runtime system is a CA-IDMS/DC or CA-IDMS/UCF (DC/UCF) task that establishes the application environment and executes the application components as a series of tasks. Operations such as building and displaying menus, allocating buffers, initializing data, editing data, and validating data are automatically performed by the runtime system.

1.5 CA-ADS screens

CUA-style screens: The dialog (ADSC), map (MAPC), and application (ADSA) compilers provide **Common User Access (CUA)** style screens. These screens provide space for the developer to enter data particular to the dialog, map, or application. There are key assignments at the bottom of each screen and CUA-style selection by means of numbers or the "/" character. Screens are consistent across tools with standard:

- Screen layout
- Terminology
- Commands
- Functions
- Key assignments

The initial screen of each compiler is made up of six areas. These areas are shown on the screen below and described in the following table.

```

Add Modify Compile Delete Display Switch
-----
                                CA-ADS Online Dialog Compiler
                                Computer Associates International, Inc.
-
Dialog name . . . . . _____
Dialog version . . . . . _____
Dictionary name . . . . . _____
Dictionary node . . . . . _____
-
Screen . . . . . 1 1. General options
                   2. Assign maps
                   3. Assign database
                   4. Assign records and tables
                   5. Assign process modules
-
                                Copyright (C) 1972,2000 Computer Associates International, Inc.

Command ===>
Enter F1=Help F3=Exit F10=Action

```

Areas of the screen

Area	Description
Activity selection area	Contains an action bar that identifies the actions that can be taken on the entity and provides pull-down windows to implement these actions.
Identification area	Allows entry of information that uniquely identifies the entity being worked on: name, version, dictionary name, and dictionary node. The dictionary name and node information default to the values established for the current terminal session.
Screen specification area	Presents entity definition steps and provides space for the user to request a specific step.
Message area	Presents informational, warning, or error messages.
Command area	Allows entry of action bar commands to pull down a window. The action bar command can be abbreviated to three characters. In the case of the SWITCH command, both the command and the desired task can be entered on the command line, thereby bypassing the window (for example, SWI OLQ).
Key assignment area	Presents the valid key choices and the action taken.

1.5.1 Action bar

The activity selection area of each Main Menu screen is composed of an action bar containing six actions. Each action on the action bar is associated with a pulldown window.

Accessing the action bar: You access an item on the action bar in the activity selection area in one of three ways:

- Tab to the item and press [Enter]
- Press [PF10] to move to the action bar and then tab to the item and press [Enter]
- Type the name of the action on the command line and press [Enter]

Any of these actions results in a pulldown window being opened.

```

Add Modify Compile Delete Display Switch
-----
Copy from dialog      A-ADS Online Dialog Compiler
Name
Version 1            ter Associates International, Inc.
-----
F3=Exit
-----
Dialog name . . . . . JPKD1
Dialog version . . . . . 1
Dictionary name . . . . . TSTDICT
Dictionary node . . . . .
Screen . . . . . 1 1. General options
                    2. Assign maps
                    3. Assign database
                    4. Assign records and tables
                    5. Assign process modules

Command ==>
Enter F1=Help F3=Exit F10=Action

```

Pulldown windows: There are six pulldown windows available from the action bar on the Main Menu screen.

Use this window...	To...
Add	Check the entity out to the current developer and (optionally) copy the definition of a currently existing entity.
Modify	Check the entity out to the current developer or release the entity. List the checked-out entities.
Delete	Delete either the current changes (since the last compilation) or the entire entity. A confirmation window is opened if the option is to delete the entire entity.
Compile	Store the definition in the data dictionary, create a load module, and present errors.
Display	Display summary information (entity size, when built, user-id, etc.). Browse the entity. View the runtime image of the entity (for maps only).
Switch	Use the transfer control facility (TCF) to transfer control to another CA-IDMS/DC task (such as IDD, OLQ, etc.).

Each of the actions on the action bar is described below.

Leaving the window: Most actions listed in the pulldown windows require that the developer enter a menu choice or data, and press [Enter].

To leave a pulldown window without entering a number or data, the developer presses [PF3]

1.5.2 Action bar actions

A description of each of the action items identified in the above table follows.

Add: Specifies that a new entity is being added.

Using the **Add** action, the developer can copy an existing entity into the current work file and give it a new name.

When the application developer specifies an entity name, the compiler ensures that no entity exists with the specified name and version number and returns the message:

```
DC498104 DIALOG1 was not found, use the Add action to create or copy the dialog
```

To request the add operation, the developer must either open the **Add** window by moving the cursor to the action bar, or type the word **ADD** on the command line.

Note: The compiler does **not** assume that the add operation is requested when it does not find the entity in the dictionary.

If an entity with the specified name and version number exists, the compiler assumes a modify operation and returns the following message:

```
DC498102 Currency set for dialog empdemo version 1
```

When you add an entity, you have explicitly checked it out, and no one else can access it until you check it in.

►► For more information about checkouts and checkins, see 1.6, “Checkout and release procedures” later in this chapter.

If another developer owns the entity, or if another entity type has already used the name, the following error message is issued:

```
DC498103 Currency not established. Dialog is currently checked out  
DC498107 to user MET on dictionary TESTDCT.
```

The screen below shows the **Add** window on the dialog compiler Main Menu.

```

Add  Modify  Compile  Delete  Display  Switch
-----
Copy from dialog      A-ADS Online Dialog Compiler
Name      JPKD5_____
Version   1_____  ter Associates International, Inc.
-----
F3=Exit
-----
Dialog name . . . . . JPKD1
Dialog version . . . . . 1
Dictionary name . . . . . TSTDICT
Dictionary node . . . . . _____

Screen . . . . . 1  1. General options
                   2. Assign maps
                   3. Assign database
                   4. Assign records and tables
                   5. Assign process modules

Dialog added using copy request.

Command ==>
Enter  F1=Help  F3=Exit  F10=Action

```

Modify: Specifies that an existing entity is being modified or, if the specified entity belongs to a suspended session, that the suspended session is being resumed.

You can resume a session by filling out the entity name. MODIFY CHECKOUT is the name CA-ADS gives to entities not yet checked out.

►► For more information about checkouts and checkins, see 1.6, “Checkout and release procedures” later in this chapter.

When the application developer specifies the **Modify** action, the compiler ensures that an entity exists with the specified name and version number and returns the message:

```
DC498102 Currency set for dialog empdemo version 1
```

If the specified entity exists, the compiler retrieves and displays the definition. When the **Compile** action is selected, a new load module is created for the entity.

If an entity with the specified name and version number does not exist, **Modify** is invalid.

If an entity is currently in use, the name of the owner is displayed. The owner can release the entity, if desired, without having to compile it and without deleting the current changes in the work file. To release an entity, the developer chooses item 2, **Release**, from the **Modify** pulldown window. Another developer can then assume control of the entity by issuing a reserve request.

To see a list of the entities checked out, the developer chooses item 3, **List Checkouts**.

Modify is the default for an existing dialog.

The screen below shows the **Modify** window on the dialog compiler Main Menu.

```

Add  Modify  Compile  Delete  Display  Switch
-----
      1  1. Checkout      Online Dialog Compiler
        2. Release
        3. List Checkouts ssociates International, Inc.
-----
      F3=Exit
-----
Dialog name . . . . . JPKD1
Dialog version . . . . . 1
Dictionary name . . . . . TSTDICT
Dictionary node . . . . .
-----
Screen . . . . . 1  1. General options
                   2. Assign maps
                   3. Assign database
                   4. Assign records and tables
                   5. Assign process modules

Command ==>
Enter F1=Help F3=Exit F10=Action

```

Compile: Specifies that the current entity is being compiled.

When the application developer specifies the **Compile** action, the compiler ensures that an entity exists with the specified name and version number.

If the specified entity exists, the compiler compiles the entity (including all process modules in the case of a dialog) and, if the compilation is successful, creates a load module. The load module is stored in the data dictionary.

Upon compilation, the compiler deletes any queue records saved for a suspended session of the entity definition.

If an entity with the specified name and version number does not exist, the **Compile** action is invalid and an error message is displayed.

If errors are encountered during the compilation process, they are written to a log file that allows scrolling access and is chosen from the **Compile** window.

The screen below shows the **Compile** action on the dialog compiler Main Menu.

```

Add Modify Compile Delete Display Switch
.-----
      1 1. Compile          log Compiler
      2. Display messages  nternational, Inc.
-----
      F3=Exit

Dialog name . . . . . METDLG1
Dialog version . . . . . 1
Dictionary name . . . . . TSTDICT
Dictionary node . . . . . _____

Screen . . . . . 1 1. General options
                  2. Assign maps
                  3. Assign database
                  4. Assign records and tables
                  5. Assign process modules

Command ==>
Enter F1=Help F3=Exit F10=Action

```

When errors are encountered in the compilation process, the application developer chooses item 2, **Display messages**, from the **Compile** action on the action bar. Messages on the Messages screen indicate where there are errors.

The screen below shows the Messages screen resulting from dialog compilation:

Sample Messages screen

```

                                Compiled Process Modules          Page  1 of  1
                                Dialog METDLG01 Ver  1

Name MET-ERROR
Version 0001 Type 2
Key _____ Value
Name _____
Version _____ Type _
Key _____ Value
Name _____
Version _____ Type _
Key _____ Value
Name _____
Version _____ Type _
Key _____ Value
Type: 1=Declaration 2=Premap 3=Response 4=Default Response
Select a process for Display or Print.

F1=Help F3=Exit F7=Bkwd F8=Fwd F11=Dialog-level messages
4B7 A
                                IBM 07/62

```

The Messages screen displays the source statements for a premap or response process that contains errors in its process code. It also contains other messages encountered during the compilation of the dialog.

Enter 1, **Display**, to display a copy of the dialog process source errors.

Sample Dialog Process Source screen

```

                                Dialog Process Source                Page 1 of 1
-----
.<PROCESS> MET-ERROR                                0001
  100 MOBE ZNTRAIL(MYNUMBER) TO WK-PART-CODE
    $
  <E> DC157001 INVALID INITIATING KEYWORD FOR COMMAND. STMT FLUSHED.
    200 DISPLAY.

-----
Module currently displayed: MET-ERROR                                VERSION: 1
F3=Exit F5=IDD F7=Bkwd F8=Fwd F11=Next.error

```

Data dictionary sequence numbers appear to the left of the source statements. If the listed code includes another process module, the source statements from the included module are listed after the INCLUDE statement.

►► The INCLUDE command is described in Chapter 12, “Introduction to Process Commands.”

Process statements that are in error are flagged with a dollar sign (\$), followed by a CA-ADS error code and message. One erroneous process source statement can cause subsequent statements to be found in error, even if they are coded correctly.

►► For a complete listing of the error messages used by CA-ADS, refer to *CA-IDMS Messages and Codes*.

The Messages screen cannot be used to correct errors in the process source code. To correct stored process code, the application developer must use IDD. To toggle to IDD press [PF5].

Note: In order to toggle to IDD, you must be running ADSC under the CA-IDMS Command Facility.

►► For more information about the CA-IDMS Command Facility, refer to the *CA-IDMS Command Facility*.

►► For more information on correcting errors in process code, refer to the *CA-ADS User Guide*.

Delete: Specifies that an existing entity or changes to an existing entity be deleted.

When the application developer specifies the **Delete dialog** action from this window, the compiler ensures that an entity exists with the specified name and version number.

If the specified entity exists and the action is **Delete dialog**, a confirmation window is presented to the user, allowing the request to be confirmed or rescinded. If the deletion is confirmed, the compiler deletes the load module from the data dictionary, any dictionary definitions, and any queue records saved for a suspended session of the definition.

If an entity with the specified name and version number does not exist, the **Delete dialog** action is invalid and an error message is displayed.

The entity must not be reserved to another user if it is to be deleted.

If **Delete changes** has been chosen from this window, the working file is reconstructed from the most recently stored (compiled) definition.

If **Delete changes** is chosen, the entity remains checked out to the current developer.

The screen below shows the **Delete** window on the dialog compiler Main Menu. A confirmation window is displayed so that the request to delete the dialog can be confirmed or rescinded.

```

Add Modify Compile Delete Display Switch
.-----
                2 1. Delete changes  piler
                2. Delete dialog
                ----- io .-----
                F3=Exit
                ----- .      2 1. Confirm
                                2. Reject
Dialog name . . . . . SOME1
Dialog version . . . . . 1
Dictionary name . . . . . TSTDICT
Dictionary node . . . . . _____

Screen . . . . . 1 1. General options
                   2. Assign maps
                   3. Assign database
                   4. Assign records and tables
                   5. Assign process modules

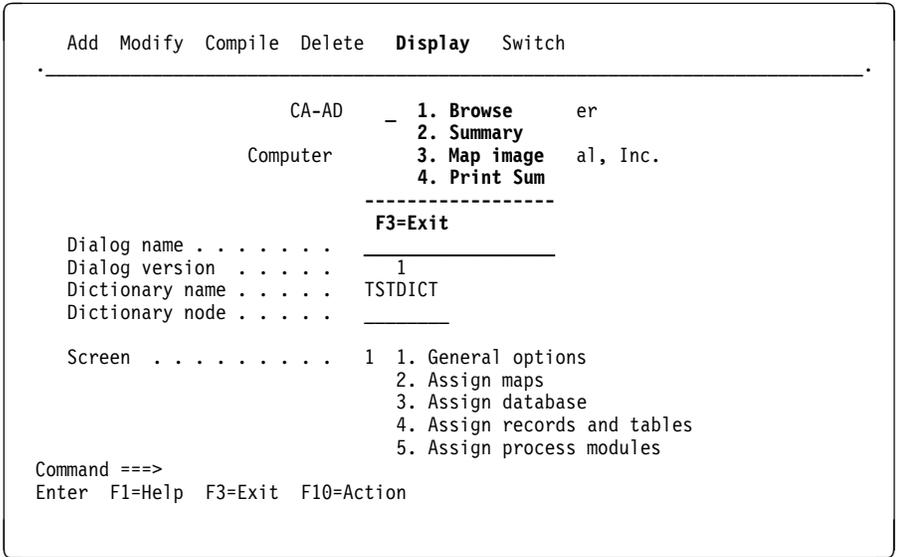
Enter 1 to confirm the delete request.

Command ==>
Enter F1=Help F3=Exit F10=Action

```

Display: Specifies that summary information for the named entity be displayed.

The screen below shows the **Display** window on the dialog compiler Main Menu.



From the **Display** window, the developer can choose **Browse** or **Summary**. Other options available from this window depend on the compiler being used.

Browse: The **Browse** option allows the developer to walk through the entity definition process without changing any information about the entity. All fields on the compiler screens are protected.

If the entity is currently checked out to another developer and changes have been made but not saved, the **Browse** option returns the entity definition as it exists in the dictionary.

Summary: The **Summary** option gives an overview of the entity definition.

The screen below shows the **Summary** option taken from the dialog compiler Main Menu.

```

                                Dialog Summary Display                                Page 1 of 1
-----
DIALOG: PROCEMP  VERSION: 1

Entry Point.....: PREMAP                      Mainline.....: YES
Symbol Table.....: YES                         Diagnostic Tables: NO
Cobol Moves.....: NO                          Retrieval Lock...: NO
Autostatus.....: YES                          Message Prefix...: DC

Status Rec: ADSO-STAT-DEF-REC                  Version: 1
Access Module: JMASQLD  ANSI-flag:  Date Format:  Time Format:

Online Map: MAP1                               Version: 1

Record: SQLCA                                 Version: 1  NC

Process: USER2-PM                             Version: 1  Premap
Process: USER1-CONTINUE                       Version: 1  Response
Execute on error: NO  Key: ENTER  Value:

-----
F3=Exit  F7=Bkwd  F8=Fwd

```

Map image: The Map Image screen displays the dialog map as it appears to the user at runtime. The application developer can position the cursor at map data fields and enter information.

Map modifications defined in process code associated with the dialog are not in effect when the screen is displayed. Map data fields do not contain any values on the Map Image screen.

Hit any key to return to the menu screen.

Print Sum: The **Print Sum** option prints a copy of the Print Summary screen.

Switch: Specifies that control is to be passed to another CA-IDMS/DC or CA-IDMS/UCF task. **Switch** suspends the current dialog compilation session and transfers control to another DC/UCF task, to the transfer control facility Selection screen, or to a new or suspended session of another task.

A task code must be entered into the **Task code** field.

The screen below shows the **Switch** window on the dialog compiler Main Menu.

```

Add Modify Compile Delete Display Switch
.-----
                CA-ADS Online  Task code _____
                Computer Associate F3=Exit
                -----

Dialog name . . . . . JPKD1
Dialog version . . . . . 1
Dictionary name . . . . . TSTDICT
Dictionary node . . . . . _____

Screen . . . . . 1 1. General options
                   2. Assign maps
                   3. Assign database
                   4. Assign records and tables
                   5. Assign process modules

Command ==>
Enter F1=Help F3=Exit F10=Action

```

1.6 Checkout and release procedures

The checkout and release procedures allow a developer to *own* an application, dialog, or map while working on it.

The developer checks out an entity to work on it. Until the entity is released attempts by another developer to work on that entity result in the following message:

If ADSC:

DC498103 Currency not established. Dialog is currently checked out DC498107 to user MET on dictionary TESTDCT.

If ADSOBCOM:

DC497042 Dialog is checked out to ADSC and cannot be compiled in batch

1.6.1 How to check out or release an entity

Types of checkouts and releases: An application developer checks out and releases entities *explicitly* or *implicitly*.

Explicit checkouts: Explicit checkouts allow the developer to control and retain scope of an entity across repeated definition sessions and entry compilations.

An *explicit* checkout begins with either:

- An ADD action from the pulldown menu.
- A CHECKOUT action from the MODIFY pulldown menu. The named entity must exist in the dictionary.

When an entity has been *explicitly* checked out, no other developer can work on an existing entity until **Release** is specified. If **Release** is not specified, all other developers are limited to the **Display** and **Switch** actions.

Explicit checkouts end with either:

- A RELEASE action from the MODIFY pulldown menu
- A successful DELETE action

Explicit releases: This action checks the named entity in and releases it for use by another developer.

An explicit release occurs when:

- The user selects the RELEASE option from the MODIFY sub menu
- The user selects the DELETE DIALOG option from the DELETE sub menu

Implicit checkout: Implicit checkout is intended to facilitate a developer's work when a long scope of retention is not required. You use implicit checkout instead of explicit checkout when rapid deletion, compilation, or simple modification of one or many entities is required.

An *implicit* checkout begins with any of the following:

- A COMPILE action
- A DELETE action
- Entering the number of a screen in the **Screen field** and pressing [Enter] from the main menu screen

Note: If COMPILE or DELETE is successful, the dialog or application is automatically released. If unsuccessful, the application or dialog remains checked out to the developer.

When the entity has been *implicitly* checked out, checkin occurs *automatically* after the entity successfully compiles.

Implicit releases: Implicit releases end an implicit checkout and occur when the application developer does one of the following:

- Successfully compiles the entity
- Selects **Modify** from the action bar and chooses the **Checkout** option from the pulldown window
- Selects **Delete** from the action bar and chooses either the **Delete changes** or **Delete dialog** option from the pulldown window

This action checks the named entity in and releases it for use by another developer.

Releasing an entity: The application developer releases an entity by selecting **Modify** from the action bar and choosing the **Release** option from the pulldown window. The named entity must be checked out to the developer before that developer can release it.

The release action suspends the current session and allows another developer to check out the entity.

If the developer has made no changes to the entity definition, the queue records for the current session are deleted and the developer checking out the released entity receives the following message:

```
DC498102 Currency set for dialog empdemo version 1
```

If the developer has made changes to the entity definition and the entity has been released, the queue records are retained. Another developer can check the entity out and receive the following message:

```
DC498106 Dialog empdemo version 1 is recovered from a suspended session
```

1.6.2 Listing checkouts (ADSL)

Within each compiler, the entities checked out to the executing user can be viewed.

The ADSL transaction allows a user to view the entities of any type checked out by any user. To invoke ADSL, enter the task code **ADSL** and enter the desired tool and user information. In this example, 'ADSC' and 'ALL USERS' have been selected:

```

RELEASE 15.0                                CAGJF0
                                CA-ADS AND MAPPING CHECKOUT LISTS

                                TOOL . . 1    1. ADSC
                                                2. ADSA
                                                3. MAPC
                                                4. ALL

                                USER . . _____

                                ALL USERS /    (/)

ENTER  F1=HELP  F3=EXIT

```

The checkout listing information you request is then displayed:

```

Dialog compiler                Checkout Listing                Page 1 of 1
Dialog:
--Name--  -Version-  -----User Id-----  Dictionary:
NEWDIAL2   1          PAGT001                  APPLDICT
NEWDIAL3   1          PAGT001                  APPLDICT
NEWDIAL1   1          PAGT001
ADDS01D   1          PAGT001

F3=Exit  F7=Bwd  F8=Fwd

```

The user can modify checkouts using ADSM.

1.6.3 Modifying checkouts (ADSM)

What you can do: The ADSM transaction can be used to delete or modify the assignment of suspended compiler sessions. For example, a project leader can use ADSM to reassign ongoing work:

```

Release 15.0                                CAGJF0
                                CA-ADS and MAPPING Checkout Modification

Action . . . . . 2          1. Delete
                                2. Reassign

Tool . . . . . 1           1. ADSC
                                2. ADSA
                                3. MAPC

Entity name . . . . . ADDS01D

Entity version . . . . . 1

Current user . . . . . PAGT001

Reassign to user . . . . . EMMWI02

                                Copyright (C) 1972,2000 Computer Associates International, Inc.
Enter  F1=Help  F3=Exit

```

How it works: When a checkout is **reassigned**, the queue for the tool session remains in place, including uncompiled changes, but the user assignment is modified.

When a checkout is **deleted**, uncompiled changes are deleted.

Releasing the entity: You can release an entity through ADSM by leaving blank the field for the new user. The queue for the entity is maintained, and the entity is available for checkout by another user.

1.7 CA-ADS help facility

CA-ADS provides context-sensitive online help when working with CA-ADS compilers and the mapping compiler. Help is available at both the map level and the field level.

Map-level help: Map-level help provides information on the purpose of the specific map and the general type of information required for the map.

Field-level help: Field-level help provides information on data required for a specific field on the map.

Using help

Use...	To...
PF1	Request help from any screen, depending on cursor position
PF3	Return from the help screen
PF7/PF8	Page backward and forward while on the help screen

Accessing help: Depending on the cursor position, either map or field help is accessed as follows:

If the cursor is positioned on...	The following will be displayed...
A map field associated with help text	The map field help text
A map field not associated with help text	The map help text
Anywhere else on the screen	The map help text

Help text for the map is displayed as full screen.

Help text for the map field is displayed as half screen covering either the top or bottom half of the screen as appropriate.

Sample help screen: The screen below shows map field help for the **Mainline** map field on the Dialog and Options screen of the dialog compiler.

```

:-----:
:      Specify MAINLINE if the dialog will be invoked from the      :
:      CA-IDMS/DC prompt or by an APPC (send-receive option) request. :
:      :
:      Mainline dialogs are potentially eligible to appear on the ADS :
:      MENU screen. :
:      :
:      :
:      :
:      :
:-----:
Return F3      Page F7/F8      Scroll: 010
Options and directives . . . . . - Mainline dialog
                               - Symbol table is enabled
                               / Diagnostic table is enabled
                               / Entry point is premap
                               COBOL moves are enabled
                               / Activity logging
                               / Retrieval locks are kept
                               / Autostatus is enabled

-----
Enter F1=Help F3=Exit F4=PrevStep F5=NextStep

```


Chapter 2. CA-ADS Application Compiler (ADSA)

- 2.1 Overview 2-3
- 2.2 Application compiler session 2-4
 - 2.2.1 Invoking the application compiler 2-4
 - 2.2.2 Sequencing through application compiler screens 2-7
 - 2.2.3 Suspending a session 2-10
 - 2.2.4 Terminating a session 2-10
- 2.3 Application compiler screens 2-11
 - 2.3.1 Main menu 2-11
 - 2.3.2 General Options screen—Page 1 2-14
 - 2.3.3 General Options screen—Page 2 2-16
 - 2.3.4 Response/Function List screen 2-19
 - 2.3.5 Response Definition screen 2-23
 - 2.3.6 Function Definition (Dialog) screen 2-27
 - 2.3.7 Function Definition (Program) screen 2-30
 - 2.3.8 Function Definition (Menu) screen 2-32
 - 2.3.9 Global Records screen 2-37
 - 2.3.10 Task Codes screen 2-39

2.1 Overview

The CA-ADS application compiler is an application design and prototyping tool. During an application compiler session, the application developer defines the components and structure of an application. When the definition is complete, the application developer compiles the application. The resulting load module is stored in the data dictionary for use at runtime. When the load module for an application is compiled, the only definitions that must exist in the data dictionary are those global records specifically associated with the application. All other entities associated with the application can be created and added to the dictionary at any time before the application is executed. This feature allows the application developer to upgrade application components without having to recompile the application.

▶▶ For more information on application compiling features, refer to *CA-ADS Application Design Guide*.

▶▶ For examples of using the application compiler, refer to *CA-ADS User Guide*.

2.2 Application compiler session

In an application compiler session, screens are displayed that prompt the application developer for information about an application and the responses and functions associated with the application. The information supplied by the application developer is used by the CA-ADS runtime system to control the execution of the application.

2.2.1 Invoking the application compiler

The application developer can invoke the application compiler from any of the three ways described below.

From CA-IDMS/DC or CA-IDMS/UCF: By specifying the appropriate CA-IDMS/DC or CA-IDMS/UCF task code, the application developer can invoke the application compiler. Task codes are defined at system generation and can vary from site to site. The default task code for the application compiler is ADSA.

TCF: To use the application compiler under the transfer control facility (TCF), specify **ADSAT**, the TCF version of the application compiler task code. The TCF task code for the dialog compiler is **ADSCT** and for the mapping facility is **MAPCT**.

When invoked, the application compiler displays a blank Main Menu screen on which the application developer can begin a new session or resume a suspended session.

From another TCF task: By specifying the appropriate CA-IDMS/DC or CA-IDMS/UCF task code in conjunction with the SWITCH command from another task executing under the transfer control facility, the application developer can invoke the application compiler.

If a new session is requested, the application compiler displays a blank Main Menu screen on which the application developer can begin a new session or resume a suspended session.

If an old session is requested, the application compiler resumes its most recently suspended session under the transfer control facility.

From the TCF Selection screen: The application compiler can be invoked by keying any nonblank character, except the underscore (_), next to the appropriate task code or descriptor, as follows:

- Keying a nonblank character next to the appropriate task code invokes the application compiler. A blank Main Menu screen on which the application developer can begin a new session or resume a suspended session is displayed.
- Keying a nonblank character next to the descriptor of a suspended application compiler session invokes the application compiler and resumes the suspended session at the Main Menu screen. The descriptor consists of the appropriate task code, the application name, and the application version number.

The transfer control facility enables the application developer to transfer from one CA-IDMS/DC or CA-IDMS/UCF task to another. For example, the application developer can transfer between the application compiler, IDD, MAPC, and the dialog compiler. When control is transferred from a task, the current session of that task is suspended, if necessary. A task can have several suspended sessions.

►► For a detailed description of the transfer control facility, see *CA-IDMS Transfer Control Facility*.

Note:

In a multiple dictionary environment, be sure to begin the application compiler session in the correct dictionary. The dictionary name can be specified in the **Dictionary name** field on the Main Menu screen.

TCF Selection screen: Sample selections on the transfer control facility Selection screen are shown below:

```

                                COMPUTER ASSOCIATES INTERNATIONAL, INC.
                                TRANSFER CONTROL FACILITY                                *** SELECTION SCREEN ***

_  SUSPEND TCF SESSION  (PF9)                                DBNAME...:                                DBNODE...:
_  TERMINATE TCF SESSION (PF3)                                DICTNAME: TSTDICT                                DICTNODE:

                                *TCF TASKCODES*
SELECT ONE TO START A NEW SESSION
_  TCF
_  SYSGENT  SYSGEN COMPILER
_  MAPCT    MAP DEFINITION
_  ADSCT    DIALOG GENERATOR
X  ADSAT    APPLICATION GENERATOR
_  ASF
_  ASFT
_  IDDT     IDD COMMAND MODE
_  SSCT     SUBSCHEMA COMPILER
_  SCHEMAT  SCHEMA COMPILER
_  IDDMT    IDD MENU MODE
_  OLQ      OLQ COMMAND MODE
_  OLQT     OLQ COMMAND MODE

                                *SUSPENDED SESSIONS*
SELECT ONE TO RESUME AN OLD SESSION
                                TASKCODE      DESCRIPTOR
_  ADSCT    MPKDIA1 0001
_  ADSAT    MPKAPP1 001
_  ADSAT    MPKAPP1 002
_  ADSCT    MPKDIA2 0001

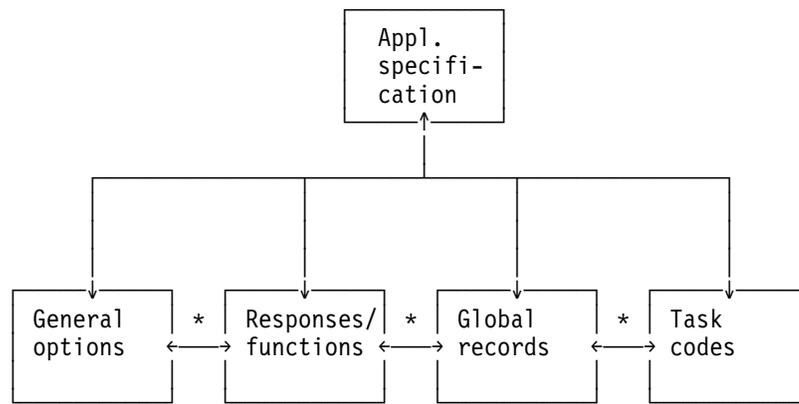
```

2.2.2 Sequencing through application compiler screens

Application compiler screens prompt the application developer for information about an application. The developer can sequence through the application definition steps or request a step in the process explicitly.

The primary steps involved in creating an application are shown below. The developer can either choose the next step from the Main Menu screen or move through the steps from screen to screen using [PF5].

Steps in creating an application



* Previous/next step (F4/F5)

Summary of application compiler process: Each step in the process of creating an application is associated with one or more screens as shown below.

Step in process	Screen	Purpose
Application specification	Main Menu	Identifies the name and characteristics of an application and specifies the action to be taken
General options	General Options	Specifies application options for date format, print options, security, and maximum number of responses
Response/function definition	Response/Function List	Specifies the relationship between functions and responses
	Response Definition	Specifies the name and characteristics of a response.
	Function Definition (Dialog)	Allows specification of a function and associated dialog and valid responses for the dialog or menu/dialog function currently being defined.
	Function Definition (Program)	Specifies the name and description of the associated program and records to be passed to a user program function.
	Function Definition (Menu)	Specifies characteristics for a function defined as a menu; allows alteration of the sequence or suppression of the display of responses on a menu screen.
Global records	Global Records	Specifies records available to all functions in an application
Task codes	Task Codes	Specifies DC/UCF task codes that initiate an application at runtime.

Control keys: While creating an application, the applications developer can use the control keys shown below to:

- Move from one step in the process to another step
- Move from one screen to another screen while remaining on one step in the process
- Obtain help
- Leave the ADSA compiler

- Move between the action command line and the specification area (Main Menu only)

Default control keys

Activity	Control key	Description
HELP	[PF1]	Displays a map or field help screen, depending on cursor position If the cursor is on a map field associated with help text, a half screen of map field help text is displayed. If the cursor is set on a map field not associated with help text or anywhere else on the map, a full screen of map help text is displayed.
RETURN	[PF3]	From a pulldown window, returns to specification area. From the Main Menu screen, returns control to DC/UCF From a screen other than the Main Menu screen, applies updates to the current screen and returns to the Main Menu screen
BACKWARD	[PF4]	Applies updates to the current screen and displays the previous step in the process , as outlined on the Main Menu screen.
FORWARD	[PF5]	Applies updates to the current screen and displays the next step in the process , as outlined on the Main Menu screen.
BACKPAGE	[PF7]	Displays the previous screen of any step containing multiple screens.
FORWARD PAGE	[PF8]	Displays the next screen of any step containing multiple screens.
ACTION	[PF10]	Toggles the cursor position between the activity selection area action bar and the specification area on the Main Menu screen

2.2.3 Suspending a session

An application compiler session is automatically suspended in the event of a system crash. Additionally, the current work is saved whenever an update is made. Application definition sessions are entirely recoverable.

The developer can also suspend a session by selecting the **Release** option from the **Modify** window on the action bar. This allows any other developer to check the application out.

When a session is suspended, the application compiler saves the application definition, including all specifications made during the session, on queue records. A suspended session can be resumed at any time, as described in 2.2.1, “Invoking the application compiler” earlier in this section.

2.2.4 Terminating a session

When a session is terminated by compiling or deleting an application, the application compiler displays a blank Main Menu screen. The application developer can begin another session or can leave the application compiler by selecting an appropriate activity, such as **Switch**, from the action bar or by pressing [PF3].

When the application definition is complete, the application developer specifies **Compile** as the next activity on the action bar in the activity selection area of the Main Menu.

The application is compiled, and the resulting load module is stored in the data dictionary load area where it is available for execution.

►► For an example of an application compiler session, refer to the *CA-ADS User Guide*.

2.3 Application compiler screens

Screens are available for use during an application compiler session. All adding, modifying, deleting, compiling, displaying and switching is initiated from the Main Menu screen.

2.3.1 Main menu

The Main Menu screen is displayed when the application developer initiates an application compiler session. This screen is used to specify the action taken regarding the application, name an application and a dictionary, and specify the next step to be taken in the application definition.

Areas: The screen is composed of six areas:

- Activity selection area
- Dialog identification area
- Screen specification area
- Message area
- Command area
- Key assignment area

Activity selection area: Displays the application compiler activities available.

The application developer selects an activity to be performed one of two ways:

- By typing the name of the activity on the Command line in the lower left hand corner of the screen.
- By pressing PF10 to reach the Activity Selection Area, and, with the Tab key, positioning the cursor on the activity name and pressing the [Enter].
 - ▶▶ See Chapter 1, “Introduction to CA-ADS” for a discussion of the activities available from the dialog compiler Main Menu screen.

Application identification area: Specifies the application name, application number, dictionary name, and the dictionary node. The fields contained in this section are described below.

Screen specification area: Allows the application developer to specify the next step in the definition process. The application developer can either:

- Press Enter to go to the default next step
- Specify a step

▶▶ See Chapter 1, “Introduction to CA-ADS” for a discussion of the activities available from the application compiler Main Menu screen.

Message area: Displays informational and error messages returned from the application compiler.

Note that the control keys as described earlier in this section, (in addition to [Enter]) are identified at the bottom of this screen.

Command area: Provides a command line for entering the name of the desired action as specified in the activity selection area above. Action names can be abbreviated to the first three letters, ADD, MOD, DEL, COM, DIS or SWI. The system recognizes more than, but not less than, the first three letters of each identification.

If more than one activity is specified on the command line, an error message is displayed. If an activity is specified on the command line, and a control key is pressed, the activity associated with the control key is executed.

If an error is detected after the application developer selects an activity, the application compiler redisplay the current screen. The activity selection is retained and executed when the error is corrected. The application developer can override the initial selection by specifying another activity on the command line, selecting the activity directly from the selection area or by using [PF10].

Key assignment area: Presents the valid key choices and the action taken.

Control keys are described earlier in this section.

Main Menu screen

```

-   Add  Modify  Compile  Delete  Display  Switch
-   _____
-                               CA-ADS Application Compiler
-                               Computer Associates International, Inc.
-
-   Application name . . . .   _____
-   Application version . .   _____
-   Dictionary name . . . .   _____
-   Dictionary node . . . .   _____
-
-   Screen . . . . . _       1. General options
-                               2. Responses and Functions
-                               3. Global records
-                               4. Task codes
-
-   Copyright (C) 1972, 2000 Computer Associates International, Inc.
-   Command ==>
-   Enter  F1=Help  F3=Exit  F10=Action
    
```

Field descriptions:

Application name: Specifies the 1- to 8- character name of the current application. The application name must begin with an alphabetic character and cannot contain embedded blanks. An application name must be specified before any other application compiler activity can be executed.

Application version: Specifies the version number, in the range 1 through 9999, of the current application. If no version number is specified, version defaults to 1.

Dictionary name: Specifies the 1- to 8- character name of the data dictionary in which the application load module is stored. If no dictionary name is specified, **dictionary name** is set to the name of the dictionary identified in the user's profile or set through a DCUF SET DICTNAME statement. The dictionary name cannot change once it is validated.

Dictionary node: (DDS only) Specifies the node that controls the data dictionary specified by **Dictionary name**. **Dictionary node** defaults to the system currently in use.

Specifying a node name is equivalent to issuing a DCUF SET DICTNODE command under CA-IDMS/DC or CA-IDMS/UCF.

Screen: Provides the application developer with a quick form of navigation through the application definition process. By specifying the number which precedes the step name, the user avoids any unnecessary scrolling through the screens.

►► For a description of the screens involved in each step, see 2.2.2, “Sequencing through application compiler screens” earlier in this section.

2.3.2 General Options screen—Page 1

The first page of the General Options screen is used to specify options for an application:

- Description
- Maximum responses
- Date format
- Application compiler execution mode
- Application execution environment
- Default print destination and class

The first page of the General Options screen is accessed from the Main Menu by choosing option 1 at the **Screen** prompt.

The current settings for the application options are displayed on the screen. Each option can be changed by overwriting the displayed setting.

Sample screen

```

                                General Options                                Page 1 of 2

Application name: TESTAPPL  Version: 1

Description . . . TEST APPLICATION
Maximum responses . . . . . 500
Date format . . . . . 1  1. mm/dd/yy  2. dd/mm/yy
                        3. yy/mm/dd  4. yy/ddd
Execution environment . . . . . 1  1. Online  2. Batch
Default execution mode. . . . . 1  1. Step    2. Fast
Default print destination . . . . .
Default print class . . . . . 1

Enter F1=Help F3=Exit F4=Prev F5=Next F8=Fwd

```

Field descriptions:

Application name: Specifies the name of the current application, as specified on the Main Menu screen. This field is protected.

Version: Specifies the version number, in the range 1 through 9999, of the current application.

Description: Specifies a 1- to 32-byte description of the current application. This field is a documentation aid. The application description is included in the load module created for the application.

Maximum responses: Specifies the maximum number of responses, in the range 0 through 9999, that can be defined for the application. The default maximum number of responses is 500.

The application compiler creates a table of responses for each application; 19 bytes are allocated in the table for each response that can be defined. Therefore, the value specified for **Maximum responses** determines the amount of space that is allocated for the response table.

To optimize processing efficiency, the space allocated for the response table should be kept as small as possible. If the application developer attempts to add more responses than the maximum number specified, the application compiler returns a message indicating that the attempt to add a response was unsuccessful because of insufficient space. The developer can return to the General Options screen at any time during an application compiler session and increase the **Maximum responses** specification.

The value does limit the number of responses that can be added, but the size of the load module is exactly tailored to the actual number of responses, not set at this limit.

Date format: Specifies the format in which the current date appears on runtime menu and help screens. At runtime, the current date is retrieved from DC/UCF and is stored in the specified format in ADSO- APPLICATION-GLOBAL-RECORD and ADSO-APPLICATION- MENU-RECORD, if applicable. When a runtime menu or help screen is displayed, the runtime system retrieves the date from the applicable record.

►► For descriptions of ADSO-APPLICATION-GLOBAL-RECORD and ADSO-APPLICATION-MENU-RECORD, see Appendix A, "System Records."

The date format is selected by entering the appropriate number in the response field following **Date format**. Available formats are as follows:

- **MM/DD/YY** (for example, 07/25/91). MM/DD/YY is the default.
- **DD/MM/YY** (for example, 25/07/91)
- **YY/MM/DD** (for example, 91/07/25)

- **DDD/YY** (for example, 207/91)

Execution environment: Specifies whether the application will execute online or under CA-ADS/Batch.

Default execution mode: Sets the default execution mode for the application.

If STEP mode is specified, the runtime system responds to a user signon with the message that the signon is accepted. The user then must press [Enter] to initiate the first function of the application. STEP mode is the default.

If FAST mode is specified, the system responds to an acceptable signon by directly initiating the first function automatically.

Default print destination: Specifies a DC/UCF print destination. If not specified, the print destination defaults at runtime to the system default.

At runtime, the specified print destination is stored in the AGR-PRINT-DESTINATION field of the ADSO-APPLICATION-GLOBAL-RECORD. WRITE PRINTER commands can use the default by specifying a print destination of AGR-PRINT-DESTINATION.

►► For a description of ADSO-APPLICATION-GLOBAL-RECORD, see Appendix A, “System Records.”

Default print class: Specifies a DC/UCF print class number in the range 1 through 64. If not specified, the print class defaults at runtime to the physical terminal default.

At runtime, the specified print class is stored in the AGR-PRINT-CLASS field of the ADSO-APPLICATION- GLOBAL-RECORD. WRITE PRINTER process commands can use the default by specifying a print class of AGR-PRINT-CLASS.

►► For a description of ADSO-APPLICATION-GLOBAL-RECORD, see Appendix A, “System Records.”

2.3.3 General Options screen—Page 2

The second page of the General Options screen is used to specify runtime security restrictions for a CA-ADS application.

How to access: The application developer accesses this screen from the first General Options screen in one of two ways:

- Pressing [PF8]
- Entering a 2 in the **Page** field and pressing [Enter]

Security classes: This screen allows the application developer to specify a DC/UCF security class for the current application. Application security class is no longer used by the CA-ADS runtime system and is not used by CA-IDMS internal security. It is provided for downward compatibility with applications compiled under previous releases of DC/UCF and for installations that use privately designed security systems that rely on the application security class being stored in the ADSO-APPLICATION-GLOBAL-RECORD during runtime.

Note: Security classes assigned to responses are checked by CA-IDMS central security if activity security has been enabled.

For more information, see Appendix G, “Security Features.”

Signon functions: The second page of the General Options screen also allows the application developer to specify a signon function to be executed before any other application function. A signon function, if specified, is the first function initiated by the runtime system. If signon is required, the application cannot be executed until an acceptable signon is entered. If signon is optional, the application can be executed whether or not a signon is entered.

►► For more information on signon menu maps, see 4.2.3, “System-defined menu maps.”

Security for runtime menus: The application developer can also specify whether runtime menus are to be security tailored. Only those responses for which the user has execution authority are displayed on security-tailored menus.

The second page of the ADSA General Options screen itself can be the object of user security restrictions imposed by the security administrator, through restricting execution authority for program ADAPGOP2. Only application developers having execution authority for ADAPGOP2 would have access to the second page of the General Options screen.

►► For information on CA-IDMS central security, refer to *CA-IDMS Security Administration*.

Sample screen

```

                                General Options                                Page 2 of 2
Application name: TESTAPPL  Version: 1
Security class . . . . . 42
Menus are . . . . . 1  1. Not used  2. Security tailored
                        3. Untailored
Signon is . . . . . 1  1. Not used  2. Optional
Signon function is. . . . .
Enter F1=Help F3=Exit F4=Prev F5=Next F7=Bkwd

```

Field descriptions:

Application name: Specifies the name of the current application, as specified on the Main Menu screen. This field is protected.

Security class: Applicable to online applications only, specifies the DC/UCF security class, in the range 1 to 256, assigned to the application.

File specification allows compatibility with Release 14.0 for applications compiled under previous releases of CA-IDMS and maintains the functionality of installation-designed security systems that rely on application security classes being stored in the ADSO-APPLICATION-GLOBAL-RECORD during runtime.

Menus are: Specifies whether runtime menus are security tailored. The application developer can select one of the following specifications by entering the appropriate number in the data field following this specification. The options are:

1. **Not used** — specifies that the application does not use menus.
2. **Security tailored** — specifies that only those responses that the user has authority to execute are displayed on the runtime menus.
3. **Untailored** — (default) specifies that all responses defined as valid for application functions are displayed on menus, regardless whether the user has the authority to execute them.

The CA-ADS runtime system tests to determine whether the user has authority to execute each menu response if menus are security tailored. Only those responses for which the user has execution authority are then displayed on the menus. If the user attempts to execute a response for which execution authority is not granted, the current function screen is redisplayed with the following message:

```
UNACCEPTABLE RESPONSE. PLEASE TRY AGAIN
```

Signon is: Specifies whether a signon function is executed for the application. The application developer can select a signon specification by entering the applicable number in the field following this specification. A signon function can be specified as follows:

1. **Not used** — (default) specifies that no signon function is executed for the application.
2. **Optional** — specifies that a signon function will be executed only when the user is not signed on to DC/UCF. When the user is already signed on to DC/UCF, the task top function is executed instead of the signon function (if it is a different function). Also, the user is not required to sign on to the application to execute unsecured functions.
3. **Required** — specifies that the application signon function is always executed regardless of whether the user has signed on to DC/UCF, and regardless of which function is the task top function. The user can only execute other functions after successfully signing on to the application.

Signon function is: Specifies the name of a signon menu function to be defined by using the **Function Definition (Menu)** screen. If **Signon** is specified as **Optional** or **Required**, a signon function must be supplied. If **Signon** is specified as **Not used**, a signon function name cannot be specified.

To identify system functions, enter the system function name. System functions are reserved and cannot be edited.

ADSA warns the application developer if a function type code (1,2, or 3), a program name, or a dialog name are reserved for a system function.

►► For information on defining signon menu functions, refer to 4.2.3, “System-defined menu maps.”

2.3.4 Response/Function List screen

The Response/Function List screen is accessed from the Main Menu by choosing option 2 at the **Screen** prompt. This screen is used to:

- Identify each response name for the application
- Identify the associated control key
- Identify the function associated with the response
- Specify the function type
- Name the program or dialog

For each response defined, the combination of response name, associated assigned key, and function initiated must be unique within the application.

Up to 12 responses and functions can be entered on one page of the Response/Function List screen.

The application developer can scroll between pages using the control keys associated with paging forward and paging backward. See earlier in this section for a listing of the default control key assignments for the application compiler.

From the Response/Function List screen, the application developer can further define both responses and functions by accessing the following screens:

- Response Definition screen
- Function Definition (Dialog) screen
- Function Definition (Program) screen
- Function Definition (Menu) screen

To access one of these screens, a nonblank character is placed in the appropriate **Select** field.

Sample screen

Response/Function List				Page 1 of 1	
Application name: TESTAPP1 Version: 1					
Select (/)	Response name	Assigned key	Select (/)	Function name/type(1,2,3)*	Program/Dialog name
-	_____	_____	-	_____ / _____	_____
-	_____	_____	-	_____ / _____	_____
-	_____	_____	-	_____ / _____	_____
-	_____	_____	-	_____ / _____	_____
* Type: 1. Dialog 2. Program 3. Menu					
Enter F1=Help F3=Exit F4=Prev F5=Next F7=Bkwd F8=Fwd					

Field descriptions:

Application name: Specifies the name of the current application, as specified on the Main Menu screen. This field is protected.

Version: Specifies the version number, in the range 1 through 9999, of the current application.

Select: Placing a nonblank character in this field allows the developer to select a particular response or function for further definition.

►► See 2.3.5, “Response Definition screen,” 2.3.6, “Function Definition (Dialog) screen” on page 2-27 , 2.3.7, “Function Definition (Program) screen,” and 2.3.8, “Function Definition (Menu) screen” on page 2-32 for further information.

Response name: Displays the name of the application response.

The following considerations apply:

- For CA-ADS, the response name cannot contain embedded blanks. At runtime, the response name can be used in a \$RESPONSE map field to select the response. The response name is also stored by the runtime system in the AMR-RESPONSE-FIELD of ADSO- APPLICATION-MENU-RECORD for use in runtime menus.
- For CA-ADS/Batch, if the response field for an input record is the concatenation of several fields, the response name specified on the Response Definition screen must include any embedded blanks that occur in a concatenation. For example, the entry 'ADD 'E' is made for a response field that is the concatenation of two fields, the first being six bytes long and the second being one byte long. The first field contains the field value of ADD and the second field contains E.

Assigned key: Specifies the control key or control event associated with the current application response. specifies an online control key or a batch control event that selects the response at runtime.

The following considerations apply:

- Valid **online** assigned key specifications are ENTER, CLEAR, PA1 through PA3, PF1 through PF24. LPEN can be specified as a control key if the use of light pens is supported by the installation. The following consideration applies:
 - **CLEAR, PA1, PA2, and PA3** do not transmit data.
- Valid **batch** control events are EOF and IOERR. The following considerations apply:
 - **EOF** indicates that the most recent input-file read operation resulted in an end-of-file condition.
 - **IOERR** indicates that the most recent input file read operation resulted in a physical input-error condition. In CA-ADS/Batch, an output error causes the runtime system to terminate the application.

Function name/type: Displays the name and type of the application function associated with the response.

The function name cannot contain embedded blanks.

The application compiler supplies a function type by crosschecking the defined functions and responses.

Function types are as follows:

1. **Dialog** — The response is associated with a dialog function.
2. **Program** — The response is associated with a user program function.
3. **Menu** — The response is associated with a menu function.

For example, if the application developer specifies **Menu** and also provides a dialog name in the **Associated dialog** field, of a Function Definition screen, the function is associated with a menu.

When the application developer associates the response process with the dialog, using the dialog compiler Process Modules screen, the **Value** and **Key** specified should match the **Response name** and **Assigned key** entered on the Response/Function List screen.

►► For a description of the Process Modules screen, see Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

Associating a response with an internal function causes the dialog's response process to be displayed as a valid response on runtime menu and help screens.

Program/dialog name: Specifies the name of the program or dialog associated with the function.

Response/Function Search: The ADSA compiler:

- Supports up to 999 pages of responses and function relationships
- Returns to the current Response/Function screen when the selection list of responses and functions has been exhausted
- Provides a search function that allows partial keys and both next (forward) and previous (backward) searches.

An example of the search function follows.

Invoking the search function: Pressing [PF6] brings up the search window. In this example, a partial key — 'IUA' — has been entered. The search will attempt to match any response name beginning with those letters:

```

RELEASE 15.0                                CAGJF0
                                Response/Function List      Page 1 of 2
Application name: METAPPL1  Version: 1

Select  Response  Assigned  Select  Function  Program/
(/)     name      key       (/)     name/type(1,2,3)*  Dialog name
-       R1       ENTER    -       F1        / 1        JPKSQLD1
-       R3
-       R2
-       R4
-       R5
-       LINKOLQR

                                Search for. . .
                                Response  Assigned  Function
                                name      key       name
                                -----
                                IUA
                                -----
                                F3=Exit  F7=Prev  F8=Next
                                -----
                                IDMSOLQS

                                * Type: 1. Dialog 2. Program 3. Menu

Enter F1=Help F3=Exit F4=Prev F5=Next F6=Search F7=Bkwd F8=Fwd

```

Search result: Pressing [PF8] initiates a forward search. In this example, response IUADOLQR is found:

```

RELEASE 15.0                                CAGJF0
                                Response/Function List      Page 2 of 2
Application name: METAPPL1  Version: 1

Select  Response  Assigned  Select  Function  Program/
(/)     name      key       (/)     name/type(1,2,3)*  Dialog name
-       IUADOLQR  PF07     -       IUADOLQF / 1      IUADOLQ1
-       _____
-       _____
-       _____
-       _____
-       _____

                                Search for. . .
                                Response  Assigned  Function
                                name      key       name
                                -----
                                IUA
                                -----
                                F3=Exit  F7=Prev  F8=Next
                                -----

                                * Type: 1. Dialog 2. Program 3. Menu
DC451536 Matching entry found on page 2

Enter F1=Help F3=Exit F4=Prev F5=Next F6=Search F7=Bkwd F8=Fwd

```

2.3.5 Response Definition screen

The Response Definition screen enables the application developer to provide extended specifications when defining responses. These specifications include:

- Description
- Security class

- Response type
- Response execution
- Assigned key
- Control command

The Response Definition screen is accessed by entering a nonblank character in the appropriate **Select** field on the Response/Function List screen and pressing [PF5].

Sample screen

```

                                Response Definition
Application name:  TEST1      Version:    1
Response name:    QUIT
Function invoked: QUIT
Description . . . . . _____ Security class:  1
Response type. . . . . 2   1. Global      2. Local
Response execution . . . . 2   1. Immediate  2. Deferred
Assigned key . . . . . PF01
Control command. . . . . 1   1. Transfer      2. Invoke
                               3. Link              4. Return
                               5. Return continue  6. Return clear
                               7. Return continue clear 8. Transfer nofinish
                               9. Invoke nosave     10. Link nosave

Enter F1=Help F3=Exit F4=Prev F5=Next
    
```

Field descriptions:

Application name: Specifies the name of the current application, as specified on the Main Menu screen. This field is protected.

Version: Specifies the version number, in the range 1 through 9999, of the current application.

Response name: Displays the name of the application response selected on the Response/Function List screen.

This field can be modified by the user. The first character of the response name cannot be blank. If modified, the user should insure that the combination of response name, assigned key, and associated function must be unique within the application.

Drop response: Removes the response definition from the application. CA-ADS does *not* drop the function associated with the dropped response.

Function invoked: Displays the function invoked by the current application response, as specified on the Response/Function List screen.

This field is protected.

Description: Specifies a 1- to 28-byte description of the current response. The response description is displayed with the associated response name on runtime menu and help screens. Note that the specified description is truncated to 12 characters on the short description menu screen.

►► For a discussion of the runtime menu and help screens, see Chapter 4, “CA-ADS Runtime System.”

Security class: Specifies the security class for the response. Valid security class values are 1 to 256. See your Security Administrator about the security class conventions being used at your site.

►► For more information about security classes, see G.3.1, “Response security.”

Response type: Specifies whether the response is global or local, as follows:

1. **Global** — The response is valid for all functions in the application. Global responses can be deselected from the list of valid responses for a specific function.
2. **Local** (default) — The response is valid only for those functions with which it is explicitly associated on the Function Definition screen.

A response is specified as global (that is, valid for all functions in the application) or local (that is, valid only if explicitly associated with a function). For each response defined, the combination of response name, associated control key, and function initiated must be unique within the application.

Response execution: Specifies whether the invoked function is immediately executable or deferred. The following considerations apply:

- In online applications, the default for all functions except the HELP, SIGNON, SIGNOFF, FORWARD, and BACKWARD system functions is deferred.
- In the batch environment, the default for all functions is immediately executable.

Defaults can be overridden by entering the appropriate number in the data field immediately following the **Response execution** prompt.

Assigned key: Specifies the control key or control event associated with the current application response. specifies an online control key or a batch control event that selects the response at runtime.

The following considerations apply:

- Valid **online** assigned key specifications are ENTER, CLEAR, PA1 through PA3, PF1 through PF24, FWD, BWD, and HDR. LPEN can be specified as a control key if the use of light pens is supported by the installation. The following considerations apply:
 - **CLEAR, PA1, PA2, and PA3** do not transmit data.
 - The **FWD, BWD, and HDR** control keys are associated with pageable maps.

FWD and **BWD** are synonymous with the keyboard control keys defined for paging forward and backward respectively. If FWD or BWD is specified and the keys defined for paging forward and backward are changed, the response definition does not have to be updated or the application recompiled.

HDR is not associated with any keyboard control key. Conditions encountered during a map paging session cause the response associated with this control key value to be selected.

►► For more information on the effect of HDR on the runtime flow of control, see Chapter 4, “CA-ADS Runtime System.”

- Valid **batch** control events are EOF and IOERR. The following considerations apply:
 - **EOF** indicates that the most recent input-file read operation resulted in an end-of-file condition.
 - **IOERR** indicates that the most recent input file read operation resulted in a physical input-error condition. In CA-ADS/Batch, an output error causes the runtime system to terminate the application.

Control command: Specifies the CA-ADS control command used to pass processing control to the function associated with the response, as follows:

1. **Transfer** (default) — Control is passed by means of a TRANSFER command.
2. **Invoke** — Control is passed by means of an INVOKE command.
3. **Link** — Control is passed by means of a LINK command.
4. **Return** — Control is passed by means of a RETURN command.
5. **Return Continue** — Control is passed by means of a RETURN command to the premap process.
6. **Return Clear** — Control is passed by means of a RETURN command and buffers are initialized.
7. **Return Continue Clear** — Control is passed by means of a RETURN command to the premap process and buffers are initialized.

In process code for dialogs associated with functions, the only control command needed is EXECUTE NEXT FUNCTION. When a valid response is made, EXECUTE NEXT FUNCTION causes the runtime system to execute the control command associated with the response. The control commands perform the same record buffer and currency maintenance as they do when they are coded in processes.

►► For more information on control commands, see Chapter 15, “Control Commands.”

2.3.6 Function Definition (Dialog) screen

The Function Definition (Dialog) screen is accessed by entering a nonblank character in the appropriate **Select** field on the Response/Function List screen and pressing [PF5]. The function chosen must be associated with a type of **dialog**.

This screen is used to:

- Provide a description of the dialog function
- Identify the associated dialog name
- Identify a user exit dialog
- Name the default response
- Specify valid responses for the current dialog function

The screen provides an alphabetical listing of all responses valid for the application. Responses that are valid are indicated by an X.

The application developer can select additional valid responses by typing a nonblank character in the 1-byte field immediately preceding the applicable response. The application developer can deselect any valid response by overwriting the 1-byte field immediately preceding the response with a blank or by using the ERASE EOF key.

Up to 6 responses can be displayed on one page of the Function Definition screen.

The application developer can scroll between pages using the control keys associated with paging forward and paging backward. See earlier in this section for a listing of the default control key assignments for the application compiler.

Sample screen

```

                                Function Definition (Dialog)           Page 1 of 2
Application name:  GWGAPP01  Version:  1
Function name:    F1          Drop function (/) _
Description . . . DEFINED

Associated dialog . . . . . D1          User exit dialog . . . . . _____
Default response . . . . . _____

Valid response(/) Response Key  Function      Valid response(/) Response Key  Function
-          R1          PF01  F1          -          R15          _____
-          R10         _____ -          R2          PF02  F2
-          R11         _____ -          R3          PF08  FWD
-          R12         _____ -          R4          _____ FORWARD
-          R13         _____ -          R6          _____ HELP
-          R14         _____ -          R7          _____

                                more ...

Enter F1=Help F3=Exit F4=Prev F5=Next F7=Bkwd F8=Fwd
    
```

Field descriptions:

Page: Specifies the page number of the Function Definition (Dialog) screen to be displayed. If more than one page exists for the screen, this field displays the current page of the total number of pages, as shown on the sample screen above.

This field is modifiable so that you can access the valid responses for the displayed dialog function quickly. To request the next map page to be displayed:

- Press the control key associated with paging forward or paging backward one page (the system generation defaults are [PF8] and [PF7], respectively)
- Enter a numeral for the page that you want to access, and press any control key other than keys assigned for paging forward or backward

Application name: Specifies the name of the current application, as specified on the Main Menu screen.

This field is protected.

Version: Specifies the version number, in the range 1 through 9999, of the current application.

This field is protected.

Function name: Displays the name of the current function, as specified on the Response/Function List screen.

This field can be modified by the user. The first character of the Function name cannot be blank. If modified, the user should insure that the combination of response name, assigned key, and associated function name must be unique within the application.

Drop function: Removes the function definition from the application.

Description: Specifies a 1- to 28-byte description of the current response. The response description is displayed with the associated response name on runtime menu and help screens. Note that the specified description is truncated to 12 characters on the short description menu screen.

You must change the literal, UNDEFINED, to something else, or CA-ADS displays the following error message:

```
DC462226 FUNCTION FUNC4 IS UNDEFINED
```

►► For a discussion of the runtime menu and help screens, see Chapter 4, “CA-ADS Runtime System.”

Only the first 12 bytes of each description are displayed.

Associated dialog: Specifies the name of the dialog or user program associated with the function. If this field is left blank, the function is associated with a system-defined menu.

If the application developer provides a dialog name and also specifies **Menu** as the **Function type** on the Response/Function List screen, the function is associated with a menu.

Menu cannot be specified for CA-ADS/Batch applications.

User exit dialog: Specifies the name of a dialog to which a dialog function can LINK internally.

When the dialog function is initiated at runtime, the name of the dialog supplied as the user exit dialog is stored in the AGR-EXIT-DIALOG field of ADSO-APPLICATION-GLOBAL-RECORD. When the runtime system encounters a LINK TO AGR-EXIT-DIALOG command, the dialog named in the AGR-EXIT-DIALOG field becomes the object of the LINK command.

►► For more information on ADSO-APPLICATION-GLOBAL-RECORD, see Appendix A, “System Records.”

Default response: Specifies the name of the response initiated by the runtime system when the user presses [Enter] without entering a specific response. The default response is displayed in bright intensity on the Function Definition screen for each function.

Valid response: A nonblank character in this field indicates that this response is valid for this function.

Response: Displays the name of a response from the Response/Function List screen.

Key: Displays the assigned key that initiates the response.

Function: Displays the name of the function initiated by the response.

2.3.7 Function Definition (Program) screen

The Function Definition (Program) screen is accessed by entering a nonblank character in the appropriate **Select** field on the Response/Function List screen and pressing [PF5]. The function chosen must be associated with a type of **program**.

The Function Definition (Program) screen is used to:

- Provide a description of the program function
- Identify the associated program name
- Specify record buffers and control blocks passed to a program at runtime.

Information provided on this screen applies only to a current function associated with a user program.

Up to eight records can be specified on one page of the Function Definition (Program) screen.

The application developer can scroll between pages using the control keys associated with paging forward and paging backward. Refer to the table earlier in this section for a listing of the default control key assignments for the application compiler.

When a function associated with a user program is initiated at runtime, CA-ADS passes control to the program. Additionally, the runtime system passes the data in the record buffers and control blocks specified on the Function Definition (Program) screen. CA-ADS maintains all application record buffers at the level at which control was relinquished.

►► For a discussion of application levels, see Chapter 15, “Control Commands.”

When a user program finishes execution, control returns to the mapout operation of the function from which the program was initiated or, if the program was initiated by an EXECUTE NEXT FUNCTION command, to the command that follows EXECUTE NEXT FUNCTION. Note that a user program must process its own responses. Valid responses cannot be specified for the function associated with the program.

The Function Definition (Program) screen is similar to the USING clause of the LINK TO PROGRAM control command.

►► For more information on the LINK TO PROGRAM control command, see Chapter 15, “Control Commands.”

Sample screen

Function Definition (Program)		
Application name: TEST1	Version: 1	
Function name: PROG01		Drop function (/) _
Associated program	PROG01	
Description	UNDEFINED	
	Records passed	Drop record (/)
1.	_____	-
2.	_____	-
3.	_____	-
4.	_____	-
5.	_____	-
6.	_____	-
7.	_____	-
8.	_____	-
Enter F1=Help F3=Exit F4=Prev F5=Next F7=Bkwd F8=Fwd		

Field descriptions:

Application name: Specifies the name of the current application, as specified on the Main Menu screen. This field is protected.

Version: Specifies the version number, in the range 1 through 9999, of the current application.

This field is protected.

Function name: Displays the name of the current function, as specified on the Response/Function List screen. The function must be associated with a user program.

This field can be modified by the user. The first character of the Function name cannot be blank. If modified, the user should insure that the combination of response name, assigned key, and associated function name must be unique within the application.

Drop function: Removes the function definition from the application.

Associated program: Displays the name of the user program with which the current function is associated, as specified on the Response/Function List screen.

This field is protected.

Description: Specifies a 1- to 28-byte description of the current function. The function description is displayed with the associated response name on runtime menu and help screens. Note that the specified description is truncated to 12 characters on the short description menu screen.

►► For a discussion of the runtime menu and help screens, see Chapter 4, “CA-ADS Runtime System.”

Records passed: Specifies the data passed to the user program. The application developer specifies record names and/or control block names as follows:

- **Record name** passes the buffer for the specified record to the user program. The specified record must be known to the issuing function.
ADSO-APPLICATION-MENU-RECORD is the only record that can be passed to a user program from a system-defined menu function.
- **MAP-CONTROL/MAP_CONTROL** passes the map request block (MRB) of the issuing function to the user program.
- **SUBSCHEMA-CONTROL/SUBSCHEMA_CONTROL** passes the subschema control block of the issuing function to the user program.

The record and control block names must be entered from left to right, top to bottom, in the same order in which they are defined in the program.

Drop record: Removes the record from its association with the program function, but does not delete the record definition from the dictionary.

2.3.8 Function Definition (Menu) screen

The Function Definition (Menu) screen is accessed by entering a nonblank character in the appropriate **Select** field on the Response/Function List screen and pressing [PF5]. The function chosen must be associated with a type of **menu**.

The Function Definition (Menu) screen is made up of two screens used to specify the characteristics of runtime menu screens and the responses to be listed on that menu.

Page 1 of the Function Definition (Menu) screen allows the application developer to specify:

- A description of the menu
- The name of the associated dialog if this is a menu/dialog
- The default response, if any
- The name of the user exit dialog, if any
- Whether the menu is defined by the site or the system
- The description length
- The number of responses per page

- The number of heading lines and their content

Page 2 of the Function Definition (Menu) screen allows the application developer to specify:

- The responses to be displayed on the menu
- The order in which the responses will be displayed at runtime

To specify the number of responses per page, the application developer specifies that the menu is user-defined and specifies the number of responses, from 0 to 50. If the application developer specifies that the menu is system-defined, the number of responses is set by CA-ADS: 12 for a signon menu, 15 for a nonsignon menu that uses long descriptions, and 30 for a nonsignon menu that uses short descriptions.

►► For further information on signon menus, see 4.2.3, “System-defined menu maps.”

►► For a discussion of the runtime menu screens, see Chapter 4, “CA-ADS Runtime System.”

The Function Definition (Menu) screen also allows the application developer to specify up to three lines of heading text for display at the top of each menu page. The heading text can use any or all of the three lines available.

Information provided on the Function Definition (Menu) screens applies only to a current function associated with a menu or a menu/dialog. The Function Definition (Menu) screen is not available when defining CA-ADS/Batch.

Page 1 of Function Definition (Menu)

Function Definition (Menu)		Page 1 of 2
Application name: TEST	Version: 1	Drop function (/) _
Function name: MENU1		
Description . . . UNDEFINED		
Associated dialog _____		
Default response _____	User exit dialog _____	
Use signon menu (/). _____		
Menu defined by: _____	1. User	2. System
Description length 1	1. Long (28)	2. Short (12)
Responses per page 15		
Number of heading lines (0-3). 0		
Heading line text		

.....+.....1.....+.....2.....+.....3.....+.....4.....+.....5.....+.....6.....+.....7.....+.....		
Enter F1=Help F3=Exit F4=Prev F5=Next F8=Fwd		

Field descriptions for page 1:

Application name: Specifies the name of the current application, as specified on the Main Menu screen. This field is protected.

Version: Specifies the version number, in the range 1 through 9999, of the current application.

This field is protected.

Function name: Displays the name of the current function, as specified on the Response/Function List screen. The function must be associated with a user program.

This field can be modified by the user. The first character of the Function name cannot be blank. If modified, the user should insure that the combination of response name, assigned key, and associated function name must be unique within the application.

Drop function: Removes the function definition from the application.

Description: Specifies a 1- to 28-byte description of the current function. The function description is displayed with the associated response name on runtime menu and help screens. Note that the specified description is truncated to 12 characters on the short description menu screen.

►► For a discussion of the runtime menu and help screens, see Chapter 4, “CA-ADS Runtime System.”

Associated dialog: Specifies the name of the dialog or user program associated with the function. If this field is left blank, the function is associated with a system-defined menu.

If the application developer provides a dialog name and also specifies **Menu** as the **Function type** on the Response/Function List screen, the function is associated with a menu.

Menu cannot be specified for CA-ADS/Batch applications.

Default response: Specifies the name of the response initiated by the runtime system when the user presses [Enter] without entering a specific response. The default response is displayed in bright intensity on the Function Definition screen for each function.

User exit dialog: Specifies the name of a dialog to which a dialog function can LINK internally.

When the dialog function is initiated at runtime, the name of the dialog supplied as the user exit dialog is stored in the AGR-EXIT-DIALOG field of ADSO-APPLICATION-GLOBAL-RECORD. When the runtime system encounters a

LINK TO AGR-EXIT-DIALOG command, the dialog named in the AGR-EXIT-DIALOG field becomes the object of the LINK command.

►► For more information on ADSO-APPLICATION-GLOBAL-RECORD, see Appendix A, “System Records.”

Use signon menu: Specifies whether the menu is a signon menu. At runtime, the menu uses the AMR-USER-ID and AMR-PASSWORD fields of ADSO-APPLICATION-MENU-RECORD.

►► For more information on ADSO-APPLICATION-MENU-RECORD, see Appendix A, “System Records.”

If the signon menu is system-defined, up to 12 responses are displayed on each page at runtime.

Menu defined by: Specifies whether the menu is system-defined or user-defined, as follows:

1. specifies that the menu is user-defined, meaning that the user can specify the number of responses per page, from 0 to 50.
2. (default) specifies that the menu is system-defined, meaning that CA-ADS determines the number of responses per menu page: 12 for a signon menu, 15 for a nonsignon menu that uses long descriptions, and 30 for a nonsignon menu that uses short descriptions.

Description length: Specifies the description length for nonsignon menus, as follows:

1. (default) specifies that each function description displayed on the menu screen contains the complete 28-byte description text. The long description allows up to 15 responses to be displayed on each page of a system-defined menu.
2. specifies that each description displayed on the menu screen is truncated to the first 12 bytes of the description text. The short description allows up to 30 responses to be displayed on each page of a system-defined menu.

At runtime, the description displayed is the description specified on the Response Definition screen. If the response definition has no description, the runtime system displays the description of the associated function for the response.

Responses per page: Specifies the maximum number of responses, in the range 0 through 50, that can be displayed on one page of a user-defined menu at runtime.

The maximum number of responses per page for a system-defined menu is determined by the menu format. A system-defined signon menu has 12 responses per page. Other system-defined menus have either 15 or 30 responses per page, depending on the length of the description (see **Description length**).

The default is 15.

Number of heading lines: Specifies the number of heading lines displayed at the top of each page of the runtime menu screen.

The default is 0 (that is, no heading lines).

Heading line text: Specifies the heading text displayed at the top of each page of the runtime menu screen. The application developer can enter free-form text in the three 79-byte fields provided.

Page 2 of Function Definition (Menu)

Function Definition (Menu)					Page 2 of 2				
Application name:		TEST1	Version:		1				
Function name:		MENU1							
Valid resp.	Seq. #	Response	Key	Function	Valid Resp.	Seq. #	Response	Key	Function
-	_____	ADD	PF02	F2	-	_____	_____	_____	_____
-	_____	QUIT	PF01	QUIT	-	_____	_____	_____	_____
-	_____	_____	_____	_____	-	_____	_____	_____	_____
Enter F1=Help F3=Exit F4=Prev F5=Next F7=Bkwd F8=Fwd F9=Update Seq _____									

Field descriptions for page 2:

Note: Page 2 of the Function Definition (Menu) only appears when you have previously defined responses.

Application name: Specifies the name of the current application, as specified on the Main Menu screen. This field is protected.

Version: Specifies the version number, in the range 1 through 9999, of the current application. This field is protected.

Function name: Displays the name of the current function, as specified on the Response/Function List screen. The function must be associated with a user program. This field is protected.

Valid response: A nonblank character in this field indicates that this response is valid for this function.

Seq.#: Specifies the sequence number of each response. The application developer can modify the position or suppress the display of a response by overwriting the sequence number in this field.

Response: Specifies the name of each response selected as valid for the current function.

This field is protected.

Key: Specifies the control key or control event associated with the current application response, as specified on the Response/Function List screen.

This field is protected.

Function: Specifies the name of the current function, as specified on the Response/Function List screen. The function must be associated with a menu or a menu/dialog.

This field is protected.

►► For information on how to specify signon menus, see 4.2.3, “System-defined menu maps.”

2.3.9 Global Records screen

The Global Records screen is used to specify records that are available to all functions in an application at runtime.

The Global Records screen is accessed from the Main Menu by choosing option 3 at the **Screen** prompt.

The application developer can scroll between pages of the Global Records screen by using the control keys associated with paging forward and paging backward. Refer to the table earlier in this section for a listing of the default control key assignments for the application compiler.

Global records must be defined in the data dictionary before the application is compiled. The records can be work records or map records. If a subschema record is specified, the application compiler uses the IDD description of the record. Individual subschema views are not used.

Once specified, global records are available to all dialogs, maps, and user programs defined for the application.

There is no limit to the number of records which can be specified.

Record buffers for global records are maintained across application functions, regardless of the means of transfer of control. Thus, values in the records are preserved for the duration of the application execution.

A global record used by a dialog must be associated with the dialog. The record can be defined as a work record on the dialog compiler Records and Tables screen or it can be associated with the dialog map or subschema. The Records and Tables screen is discussed in Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

►► For information on associating records with maps, refer to the *CA-IDMS Mapping Facility* manual.

Application global records are optional. Note, however, that ADSO-APPLICATION-GLOBAL-RECORD is automatically included in the list of global records. The application developer can delete this record, but should be aware that deleting ADSO- APPLICATION-GLOBAL-RECORD disables many of the runtime capabilities provided by CA-ADS.

►► For a description of ADSO-APPLICATION-GLOBAL-RECORD, see Appendix A, “System Records.”

Sample screen

Global Records		Page 1 of 1	
Application name: TESTAPP1		Version: 1	
	Record name	Version	Drop record (/)
1.	ADSO-APPLICATION-GLOBAL-RECORD	1	-
2.	_____	_____	-
3.	_____	_____	-
4.	_____	_____	-
5.	_____	_____	-
6.	_____	_____	-
7.	_____	_____	-
8.	_____	_____	-
Enter F1=Help F3=Exit F4=Prev F5=Next F7=Bkwd F8=Fwd			

Field descriptions:

Application name: Specifies the name of the current application, as specified on the Main Menu screen. This field is protected.

Version: Specifies the version number, in the range 1 through 9999, of the current application.

Record name: Specifies the 1- to 32-character name of each global record for the application. The named record must be defined in the data dictionary before the application is compiled.

The application developer can delete records already specified by overwriting the record name with blanks or by using the ERASE EOF key.

Version Specifies the version number, in the range 1 through 9999, of the current application. If no version number is specified, version defaults to 1.

Drop record (/): Removes the record from its association with the application, but does not delete the record definition from the dictionary.

2.3.10 Task Codes screen

The Task Codes screen is used to specify DC/UCF task codes that initiate an application at runtime. Each task code is associated with an application function.

The application compiler updates the table of task codes and associated functions (task application table) that is referenced by the CA-ADS runtime system.

The Task Code screen is accessed from the Main Menu by choosing option 4 at the **Screen** prompt.

At runtime, the user can enter one of the specified task codes. If a signon is not required, the associated function is executed as the first function in the application. If a signon is required, the associated function is executed as the first function after an acceptable signon is entered.

►► For more information on the use of application task codes, see Chapter 4, “CA-ADS Runtime System.”

At least one task code must be specified for each application. Up to eight task codes and corresponding functions can be specified on one page of the Task Codes screen. The application developer can specify additional task codes by pressing [Enter] to enter the specified task codes and then pressing the applicable control key to display a blank Task Codes screen.

The application developer can scroll between pages of the Task Codes screen by using the control keys associated with paging forward and paging backward. See earlier in this section for a listing of the default control key assignments.

Sample screen

Task Codes		Page 1 of 1
Application name: TEST1	Version: 1	
Task Code	Function	Drop (/)
1. _____	_____	-
2. _____	_____	-
3. _____	_____	-
4. _____	_____	-
5. _____	_____	-
6. _____	_____	-
7. _____	_____	-
8. _____	_____	-
Enter F1=Help F3=Exit F4=Prev F5=Next F7=Bkwd F8=Fwd		

Field descriptions:

Application name: Specifies the name of the current application, as specified on the Main Menu screen. This field is protected.

Version: Specifies the version number, in the range 1 through 9999, of the current application.

This field is protected.

Task code: Specifies the 1- to 8-character name of each DC/UCF task code for the application. Task code names cannot contain embedded blanks. The application developer can delete task codes already specified by entering a nonblank character in the Drop ID column opposite the task code to be dropped.

Note: Task codes must be defined to DC/UCF at system generation by means of the TASK statement before they can be used to initiate an application directly from DC/UCF without also having to specify the task code for the runtime system. Task codes defined at system generation must invoke ADSORUN1.

►► For more information on the TASK statement, refer to *CA-IDMS System Generation*.

►► For more information on initiating an application, see Chapter 4, “CA-ADS Runtime System.”

Function name: Displays the name of the current function, as specified on the Response/Function List screen. The function must be associated with a user program.

This field is protected.

Drop code: Removes the task code from its association with the application.

Chapter 3. CA-ADS Dialog Compiler (ADSC)

- 3.1 Overview 3-3
- 3.2 Dialog compiler session 3-4
 - 3.2.1 Invoking the dialog compiler 3-4
 - 3.2.2 Sequencing through dialog compiler screens 3-5
 - 3.2.3 Suspending a session 3-8
 - 3.2.4 Terminating a session 3-9
- 3.3 Dialog compiler screens 3-10
 - 3.3.1 Main menu 3-10
 - 3.3.2 Options and Directives screen 3-13
 - 3.3.3 Map Specifications screen 3-17
 - 3.3.4 Database Specifications screen 3-20
 - 3.3.5 Records and Tables screen 3-23
 - 3.3.6 Process Modules screen 3-25

3.1 Overview

The CA-ADS dialog compiler is used to define dialogs for online and batch applications. Dialogs perform database retrieval and update, and any required processing within an application. Additionally, batch dialogs perform file input and output, and application processing.

When the definition is complete, the dialog is compiled and the resulting load module stored in the data dictionary. When using SQL, access modules are stored in the catalog component of the dictionary by the CA-IDMS access module compiler.

Modification or deletion of dialog components do not change the existing dialog until the dialog is explicitly recompiled to create a new load module.

►► For more information on modifying dialogs, see *CA-ADS User Guide*.

A dialog created by the dialog compiler can be associated with an application function or can stand alone as a structural unit in an application that consists only of dialogs. A dialog is associated with an application function by specifying the dialog name on the Response/Function List screen during an application compiler (ADSA) session. (See Chapter 2, “CA-ADS Application Compiler (ADSA)”)

Batch and online definition and execution modes: It is important not to confuse batch and online **definition** modes with batch and online **execution** modes. Batch dialogs and online dialogs can be **defined** using the dialog compiler in online or batch mode. The dialog compiler, ADSC, is the online dialog definition tool. ADSOBCOM, discussed in Appendix D, “Application and Dialog Utilities,” defines dialogs in batch mode. Once defined, dialogs can be **executed** in a batch environment or an online environment.

Process commands for online and batch: Process modules contained in dialogs can include process commands appropriate for online execution as well as commands designed exclusively for batch execution. The dialog compiler, when compiling a process module, accepts both types of commands, regardless of the environment of the dialog. This allows a process module to be used for both online and batch applications. If, however, the runtime system encounters a disallowed command or command parameter, the application abends.

Execution mode: The environment in which a dialog can be executed depends on the map associated with it, as follows:

- A dialog with an **online map** executes only in the online environment.
- A dialog with a **file map** executes only in the batch environment.
- A **mapless** dialog executes in either environment.

3.2 Dialog compiler session

In a dialog compiler session, screens are displayed that prompt the application developer for information about a dialog and the components with which the dialog is to be associated. The information supplied by the application developer is used by the CA-ADS runtime system to execute the dialog.

3.2.1 Invoking the dialog compiler

The application developer can invoke the dialog compiler from any of the three ways described below.

From CA-IDMS/DC or CA-IDMS/UCF: By specifying the appropriate CA-IDMS/DC or CA-IDMS/UCF (DC/UCF) task code, the application developer can invoke the dialog compiler. Task codes are defined at system generation and can vary from site to site. The default task code for the dialog compiler is ADSC. To use the dialog compiler under the transfer control facility, specify the transfer control facility version of the dialog compiler task code, ADSCT.

When invoked, the dialog compiler displays a blank Main Menu screen on which a new session can begin or a suspended session can be resumed.

►► For more information on using the Main Menu screen, see 3.3, “Dialog compiler screens” later in this section.

From another TCF task: By specifying the appropriate DC/UCF task code in conjunction with the SWITCH command from another task executing under the transfer control facility, the application developer can invoke the dialog compiler.

If a new session is requested, the dialog compiler displays a blank Main Menu screen on which a new session can begin or a suspended session resumed.

If an old session is requested, the dialog compiler resumes its most recently suspended session under the transfer control facility.

From the TCF Selection screen: By keying any nonblank character, except the underscore (_), next to the appropriate task code or descriptor, the application developer can invoke the dialog compiler as follows:

- Keying a nonblank character next to the appropriate task code invokes the dialog compiler, which displays a blank Main Menu screen on which a session can begin or be resumed.
- Keying a nonblank character next to the descriptor of a suspended dialog compiler session invokes the dialog compiler and resumes the suspended session at the Main Menu screen. The descriptor consists of the appropriate task code, the dialog name, and the dialog version number.

The transfer control facility enables the application developer to transfer from one DC/UCF task to another. For example, transfers between the dialog compiler, IDD, MAPC, and the application compiler can be made. When control is transferred from a task, the current session of that task is suspended, if necessary. A task can have several suspended sessions.

►► For a detailed description of the transfer control facility, see *CA-IDMS Transfer Control Facility*.

Note: Be sure to begin the dialog compiler session in the correct dictionary. The dictionary name can be specified in the Dictionary name field of the Main Menu screen.

Sample selections on the transfer control facility Selection screen are shown below:

```

COMPUTER ASSOCIATES INTERNATIONAL, INC.
TRANSFER CONTROL FACILITY          *** SELECTION SCREEN ***

_ SUSPEND TCF SESSION (PF9)      DBNAME..:      DBNODE..:
_ TERMINATE TCF SESSION (PF3)    DICTNAME: TSTDICT  DICTNODE:

          *TCF TASKCODES*
SELECT ONE TO START A NEW SESSION

_ TCF
_ SYSGENT  SYSGEN COMPILER
_ MAPCT    MAP DEFINITION
X ADSCT    DIALOG GENERATOR
_ ADSAT    APPLICATION GENERATOR
_ ASF
_ ASFT
_ IDDT     IDD COMMAND MODE
_ SSCT     SUBSCHEMA COMPILER
_ SCHEMAT  SCHEMA COMPILER
_ IDDMT    IDD MENU MODE
_ OLQ      OLQ COMMAND MODE
_ OLQT     OLQ COMMAND MODE

          *SUSPENDED SESSIONS*
SELECT ONE TO RESUME AN OLD SESSION
TASKCODE      DESCRIPTOR
_ ADSCT        MPKDIA1 0001
_ ADSAT        MPKAPP1 001
_ ADSAT        MPKAPP1 002
_ ADSCT        MPKDIA2 0001
_ OLMT         CEXME2220001

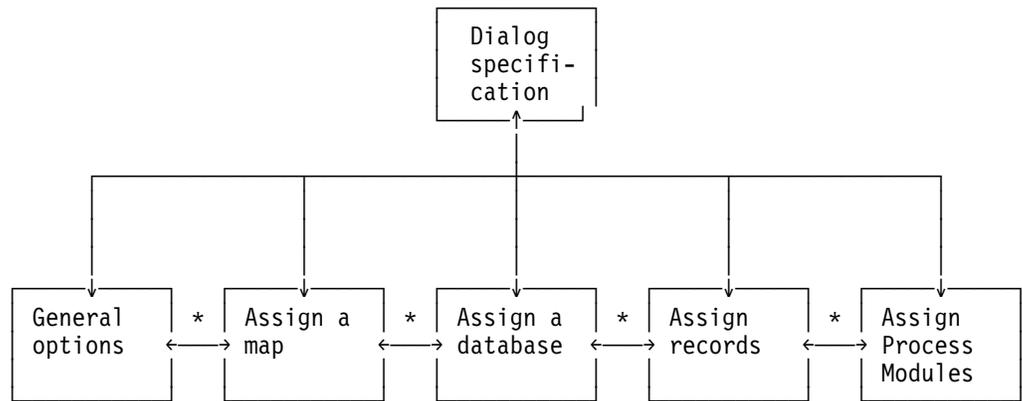
```

3.2.2 Sequencing through dialog compiler screens

Dialog compiler screens prompt the application developer for information about a dialog. The developer can sequence through the dialog definition steps or request a step in the process explicitly. A step in the definition process can contain more than one screen.

The primary steps involved in creating a dialog are shown below. The developer can either choose the next step from the Main Menu screen or move through the steps from screen to screen using [PF5].

Steps in creating a dialog



* Previous/next step (F4/F5)

Summary of dialog compiler process: Each step in the process of creating a dialog is associated with one or more screens as shown below.

Step in process	Screens	Purpose
Dialog specification	Main Menu	Identifies the name of a dialog and specifies the action to be taken
General options	Options and Directives	Specifies dialog options for activity logging, symbol and diagnostic table building, entry point, COBOL moves, retrieval locks, and autostatus capability
Assign maps	Map Specifications	Associates a map with the dialog, specifies paging options
Assign database	Database Specifications	Associates a schema and subschema or an access module with the dialog; identifies SQL options
Assign records and tables	Records and Tables	Associates work records with the dialog; specifies records for which new buffers are allocated when the dialog executes at runtime
Assign process modules	Process Modules	Associates a premap process, one or more response processes, and a declaration module with the dialog

Additional screens: The table below lists additional screens accessed through the **Display** and **Compile** windows on the action bar on the dialog compiler Main Menu.

Screen	Purpose
Map image	Displays a dialog's map as it appears to the terminal operator at runtime
Summary	Displays a summary listing of a dialog's components
Messages	Displays messages and errors encountered during the compilation process including errors in the source code for a premap or response process associated with a dialog
General options	Displays screens (when errors occur) listing modules, error browsing, and connections to IDD and DME

Control keys: While creating a dialog, the applications developer can use the control keys shown in the table below to:

- Move from one step in the process to another step
- Move from one screen to another screen while remaining on one step in the process
- Obtain help
- Leave the ADSC compiler
- Move between the action command line and the specification area (Main Menu only)

Default control keys

Activity	Control key	Description
HELP	[PF1]	Displays a map or field help screen, depending on cursor position If the cursor is on a map field associated with help text, a half screen of map field help text is displayed. If the cursor is set on a map field not associated with help text or anywhere else on the map, a full screen of map help text is displayed.
RETURN	[PF3]	From a pulldown window, returns to specification area. From the Main Menu screen, returns control to DC/UCF From a screen other than the Main Menu screen, applies updates to the current screen and returns to the Main Menu screen
BACKWARD	[PF4]	Applies updates to the current screen and displays the previous step in the process , as outlined on the Main Menu screen.
FORWARD	[PF5]	Applies updates to the current screen and displays the next step in the process , as outlined on the Main Menu screen.
BACKPAGE	[PF7]	Displays the previous screen of any step containing multiple screens.
FORWARD PAGE	[PF8]	Displays the next screen of any step containing multiple screens.
ACTION	[PF10]	Toggles the cursor position between the activity selection area action bar and the specification area on the Main Menu screen

3.2.3 Suspending a session

A dialog compiler session is automatically suspended in the event of a system crash.

Leaving ADSC automatically suspends the session. The developer can also suspend a session by selecting the **Release** option from the **Modify** window on the action bar. This allows any other developer to check the dialog out.

When a session is suspended, the application compiler saves the dialog definition, including all specifications made during the session, on queue records. A suspended session can be resumed at any time, as described in 3.2.1, “Invoking the dialog compiler” earlier in this section.

3.2.4 Terminating a session

When a session is terminated by compiling or deleting a dialog, the dialog compiler displays a blank Main Menu screen. The application developer can begin another session or can leave the dialog compiler by selecting an appropriate activity, such as **Switch**, from the action bar or by pressing [PF3].

3.3 Dialog compiler screens

3.3.1 Main menu

The Main Menu screen is displayed when the application developer initiates a dialog compiler session. This screen is used to specify the action taken regarding the dialog, name a dialog and dictionary, specify the next step to be taken in the dialog definition.

Areas: The screen is composed of six areas:

- Activity selection area
- Dialog identification area
- Screen specification area
- Message area
- Command area
- Key assignment area

Activity selection area: Displays the dialog compiler activities available.

The application developer selects an activity to be performed one of these ways:

- By typing the name of the activity on the Command line in the lower left-hand corner of the screen.
- By pressing [PF10] to reach the Activity Selection Area, and, with the tab key, positioning the cursor on the activity name and pressing [Enter].
 - ▶▶ See Chapter 1, “Introduction to CA-ADS” for a discussion of the activities available from the dialog compiler Main Menu screen.

Dialog identification area: Specifies the dialog name, dialog version number, the dictionary name, and the dictionary node. The fields contained in this section are described below.

Screen specification area: Allows the application developer to specify the next step in the definition process. The application developer can either:

- Press [Enter] to go to the default next step
 - Note:** See the table earlier in this chapter for information on the default dialog definition sequence.
- Specify a step

See Chapter 1, “Introduction to CA-ADS” for a discussion of the activities available from the application compiler Main Menu screen.

Message area: Displays informational and error messages returned from the dialog compiler.

Note that the control keys as described earlier in this section, (in addition to [Enter]) are identified at the bottom of this screen.

Command area: Provides a command line for entering the name of the desired action as specified in the activity selection area above. Action names can be abbreviated to the first three letters, ADD, MOD, DEL, COM, DIS or SWI. The system recognizes more than, but not less than, the first three letters of each identification.

If more than one activity is specified on the command line, an error message is displayed. If an activity is specified on the command line, and a control key is pressed, the activity associated with the control key is executed.

If an error is detected after the application developer selects an activity, the dialog compiler redisplay the current screen. The activity selection is retained and executed when the error is corrected. The application developer can override the initial selection by specifying another activity on the command line, selecting the activity directly from the selection area or by using [PF10].

Key assignment area: Presents the valid key choices and the action taken.

Control keys are described earlier in this section.

Main Menu screen

```

-
  Add Modify Compile Delete Display Switch
-
                                     CA-ADS Online Dialog Compiler
                                     Computer Associates International, Inc.
-
Dialog name . . . . . _____
Dialog version . . . . . _____
Dictionary name . . . . . _____
Dictionary node . . . . . _____
-
Screen . . . . . 1 1. General options
                  2. Assign maps
                  3. Assign database
                  4. Assign records and tables
                  5. Assign process modules
-
                  Copyright (C) 1972,2000 Computer Associates International, Inc.

Command ==>
Enter F1=Help F3=Exit F10=Action

```

Field descriptions:

Dialog name: Specifies the 1- to 8- character name of the current dialog. The dialog must begin with an alphabetic or national (@, #, and \$) character and cannot contain embedded blanks. A dialog name must be specified before any other dialog compilation activity can be executed. Once specified, the dialog name cannot be changed.

Dialog version: Specifies the version number, in the range 1 through 9999, of the current dialog. The default version is 1.

Dictionary name: Specifies the 1- to 8-character name of the data dictionary that contains the source modules and the map, access modules and subschema, and record definitions used by the specified dialog.

The dialog compiler stores the dialog load module in the specified dictionary when the dialog is compiled. If no dictionary name is specified, **dictionary name** is set to the name of the dictionary identified in the user's profile or set through a DCUF SET DICTNAME statement.

The dictionary name cannot change once it is validated.

Dictionary node: (DDS only) Specifies the node that controls the data dictionary specified by **Dictionary name**. **Dictionary node** defaults to the system currently in use.

Specifying a node name is equivalent to issuing a DCUF SET DICTNODE command under DC/UCF. The node name cannot change once it is validated.

Screen: Provides the application developer with a quick form of navigation between steps in the dialog definition process. By specifying the number which precedes the screen name, the user avoids unnecessary scrolling through the screens.

►► For a description of each screen, see 3.2.2, “Sequencing through dialog compiler screens” earlier in this section.

3.3.2 Options and Directives screen

The Options and Directives screen is used to specify options for a dialog, such as:

- Alternative message prefixes
- Autostatus
- Specifying the mainline dialog
- Including a symbol table
- Including a diagnostic table
- Specifying the premap as the entry point
- Using COBOL or CA-ADS rules in handling data types and arithmetic and assignment commands
- Activity logging
- Selectively disabling retrieval locks

The current settings for the dialog options are displayed on the screen. Each option can be changed by overwriting the displayed setting or by placing a slash (/) or other nonblank character in the space to the left of the option.

Sample screen

```

                                Options and Directives
                                Dialog JPKTD10  Version 1

Message prefix . . . . . DC
Autostatus record . . . . . ADSO-STAT-DEF-REC
Version . . . . . 1
Description . . . . . ADS DIALOG

Options and directives . . . . . _ Mainline dialog
                                / Symbol table is enabled
                                / Diagnostic table is enabled
                                / Entry point is premap
                                / COBOL moves are enabled
                                / Activity logging
                                / Retrieval locks are kept
                                / Autostatus is enabled

Enter F1=Help F3=Exit F4=Prev F5=Next

```

Field descriptions:

Dialog: Displays the name of the current dialog, as specified on the Main Menu screen. This field is protected.

Version: Displays the version number, in the range 1 through 9999, of the current dialog. This field is protected.

Message prefix: Specifies a 2-character prefix for a message at the dialog level. DC is the default prefix.

►► For more information on message prefixes, see Chapter 4, “CA-ADS Runtime System”

Autostatus record: Specifies the 1- to 32-character name of the status definition record used when the current dialog executes at runtime. The specified status definition record must be defined in the data dictionary. If no record name is specified, **Autostatus record** defaults to the name of the status definition record defined at DC/UCF system generation.

►► For more information on status definition records, see Chapter 10, “Error Handling.”

An autostatus record is required if the **Autostatus is enabled** option is chosen.

Version: Specifies a 1- to 4-digit version number, in the range of 1 through 9999, of the named status definition record. If a version number is not specified, **Version** defaults to the system default version number specified at system generation. If no system default version number is specified, **Version** defaults to 1.

Mainline dialog: Inserting a nonblank character in the accompanying data field specifies that the current dialog is a mainline dialog.

At runtime, the dialog that executes first in a series of dialogs that make up an application must be a mainline dialog. If a dialog function is initiated by an application task code, the dialog associated with the function must be a mainline dialog.

►► For more information on mainline dialogs, see Chapter 15, “Control Commands.”

Symbol table is enabled: Inserting a nonblank character in the accompanying data field specifies that a symbol table is created for a dialog.

A symbol table facilitates the use of element names and process line numbers by the online debugger.

►► For more information on online debugging, see Appendix H, “Debugging a CA-ADS Dialog.”

Diagnostic table is enabled: Inserting a nonblank character in the accompanying data field specifies that the dialog load module contains diagnostic tables (line number tables and offset tables).

During the testing of a dialog, the **Diagnostic table is enabled** option should be selected.

Diagnostic tables facilitate the testing and debugging of a dialog. If a dialog aborts, diagnostic tables are used to display the process command in error on the Dialog Abort Information screen. The ADSORPTS utility uses diagnostic tables to format the dialog report for easy reference.

Once a dialog has been tested thoroughly, the **Diagnostic table is enabled** option should be deselected and the dialog recompiled if dialog load module size is a consideration. The size of a large dialog load module can be reduced significantly by compiling the dialog without diagnostic tables.

The **Diagnostic table is enabled** option is deselected by spacing over the slash.

Note: The **Diagnostic table is enabled** option must be selected if the **Symbol table is enabled** option is selected.

Entry point is premap: The entry point of a dialog specifies the point at which the dialog becomes operative in the application thread.

Inserting a nonblank character in the accompanying data field specifies that the dialog begins with its premap process.

Regardless of the specification, a dialog without an online map or a batch input file map begins with its premap process. A dialog without a premap process begins with its first mapping operation.

COBOL moves are enabled: Inserting a nonblank character in the accompanying data field specifies that the rules of COBOL are used in the conversion between data types and in the rounding or truncation of the results of arithmetic and assignment commands.

►► A comparison of a CA-ADS MOVE and a COBOL MOVE is provided 13.4, “Assignment command.”

If COBOL moves are enabled, certain types of invalid expression may be allowed by the CA-ADS compiler. When a MOVE, COMPUTE, ADD, SUBTRACT, MULTIPLY, or DIVIDE statement has a numeric source expression and an EBCDIC target expression, the source expression must be a literal or a simple dataname with an optional subscript.

The default setting for COBOL MOVE is defined in the DC/UCF system generation ADSO statement. NO is the system generation default status.

The **COBOL moves are enabled** option can be modified only if the DC/UCF system generation COBOL MOVE subclause has been defined as OPTIONAL. OPTIONAL is the system default.

►► For more information on the COBOL MOVE subclause of the system generation ADSO statement, see *CA-IDMS System Generation*.

Activity logging: Inserting a nonblank character in the accompanying data field specifies that the dialog uses the activity logging facility. This facility documents all potential database activity by a dialog, based on the database commands issued explicitly or implicitly by the dialog's processes.

►► For more information on activity logging, see Appendix E, “Activity Logging for a CA-ADS Dialog.”

The default setting for the **Activity logging** option is defined at DC/UCF system generation.

►► For more information on the system generation ADSO statement, see *CA-IDMS System Generation*.

Retrieval locks are kept: Inserting a nonblank character in the accompanying data field specifies that database record retrieval locks will be held on behalf of run units started by the dialog.

Retrieval locks should be disabled only for retrieval dialogs that do not update the database or pass currencies to update dialogs. When retrieval locks are disabled for dialogs that *do* update the database or pass currencies, CA-ADS displays the following message:

```
DC173015  DIALOG ABORTED DUE TO VIOLATION OF NO RETRIEVAL LOCKING RULES
```

In addition, the update dialog abends when a higher dialog in the application thread does not have retrieval locks kept and system-wide RETRIEVAL NOLOCKS are specified.

The update dialog or program is allowed to update the retrieval dialog's database records when the dialog with retrieval locks turned off readies the area in UPDATE mode or when the update dialog or program does not receive currencies when control passes to it.

►► To avoid passing currency, see the TRANSFER command or the NOSAVE clause of the DISPLAY, INVOKE, or LINK commands in Chapter 15, "Control Commands."

Autostatus is enabled: Inserting a nonblank character in the accompanying data field specifies that the autostatus facility is to be used when the current dialog executes at runtime.

The initial setting corresponds to the autostatus specification defined at DC/UCF system generation. If autostatus is defined as optional, the application developer can override the initial setting. If autostatus is defined as mandatory, this field is protected and the initial setting cannot be changed.

►► For a discussion of the autostatus facility, see Chapter 10, "Error Handling."

3.3.3 Map Specifications screen

The Map Specifications screen is used to specify a map and map options for a dialog, such as the:

- Map name
- Method of map paging, including overriding automatic display of the first page of a pageable map

Sample screen

Map Specifications			
Dialog	JKPTD01	Version	1
Map name	_____	Input map	_____
Version	_____	Version	_____
Paging options	- 1. Wait	Label	_____
	- 2. No Wait	Output map	_____
	- 3. Return	Version	_____
Paging mode	- Update	Label	_____
	- Backpage	Suspense file label	_____
	- Auto display		
Enter F1=Help F3=Exit F4=Prev F5=Next F6=Switch Protection			

Field descriptions:

Dialog: Displays the 1- to 8- character name of the current dialog, as specified on the Main Menu screen. This field is protected.

Version: Displays the version number, in the range 1 through 9999, of the current dialog, as specified on the Main Menu screen. This field is protected.

Map name: Specifies the 1- to 8-character name of the map associated with the current dialog.

The specified map must be defined in the data dictionary. The map load module does not have to exist. If no map name is specified, only a premap process (not a response process) can be associated with the dialog.

Version: Specifies a 1- to 4-digit version number, in the range 1 through 9999, of the corresponding map.

If no version number is specified, version defaults to 1.

Paging options: Specifies the method used to determine the runtime flow of control when the user presses a control key during a map paging session.

►► For more information, see 17.6, “Pageable maps.”

Nowait is the default for pageable maps.

Note: The map paging dialog options **Nowait** and **Update** cannot be specified together.

Paging mode: Specifies parameters for a map paging session.

- **Update**

Specifies that the user can modify map data fields in a map paging session, subject to restrictions specified in the mapping facility and by the map modification process commands, as described in 17.6, “Pageable maps.” **Update** is the default for pageable maps.

Note: The map paging dialog options **Nowait** and **Update** cannot be specified together.

Note: When **Update** is not selected, all map data fields except \$RESPONSE and \$PAGE will be protected.

- **Backpage**

Specifies that a previous map pages can be displayed during a map paging session, as described in 17.6, “Pageable maps.” **Backpage** is the default for a pageable map.

- **Auto display**

Specifies an override of the automatic mapout of the first page of a pageable map.

When the **Auto display** option is not chosen, process logic must detect when the first page of the map is built and map out the first page when it is ready for display, even if the page is not full.

The \$PAGE-READY pageable map condition should be used to detect completion of the first page. The \$PAGE-READY condition should be tested while building the map page to determine when the page is ready for display.

►► For more information, see Chapter 8, “Conditional Expressions.”

Auto display is the default for pageable maps.

Input map: (CA-ADS/Batch only) Specifies that the named map is an input file map.

Note: Select Switch Protection [PF6] to unprotect the CA-ADS/Batch input fields.

Version: Specifies the version number, in the range 1 through 9999, of the current map. If no version number is specified, version defaults to 1.

Label: (CA-ADS/Batch only) Specifies the OS/390 ddname (VSE/ESA filename, BS2000/OSD linkname, VM/ESA ddname) of a batch dialog input file map. Specifications made in these fields can be overridden at runtime.

The runtime label for an input map can be specified only if the dialog is associated with an input file map.

Output map: (CA-ADS/Batch only) Specifies that the named map is an output file map.

Version: Specifies the version number, in the range 1 through 9999, of the current map. If no version number is specified, version defaults to 1.

Label: (CA-ADS/Batch only) Specifies the OS/390 ddname (VSE/ESA filename, BS2000/OSD linkname, VM/ESA ddname) of a batch dialog output file map. Specifications made in these fields can be overridden at runtime.

A runtime label for an output map can be specified only if the dialog is associated with an output file map.

Suspense file label: (CA-ADS/Batch only) Specifies the OS/390 ddname (VSE/ESA filename, BS2000/OSD linkname, VM/ESA ddname) of a batch dialog suspense file. Specifications made in these fields can be overridden at runtime.

The runtime label for a suspense file can be specified only if the dialog is associated with an input file map. A runtime label for a suspense file implicitly specifies that a suspense file is required for the dialog.

Map considerations: The specified map must be defined in the data dictionary. However, the map load module does not have to exist. If no map name is specified, only a premap process (not a response process) can be associated with the dialog.

The following rules apply to the environments in which a map can be executed.

- A dialog associated with an online map cannot be associated with an input or output file map.
- A dialog associated with an input or output map cannot be run in the online environment.
- A dialog can be associated with both an input and an output file map.
- If a dialog is not associated with any map, it is a mapless dialog and can be executed in both batch and online environments.

3.3.4 Database Specifications screen

The Database Specifications screen is used to specify database options for a dialog, such as the:

- Subschema name
- Access module name

Sample screen

```

                                Database Specifications
                                Dialog NAME1      Version      1
Subschema . . . . . _____
Schema . . . . . _____
Version . . . . . _____
Access Module . . . . . NAME1
SQL Compliance . . . . . _ 1. ANSI-standard SQL
                                   2. FIPS
Date Default Format . . . . . _ 1. ISO  2. USA  3. EUR  4. JIS
Time Default Format . . . . . _ 1. ISO  2. USA  3. EUR  4. JIS

Enter F1=Help F3=Exit F4=Prev F5=Next

```

Field descriptions:

Dialog name: Displays the name of the current dialog, as specified on the Main Menu screen. This field is protected.

Version: Specifies the version number, in the range 1 through 9999, of the current dialog. This field is protected.

Subschema: Specifies the 1- to 8-character name of the subschema associated with the current dialog.

The specified subschema must be defined in the data dictionary. If no subschema is specified, the dialog cannot perform database access.

Schema: Specifies the 1- to 8-character name of the schema with which the named subschema is associated.

If the named subschema is associated with more than one schema or version of a schema, a schema name must be specified. If the named subschema is associated with exactly one schema and version, **Schema** defaults to the name of that schema.

Version: Specifies the version number, in the range 1 through 9999, of the named schema. If no version number is specified, version defaults to the version of the named schema that was most recently defined.

Access module: Specifies the 1- to 8-character name of the access module associated with the current dialog. The access module need not exist when the dialog is compiled, but it must exist at runtime if the dialog accesses a database with SQL DML (other than dynamic SQL). If the dialog will not require an access module to be loaded at runtime, clear this field.

The dialog process logic can override the specification on this screen at runtime by issuing a SET ACCESS MODULE statement.

If you do not change the value in this field, the default value assigned by CA-ADS is the dialog name. If the dialog was copied from another dialog, the default value is:

- The name of the target dialog *if* the name of the access module name associated with the source dialog matches the name of the target dialog
- The name of the access module associated with the source dialog *if* the access module name does not match the name of the source dialog

About access modules: An access module is the executable form of the SQL statement that a program issues. When an access module is created, CA-IDMS/DB automatically determines the most effective access to the data requested by the SQL statements. The CA-IDMS access module compiler incorporates the access strategy in the access module, which is stored in the catalog component of the dictionary.

An access module is defined with a CREATE ACCESS MODULE statement in an SQL session, and it is associated with an SQL schema. It is built at runtime for the dialog if it is specified for the dialog on this screen and it has been created. Under CA-IDMS internal security, ownership of the schema qualifying the access module affects authority to use the access module.

►► For more complete information on creating and executing access modules, refer to:

- *CA-IDMS SQL Programming*
- *CA-IDMS SQL Reference*
- *CA-IDMS Security Administration*

SQL Compliance: Specifies the SQL standard you are enforcing. If you select neither ANSI-standard SQL nor FIPS, the default is CA-IDMS extended SQL.

►► For more information on SQL standards, refer to *CA-IDMS SQL Reference*.

Date Default Format: Specifies the external date representation format. The date format can be one of the following:

- **ISO** specifies the International Standards Organization standard
- **USA** specifies the IBM USA standard
- **EUR** specifies the IBM European standard
- **JIS** specifies the Japanese Industrial Standard Christian Era standard

Time Default Format: Specifies the external time representation format. The time format can be one of the following:

- **ISO** specifies the International Standards Organization standard

- **USA** specifies the IBM USA standard
- **EUR** specifies the IBM European standard
- **JIS** specifies the Japanese Industrial Standard Christian Era standard

►► For more information on date/time representations, refer to the *CA-IDMS SQL Reference*.

3.3.5 Records and Tables screen

The Records and Tables screen is used to associate a record with a dialog definition and to assign the **New** and **Work** record attributes.

New attribute: The **New** attribute identifies records for which new buffers are allocated and initialized when the dialog executes at runtime. Previously established buffers for records assigned the **New** attribute are retained but are not available to the dialog. A record that is assigned the **New** attribute must be known to the dialog as a subschema, map, or work record.

Work attribute: The **Work** attribute associates a record with a dialog as a work record. The CA-ADS runtime system allocates buffers for work records; in this way, records with the **Work** attribute establish working storage for a dialog. A record must be defined in the data dictionary before it can be associated with a dialog as a work record.

Records are dissociated from a dialog definition by:

- Placing a non-blank character in the **Drop** column opposite the record to be dissociated
- Overtyping the name of the record to be dissociated with the name of a new record

Up to 7 records can be specified on one page of the Records and Tables screen. Using the [PF8], additional pages are displayed.

Sample screen

Records and Tables			Page 1 of 1		
Dialog NAME1			Version 1		
Name	Version	Work	New copy	Drop	
1. AA	1	/	-	-	
2. AA	1	/	-	-	
3. _____	---	-	-	-	
4. _____	---	-	-	-	
5. _____	---	-	-	-	
6. _____	---	-	-	-	
7. _____	---	-	-	-	

DC498240 Record 2 is defined twice as a work record.

Enter F1=Help F3=Exit F4=Prev F5=Next F7=Bkwd F8=Fwd

Field Descriptions:

Dialog: Displays the name of the current dialog, as specified on the Main Menu screen. This field is protected.

Version: Specifies the version number, in the range 1 through 9999, of the current dialog. This field is protected.

Name: The 1- to 32-character name of each record assigned the **WORK** and/or **NEW COPY** attribute. Records associated with the dialog's map or subschema will be automatically associated with the dialog and need not be listed. If the dialog is to use its own copy of a record or if the dialog must distinguish between logical records or between a logical record and a database record, the required record or logical record can be named.

The value of the Name field can also be the 1- to 18-character schema name, followed by a period (.), followed by the 1- to 18-character table name of every table to be assigned as a host variable of an SQL command.

Version: Specifies the version number, in the range 1 through 9999, of the named record. The default version number is the system version default version number, as specified at system generation. If no system number is specified, the default version number is 1.

Work: Associates the **Work** attribute with the corresponding record. Records with the **Work** attribute are available to the dialog as working storage at runtime.

The application developer associates the **Work** attribute with a record by entering a nonblank character in the **Work** field corresponding to the applicable record.

If no attribute is specified when a record name is entered, **Work** is assigned as the default. If **New** is specified for a record, **Work** is automatically unassigned.

To remove the **Work** attribute from a record, the application developer places a nonblank character in the **Drop** column opposite the record to be dissociated.

New copy: Associates the **New** attribute with the corresponding record. Records with the **New** attribute are allocated new record buffers when the dialog executes at runtime.

The application developer associates the **New** attribute with a record by entering a nonblank character in the **New** field corresponding to the applicable record. To remove the **New** attribute from a record, the application developer places a nonblank character in the **Drop** column opposite the record to be dissociated.

Drop: Removes the record from its association with the dialog, but does not delete the record definition from the dictionary.

3.3.6 Process Modules screen

The Process Modules screen is used to associate a declaration, premap, response process, or default response process with a dialog.

Premap process: The Process Modules screen is used to associate or dissociate a premap process with a dialog. A premap process must exist in the data dictionary as a process module before it can be associated with a dialog.

Response process: A response process must exist in the data dictionary as a process module before it can be associated with a dialog. For a response process, the screen prompts the application developer for a control key and/or a response field value used to initiate the response process when the dialog executes at runtime.

If a batch dialog response field for an input record is the concatenation of several fields, the response field value specified on this screen must include any embedded blanks that occur in a concatenation.

The Process Modules screen allows the application developer to specify whether the response process is to be executed even if the map contains input errors. If map input errors are allowed, automatic editing is performed as usual for the dialog's map. The user is not required to correct errors before the response process begins execution. The response process is executed, but the erroneous data is not mapped in. The response process can test for map fields in error with an IF statement.

►► For a description of the IF command, see Chapter 14, "Conditional Commands."

►► For more information on automatic editing, refer to the *CA-IDMS Mapping Facility* manual. More than one control key or response field value can be associated with a response process. The application developer defines the response process repeatedly as a *new* response, each time specifying a different control key and/or response field value until all control keys and response field values to be associated with the response process have been specified. Note that the response process is compiled and stored in the dialog load module only once.

Declaration module: A declaration module is used under the SQL Option to declare cursors and to issue global WHENEVER statements. The statements in a declaration module are **not** executed. They are compiler directives used by the CA-ADS dialog compiler at dialog compilation.

Declaration modules allow you to store declarations you have specified as global to your application.

Unlike the premap and response process modules, the declaration module cannot contain executable CA-ADS commands. This module can contain only DECLARE CURSOR statements and WHENEVER directives.

A WHENEVER directive or DECLARE CURSOR statement is also valid in a premap or response process, but the scope of such a statement is not global.

►► Further explanation of usage for WHENEVER and DECLARE CURSOR can be found in the *CA-IDMS SQL Reference*.

►► For further considerations regarding the declaration module, see the *CA-IDMS SQL Programming*.

Default response: Specifies that the named process module is the optional default response process of the dialog. At runtime, after a mapin operation, the runtime system executes the default response process if no response process can be selected based on control event or response field value.

Dissociating a process: The Process Modules screen is also used to dissociate a response process from a dialog. The developer places a nonblank character next to **Drop** opposite the process to be deleted.

Multiple processes: Up to 4 processes can be specified on one page of the Process Modules screen. Using [PF8], you can display additional pages.

Compiling the process: When the application developer chooses **Compile** from the action bar in the activity selection area of the Main Menu screen, the dialog compiler compiles all processes associated with the dialog. ADSC returns the following message after a successful compile:

```
DC498140 Dialog TESTDIAL version 1 has been successfully compiled.
```

If a process does not compile successfully, the application compiler indicates the number of errors encountered.

The application developer can view the source code and error messages for the process by selecting item 2, **Display messages**, from the **Compile** window on the action bar in the activity selection area of the Main Menu screen.

►► For more information see Chapter 1, “Introduction to CA-ADS.”

The application developer cannot enter changes on the Messages screen.

►► For a description of the Messages screen see 1.5, “CA-ADS screens.”

To modify the process source code, the application developer must suspend the dialog compiler session.

►► For more information, see 3.2.3, “Suspending a session” earlier in this chapter.

Sample screen

Process Modules		Page	1 of	1
Name	_____	_____	Type	
Version	_____	_____	Execute on errors	
Key	_____	Value _____	Drop	
Name	_____	_____	Type	
Version	_____	_____	Execute on errors	
Key	_____	Value _____	Drop	
Name	_____	_____	Type	
Version	_____	_____	Execute on errors	
Key	_____	Value _____	Drop	
Name	_____	_____	Type	
Version	_____	_____	Execute on errors	
Key	_____	Value _____	Drop	

* Type : 1=Declaration 2=Premap 3=Response 4=Default Response
 DC498166 Neither a map nor premap are defined

Enter F1=Help F3=Exit F4=Prev F5=Next F7=Bkwd F8=Fwd

Field descriptions:

Name: Specifies the 1- to 32-character name of the module associated with the current dialog as a premap process, a response process, or a declaration module. The specified source module must exist in the data dictionary.

Version: Specifies the version number in the range 1 through 9999, of the named process source module. The default version number is the system version default version number, as specified at system generation. If no system number is specified at system generation, the default version number is 1.

Key: Specifies the online control key or batch control event that initiates the runtime response process. Valid control key specifications are:

- ENTER
- CLEAR
- PA1, PA2, PA3
- PF1 through PF24,
- FWD
- BWD
- HDR

Considerations

- FWD, BWD, and HDR can be specified only if the dialog is associated with a pageable map.
- LPEN can be specified as a control key if the use of light pens is supported by the installation.
- CLEAR, PA1, PA2, and PA3 do not transmit data; that is, input is not mapped in when these keys are pressed at runtime.
- The FWD, BWD, and HDR control keys are associated with pageable maps. FWD and BWD are synonymous with the keyboard control keys defined for paging forward and backward, respectively.

If FWD or BWD are specified and the keys defined for paging forward and backward are changed, the dialog does not have to be recompiled.

- HDR is not associated with any keyboard control key; rather, conditions encountered during a map paging session cause a response process associated with this control key value to be initiated.

►► For more information on the effect of HDR on the runtime flow of control, see Chapter 4, “CA-ADS Runtime System.”

Valid batch control events are as follows:

- **EOF** indicates the most recent input file read operation resulting in an end-of-file condition.
- **IOERR** indicates the most recent input file read operation resulting in a physical input-error condition. In CA-ADS/Batch, output errors cause the runtime system to terminate the application.

Value: Specifies a 1- to 32-character response name that can be entered in a \$RESPONSE map field to initiate the response process at runtime. The response field value can contain embedded blanks.

If the current dialog is associated with an application function, the application developer can associate a response process in the dialog with an application response. This is done by entering in the **Value** field the specification entered in the **Response Name** field of the Response Definition screen during application definition. Additionally, the same control key must be specified in the **Key** field on both the dialog compiler Process Modules screen and the application compiler Response/Function List screen.

By associating a dialog's response process with an application response, the application developer can place security restrictions on the response process. Additionally, the response process can be displayed as a valid response on runtime menus.

Type: Specifies the type of module named module is.

Execute on errors: Specifies that the response process executes even if the map contains input errors. Map fields in error are not mapped in. The map field status condition test can be used to test for fields in error, as described in Chapter 8, "Conditional Expressions."

When this option is not selected, the user must correct all map fields in error before processing continues.

Drop: Specifies that an existing process module is being dropped from the dialog definition.

If **Drop** is specified, the dialog compiler dissociates the process module from the dialog but does not delete the source from the data dictionary.

Chapter 4. CA-ADS Runtime System

4.1	Initiating the CA-ADS runtime system	4-3
4.1.1	How to define runtime tasks	4-3
4.1.2	How to start a CA-ADS application	4-4
4.2	Runtime menu and help screens	4-8
4.2.1	Menu screens	4-8
4.2.2	Site-defined menu maps	4-9
4.2.3	System-defined menu maps	4-10
4.2.4	Application help screen	4-16
4.3	Runtime flow of control	4-19
4.3.1	Effects of automatic editing on flow of control	4-22
4.4	Message prefixes	4-23
4.5	CA-ADS tasks, run units, and transactions	4-24
4.5.1	Run units and database access	4-25
4.5.2	Extended run units	4-25
4.6	Dialog Abort Information screen	4-28
4.7	Debugging a dialog	4-32
4.8	Linking From CA-ADS to CA-OLQ	4-33
4.8.1	Linking to CA-OLQ	4-33
4.8.2	Passing syntax to CA-OLQ	4-33
4.9	Linking built-in functions with the runtime system	4-34
4.9.1	Linking system-supplied built-in functions	4-34
4.9.2	Linking user-written built-in functions	4-39
4.10	Managing storage	4-40
4.10.1	Adjusting record compression	4-40
4.10.2	Calculating RBB storage	4-40
4.10.3	Writing resources to scratch records	4-41
4.10.4	Using XA storage	4-42

4.1 Initiating the CA-ADS runtime system

CA-ADS applications are executed using the CA-ADS runtime system.

To execute a CA-ADS/Batch application, use the CA-ADS/Batch runtime system.

►► For more information about the CA-ADS/Batch runtime system, refer to *CA-ADS Batch User Guide*.

4.1.1 How to define runtime tasks

Tasks that initiate the CA-ADS runtime system are defined at CA-IDMS/DC or CA-IDMS/UCF (DC/UCF) system generation to activate program ADSORUN1. The task codes are specified by means of the system generation TASK statement and are associated with CA-ADS in one of the following ways:

- By means of the ADSTASK clause of the system generation ADSO statement.
- By means of the application compiler Task Codes screen (described in Chapter 2, “CA-ADS Application Compiler (ADSA)”). At runtime, a task code defined on the Task Codes screen directly initiates the *application* for which it is defined.

A task code specified on the Task Codes screen can be used to initiate the *runtime system* only if one of the following conditions is met:

- If the task code specified on the Task Codes screen is also defined in the system generation TASK statement
- If the default ADSTASK task code (ADS) is entered in conjunction with the task code specification on the Task Codes screen

- By means of a task code invoking a mainline dialog provided the task code invokes ADSORUN1.

At runtime, the named dialog is directly initiated as the first dialog in an application consisting of a series of dialogs.

►► For more information on the system generation TASK statement, refer to *CA-IDMS System Generation*.

When the user initiates a CA-ADS application, the runtime system loads the required load modules into storage and sets up control blocks and record buffers for the application.

4.1.2 How to start a CA-ADS application

After signing on to DC/UCF, the user can initiate the CA-ADS runtime system by responding to the ENTER NEXT TASK CODE prompt, as follows:

- By entering the task code that directly initiates an application
- By entering the task code specified in the ADSTASK clause of the system generation ADSO statement, followed either by a task code defined for an application or by the name of a mainline dialog
- By entering only the task code specified in the ADSTASK clause of the system generation ADSO statement

Task Application Table: In either of the first two cases, the CA-ADS runtime system responds by checking the Task Application Table (TAT) for the specified task code. If the specified task code is in the TAT, the runtime system begins execution of the application with the function associated with the task code or with a signon function if one is specified for the application. The TAT is updated by the application compiler by using information entered on the Task Codes screen.

►► For a description of the Task Codes screen, see Chapter 2, “CA-ADS Application Compiler (ADSA).”

The TAT table can also be updated online using ADSOTATU, or in batch using ADSOBTAT.

►► For more information on ADSOTATU and ADSOBTAT, see Appendix D, “Application and Dialog Utilities.”

If the specified task code is not in the TAT, the runtime system checks for a mainline dialog whose name matches the task code. If the dialog exists, the runtime system begins execution of the dialog. If the dialog does not exist, the runtime system terminates abnormally.

If the user enters only the task code specified in the ADSTASK clause of the system generation ADSO statement, the runtime system responds by displaying the Dialog Selection screen.

The Dialog Selection screen displays a menu of the mainline dialogs available to the user. The user selects a dialog, and the runtime system begins execution of the dialog.

Parameters

ads-task-code

Specifies the task code defined in the ADSTASK clause of the system generation ADSO statement.

Ads-task-code must be defined in a system generation TASK statement to invoke program ADSORUN1. The default task code is ADS.

DIAlog=dialog-name

Specifies the name of a mainline dialog to begin execution as the first dialog in an application.

Note: There is no space between the keyword DIAlog and = or between = and *dialog-name*.

application-task-code

Specifies a task code defined for an application by means of the Task Codes screen.

If no signon function is specified, the function associated with the task code begins execution as the first function in the application. If a signon function is specified, the function associated with the task code begins execution after an acceptable signon is entered.

idms-dc/ucf-task-code

Specifies either a task code defined for an application by means of the Task Codes screen or the name of a mainline dialog.

Idms-dc/ucf-task-code must be defined in a system generation TASK statement to invoke program ADSORUN1.

NODe=node-name

Specifies the node that controls the data dictionary in which the definitions and load modules for the requested application are stored.

Node-name must be defined at DC/UCF system generation.

Note: There is no space between the keyword NODe and = or between = and *node-name*.

DBName=database-name

Specifies the database accessed by the application.

Database-name must be defined at system generation.

Note: There is no space between the keyword DBName and = or between = and *database-name*.

TRACE=

Specifies that the CA-ADS trace facility is to be used for the application.

Note: There is no space between the keyword TRACE and = or between = and ALL or CTL.

ALL

Writes trace records to the system log for each of the following:

- Dialog entry
- Process module entry
- Subroutine entry
- Process command execution for dialogs having symbol tables
- Database status information
- Currency save and restore operations

CTL

Writes the same trace records as ALL only for the following subset of process commands:

- Control commands
- Database commands

Examples

Example 1: Specifying a task code that directly initiates an application: The following statement initiates the run-time system with the system-generated task code REPORTS. Because REPORTS is defined in the TAT and no signon function is specified for the application that REPORTS initiates, the runtime system begins execution of the function associated with REPORTS.

```
REPORTS
```

Example 2: Specifying the run-time system task code without an application task code: The following statement initiates the run-time system with the default task code ADS. The system displays the Dialog Selection screen.

```
ADS
```

Example 3: Specifying the run-time system task code with an application task code: The following statement initiates the run-time system with the default task code ADS and specifies the application task code TESTAPPL:

```
ADS TESTAPPL DBNAME=TESTDB
```

The DBNAME clause is included to specify that the database to be accessed by the application is TESTDB. If the clause were not included in this statement, the application would access the system's current database or the dbname set by a DCUF SET DBNAME command.

4.2 Runtime menu and help screens

The CA-ADS runtime system builds and displays menu and help screens for application functions. These screens display valid responses for a function and allow the user to select a response.

Note: Online help does not support terminal access methods that do not provide the READ BUFFER functions (for example, VTAM does provide this function; TCAM does not). Terminals running under a method that does not support READ BUFFER are detected, and invocation of HELP at runtime is ignored.

Menu screens and the application help screen are discussed separately below.

4.2.1 Menu screens

Specifying a menu function: Functions are defined as menu functions or menu/dialog functions on the Response/Function List screen during application definition. The menu associated with a function can be further described by using the Function Definition (Menu) screen. The application developer can indicate on this series of screens whether the menu map is system-defined or site-defined and provide a heading for the menu map.

►► For descriptions of the screens used during application definition, see Chapter 2, “CA-ADS Application Compiler (ADSA).”

System-defined menu maps can be associated with menu functions or with menu/dialog functions. A site-defined menu map must be associated with a menu/dialog function.

ADSO- APPLICATION- MENU-RECORD: The CA-ADS runtime system uses ADSO-APPLICATION-MENU-RECORD to create menus. All menu maps must include ADSO-APPLICATION-MENU-RECORD as a map record.

►► For a description of ADSO-APPLICATION-MENU-RECORD, see Appendix A, “System Records.”

When the map of a menu function or menu/dialog function is mapped out, the runtime system initializes ADSO-APPLICATION-MENU-RECORD and moves as many responses and descriptions as possible into the fields provided by the menu map. If the menu is defined on the Function Definition (Menu) screen as a site-defined menu, the runtime system moves as many responses and descriptions as specified on the Function Definition (Menu) screen. If a response has no description, the runtime system displays the description for the function associated with the response. AMR-RESPONSE-FIELD is initialized with the default response specified for the function.

Selecting a response: When the menu is displayed on the screen, the user can select a response in one of the following ways:

- By pressing the control key associated with the applicable response
- By entering a nonblank character in the field immediately preceding the applicable response
- By entering a response name in the field that maps to AMR-RESPONSE-FIELD

The runtime system passes the selected response to the AMR- RESPONSE-FIELD of ADSO-APPLICATION-MENU-RECORD. If the user uses more than one type of response selection, the response selected by using a control key has precedence over a response selected by a nonblank character, which has precedence over a response name entered in a RESPONSE field. If the user presses the ENTER key without selecting a response, the default response remains in AMR-RESPONSE-FIELD and is considered in the determination of the runtime flow of control.

The value in AMR-RESPONSE-FIELD determines the next function to be executed from a menu function.

►► For more information on how the CA-ADS runtime system processes responses, see "Runtime Flow of Control" later in this section.

The CA-ADS runtime system processes system-defined and site-defined menu maps in the same way, as described below.

4.2.2 Site-defined menu maps

In order for the runtime system to perform this processing automatically, a site-defined menu map must have the following characteristics:

- The menu must map to ADSO-APPLICATION-MENU- RECORD. Fields in ADSO-APPLICATION-MENU-RECORD are described in Appendix A, "System Records."
- The number of responses specified per menu page must not exceed the number of occurrences defined for the AMR-SELECT-SECTION of ADSO-APPLICATION-MENU-RECORD (that is, 50). The application developer specifies the number of responses per page on the Function Definition (Menu) screen during application definition.
 - For a description of the Function Definition (Menu) screen, see Chapter 2, "CA-ADS Application Compiler (ADSA)."

Considerations: The following considerations apply:

- AMR-RESPONSE-FIELD can be used to display default responses and to accept a response from the user.

- Unless specifically specified, unused occurrences of the AMR-SELECT-SECTION are not protected on a site-defined menu map.
- The menu record always appears initialized to a site-defined menu dialog.

►► For more information on site-defined menu maps, see the *CA-ADS Application Design Guide*.

4.2.3 System-defined menu maps

CA-ADS provides three system-defined menu maps, as follows:

- **ADSOMUR1** — The short description menu map
- **ADSOMUR2** — The long description menu map
- **ADSOMSON** — The signon menu map

The format for a system-defined nonsignon menu map is specified on the Function Definition (Menu) screen during application definition. The application developer can select a short description format (ADSOMUR1) or a long description format (ADSOMUR2) for nonsignon menus.

Short description format: The short description format displays 30 responses per menu page; the long description format displays 15 responses per menu page. The number of responses that are displayed can be modified on the Function Definition (Menu) screen by specifying that the menu is site-defined and by entering the number of responses per page.

►► For more information on specifying the format for a menu map, see Chapter 2, “CA-ADS Application Compiler (ADSA).”

Runtime display: When a menu screen is displayed at runtime, ADS builds the menu by storing the appropriate information in ADSO-APPLICATION-MENU-RECORD. System-defined nonsignon menus map to all but two of the fields in ADSO-APPLICATION-MENU-RECORD.

►► For descriptions of the fields in ADSO-APPLICATION-MENU-RECORD, see Appendix A, “System Records.”

Sample short description menu screen (ADSOMUR1)

DIALOG:		PAGE: 1 OF: 1
DATE:		NEXT PAGE:
	TEST APPLICATION	
_ PAGETEST (PF22)	TEST DIALOG	_ FORWARD (PF8) FORWARD
_ BACKWARD (PF7)	BACKWARD	_ POPTOP (PF3) POP TO TOP
_ HELP (PF1)	HELP	_ QUIT (PF24) QUIT
_ POP (PF2)	POP 1 LEVEL	_ LINKMENU (PF13) LINKTO SUBMENU
_ LINKPAGE (PF14)	LINK TO DIALOG	_ START
_ EDIT (PF17)	TESTING BUG FIX	START OF LIST
RESPONSE:	SEND DATA-->	MODE: STEP

Sample long description menu screen (ADSOMUR2)

DIALOG:		PAGE: 1 OF: 1
DATE: 11/27/99		NEXT PAGE:
	TEST APPLICATION	
_ PAGETEST (PF22)	TEST DIALOG	
_ FORWARD (PF8)	FORWARD	
_ BACKWARD (PF7)	BACKWARD	
_ POPTOP (PF3)	POP TO TOP	
_ HELP (PF1)	HELP	
_ QUIT (PF24)	QUIT	
_ POP (PF2)	POP 1 LEVEL	
_ LINKMENU (PF13)	LINKTO SUBMENU	
_ LINKPAGE (PF14)	LINK TO DIALOG	
_ START	START OF LIST	
_ EDIT (PF17)	TESTING BUG FIX	
RESPONSE:	SEND DATA-->	MODE: STEP

Field descriptions:

DIALOG: Specifies the name of the dialog associated with the current menu/dialog function. The field is blank if no dialog is associated with the current function.

This field is protected.

DATE: Specifies the current date in the format selected on the Main Menu screen during application definition.

This field is protected.

PAGE: Specifies the current page of the menu screen.

This field is protected.

OF: Specifies the total number of pages for the current menu.

This field is protected.

NEXT PAGE: Specifies the next page of the menu screen to be displayed. To page forward or backward, the user enters the applicable page number and presses the ENTER key. The FORWARD and BACKWARD system functions can also be used if they are valid for the current function.

HEADING TEXT: Displays the heading for the current menu, as specified on the Function Definition (Menu) screen during application definition.

RESPONSE LISTING: Displays the available valid responses for the current function and, for each response, provides a 1-byte field that the user can use to select the response. A short description menu displays 30 responses per page; a long description menu displays 15 responses per page.

The responses are listed in the order specified on the Function Definition (Menu) screen during application definition. For each response listed, the following information, which is supplied on the Response/Function List screen during application definition, is displayed:

- The response name.
- The control key associated with the response.
- The response description.

If the menu has a short description format, the description text is truncated to 12 bytes. If the menu has a long description format, the entire 28-byte description is displayed. If the response has no description, the description for the function associated with the response is displayed.

SYSTEM MESSAGE AREA: Displays informational and error messages returned by the CA-ADS runtime system.

This area is protected.

RESPONSE: Specifies the default response (if any) for the current function.

The user can select the default response by pressing the ENTER key without modifying the screen.

The user can select a different response than the default response by overwriting the default response with a nonblank character in the 1-byte field preceding the applicable response, or by pressing the control key associated with the response. If an invalid response name is entered, the value is replaced by the default next response.

SEND DATA: Specifies a 32-byte field that is mapped to the AMR-PASSING field of ADSO-APPLICATION-MENU-RECORD and then moved to the AGR-PASSED-DATA field of ADSO-APPLICATION-GLOBAL-RECORD. The AGR-PASSED-DATA field can be accessed in the process code of a dialog function or user program. AMR-PASSING is initialized to spaces before the menu is mapped out. If AMR-PASSING contains all spaces, nothing is moved to AGR-PASSED-DATA.

MODE: Specifies an execution mode of STEP or FAST for the function. This specification is valid only if the application developer coded procedures for controlling the execution mode of the current dialog function.

Signon menu maps: The application developer defines a menu as a signon menu on the application compiler Function Definition (Menu) screen.

Signon menus are similar to nonsignon menus, except that signon menus map to two additional fields (AMR-USER-ID and AMR-PASSWORD) in ADSO-APPLICATION-MENU-RECORD. If the system function SIGNON is associated with a valid response for the signon menu function, the runtime system submits the values entered in the AMR-USER-ID and AMR-PASSWORD fields to DC/UCF for security clearance when SIGNON is initiated.

If the return code from DC/UCF indicates a successful signon, CA-ADS moves the value in AMR-USER-ID to the AGR-USER-ID field of ADSO-APPLICATION-GLOBAL-RECORD. The AMR-PASSWORD field is overwritten with blanks after being passed to DC/UCF.

If a signon is required, the runtime system does not allow any other application activity to occur until a successful signon is processed.

►► For information about ADSO-APPLICATION-MENU-RECORD and ADSO-APPLICATION-GLOBAL-RECORD, see Appendix A, “System Records.”

For more information on implementing a signon procedure, see Appendix G, “Security Features.”

Signon menu maps can be site-defined or system-defined.

Sample signon menu screen (ADSOMSON)

```

DIALOG:                                     PAGE: 1 OF: 1
DATE: 11/27/99                             NEXT PAGE:

                                     TEST APPLICATION

ENTER USER ID--->
PASSWORD----->

- PAGETEST (PF22) TEST DIALOG
- FORWARD (PF8) FORWARD
- BACKWARD (PF7) BACKWARD
- POPTOP (PF3) POP TO TOP
- HELP (PF1) HELP
- QUIT (PF24) QUIT
- POP (PF2) POP 1 LEVEL
- LINKMENU (PF13) LINKTO SUBMENU
- LINKPAGE (PF14) LINK TO DIALOG
- START START OF LIST
- EDIT (PF17) TESTING BUG FIX

RESPONSE: SEND DATA--> MODE: STEP

```

Field descriptions:

DIALOG: Specifies the name of the dialog associated with the current menu/dialog function. The field is blank if no dialog is associated with the current function.

This field is protected.

DATE: Specifies the current date in the format selected on the Main Menu screen during application definition.

This field is protected.

PAGE: Specifies the current page of the menu screen.

This field is protected.

OF: Specifies the total number of pages for the current menu.

This field is protected.

NEXT PAGE: Specifies the next page of the menu screen to be displayed. To page forward or backward, the user enters the applicable page number and presses the ENTER key. The FORWARD and BACKWARD system functions can also be used if they are valid for the current function.

HEADING TEXT: Displays the heading for the current menu, as specified on the Function Definition (Menu) screen during application definition.

ENTER USER ID: Prompts for the user's user id.

This 32-byte field is mapped to the AMR-USER-ID field of ADSO-APPLICATION-MENU-RECORD.

PASSWORD: Prompts for the user's password.

This 8-byte field is mapped to the AMR-PASSWORD field of ADSO-APPLICATION-MENU-RECORD. This is a darkened field; characters entered in this field do not appear on the screen.

RESPONSE LISTING: Displays the available valid responses for the current function and, for each response, provides a 1-byte field that the user can use to select the response.

The signon menu screen displays 12 responses per page. The responses are listed in the order specified on the Function Definition (Menu) screen during application definition.

For each response listed, the following information, which is supplied on the Response/Function List screen during application definition, is displayed:

- The response name.
- The control key associated with the response.
- The 28-byte response description.

If a response has no description, the description for the function associated with the response is displayed.

Note: If a signon is required for the application, at least one valid response must be associated with the system function SIGNON.

SYSTEM MESSAGE AREA: Displays informational and error messages returned by the CA-ADS runtime system.

This area is protected.

RESPONSE: Specifies the default response (if any) for the current function.

The user can select the default response by pressing the ENTER key without modifying the screen.

The user can select a different response than the default response by overwriting the default response, by entering a nonblank character in the 1-byte field preceding the applicable response, or by pressing the control key associated with the response.

SEND DATA: Specifies a 32-byte field that is mapped to the AMR-PASSING field of ADSO-APPLICATION-MENU-RECORD and then moved to the AGR-PASSED-DATA field of ADSO-APPLICATION-GLOBAL-RECORD. The AGR-PASSED-DATA field can be accessed in the process code of a dialog function

or user program. AMR-PASSING is initialized to spaces before the menu is mapped out. If AMR- PASSING contains all spaces, nothing is moved to AGR-PASSED-DATA.

MODE: Specified an execution mode of STEP or FAST for the function. If the user specifies an acceptable signon and the execution mode is STEP, the runtime system redisplay the signon menu. The user must press the ENTER key to proceed to the first application function.

If the execution mode is FAST, the runtime system immediately proceeds to the first function. Except for its use in signon menus, the execution mode specification is valid only if the application developer coded procedures for controlling the execution mode of the current dialog function.

4.2.4 Application help screen

The runtime application help screen lists all the valid responses for the current function. The screen is displayed when the user selects a response that initiates the system function HELP.

The user can perform any of the following actions from the application help screen:

- Select another page for display by entering the applicable page number in the NEXT PAGE field
- Select a response, as follows:
 - By pressing the control key associated with the applicable response
 - By entering a nonblank character in the 1-byte field immediately preceding the applicable response name
 - By entering the applicable response name in the RESPONSE field
- Return to the current function by pressing the ENTER key without modifying the screen

Sample application help screen

CURRENT FUNCTION: DUDMENU	PAGE: 1 OF: 1
DATE: 11/27/99	NEXT PAGE:
APPLICATION CONTROL FACILITY HELP SCREEN	
- PAGETEST (PF22)	TEST DIALOG
- QUIT (PF24)	QUIT
- HELP (PF1)	HELP
- FORWARD (PF8)	FORWARD
- BACKWARD (PF7)	BACKWARD
- POPTOP (PF3)	POP TO TOP
- POP (PF2)	POP 1 LEVEL
- LINKMENU (PF13)	LINKTO SUBMENU
- LINKPAGE (PF14)	LINK TO DIALOG
- START	START OF LIST
- EDIT (PF17)	TESTING BUG FIX
RESPONSE:	

Field descriptions:

CURRENT FUNCTION: Specifies the name of the function for which the listed responses are valid.

This field is protected.

DATE: Specifies the current date in the format selected on the General Options screen during application definition.

This field is protected.

PAGE: Specifies the current page of the help screen.

This field is protected.

OF: Specifies the total number of pages for the current help screen. This field is protected.

NEXT PAGE: Specifies the next page of the help screen to be displayed. To page forward or backward, the user enters the applicable page number and presses the ENTER key. The FORWARD and BACKWARD system functions cannot be used to page through this screen.

RESPONSE LISTING: Displays the valid responses for the current function and, for each response, provides a 1-byte field that the user can use to select the response. The application help screen displays 15 responses per page.

For each response listed, the following information, which is supplied on the Response Definition screen during application definition, is displayed:

- The response name.
- The control key associated with the response.
- The 28-byte response description.

If a response has no description, the description for the function associated with the response is displayed.

SYSTEM MESSAGE AREA: Displays informational and error messages returned by the CA-ADS runtime system.

This area is protected.

RESPONSE: Specifies a response name entered by the user.

4.3 Runtime flow of control

Flow of control is the way control is passed from one application function or dialog to another at runtime. In CA-ADS, the runtime flow of control is determined by user requests or runtime events, based on specifications made at definition time.

AGR-CURRENT- RESPONSE: The CA-ADS runtime system uses the AGR-CURRENT-RESPONSE field of ADSO-APPLICATION-GLOBAL-RECORD to direct the flow of control in applications defined by using the application compiler.

When the user presses a control key, the value of AGR-CURRENT-RESPONSE is established by means of the following steps:

1. The runtime system moves spaces to AGR-CURRENT- RESPONSE.
2. The runtime system checks the AGR-MAP-RESPONSE field of ADSO-APPLICATION-GLOBAL-RECORD (AMR- RESPONSE-FIELD of ADSO-APPLICATION-MENU- RECORD for menu functions) for a response entered by the user. If the user entered a response and pressed the ENTER key, the runtime system moves the response to AGR-CURRENT-RESPONSE. If the user pressed a control key other than the ENTER key, the runtime system proceeds to Step 4 below.
3. If the user did not enter a response, the runtime system checks the AGR-DEFAULT-RESPONSE field of ADSO-APPLICATION-GLOBAL-RECORD for a default response for the current function. If a default response exists and the user pressed the ENTER key, the runtime system moves the default response to AGR-CURRENT-RESPONSE. If a default response does not exist or if the terminal operator did not press the ENTER key, the runtime system proceeds to Step 4 below.
4. If a default response does not exist or if the user did not press the ENTER key, the runtime system checks the AGR-AID-BYTE field for the control key pressed by the user. If the control key pressed is associated with a response, the runtime system moves the associated response to AGR- CURRENT-RESPONSE. If the control key pressed is not associated with a response, spaces remain in AGR-CURRENT- RESPONSE.

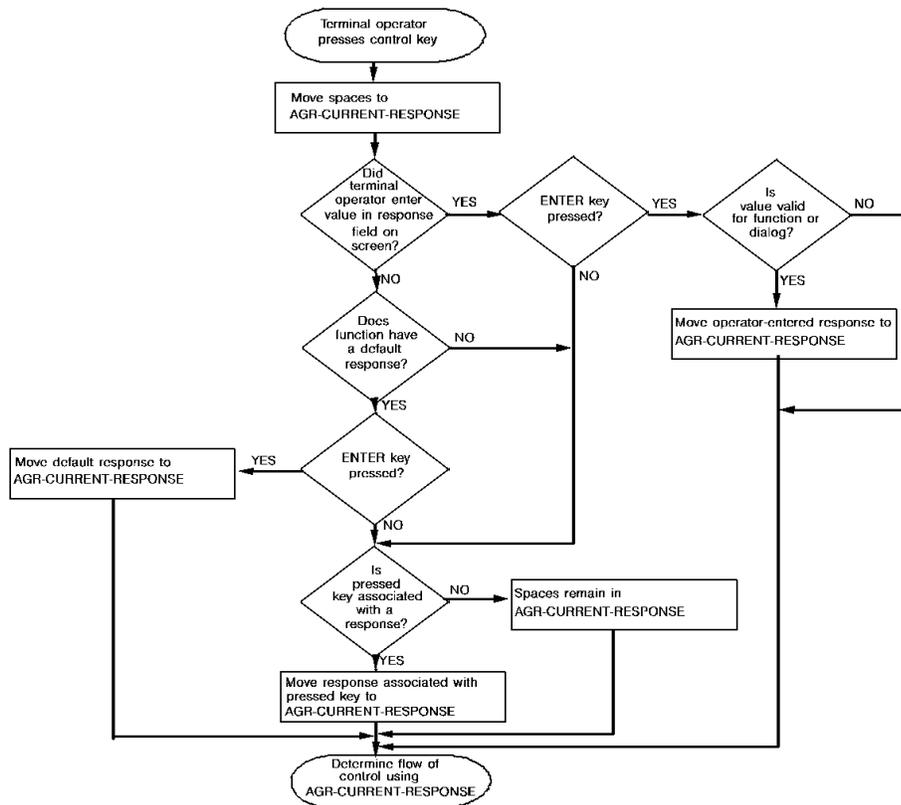
The following diagram shows how the runtime system establishes the value of AGR-CURRENT-RESPONSE.

►► For more information on the fields in ADSO-APPLICATION-GLOBAL-RECORD, see Appendix A, “System Records.”

Note: When a series of dialogs that are not associated with application functions is executed as an application, the flow of control is directed by the control commands coded in the premap and response processes. The control commands specify either explicitly or implicitly the next component to be executed.

For more information on the CA-ADS control commands, see Chapter 15, “Control Commands.”

Establishing the value of AGR-CURRENT-RESPONSE



Valid response: If the response established in AGR-CURRENT-RESPONSE is valid for the current function, the runtime system moves the name of the function associated with the response to the AGR-NEXT-FUNCTION field of ADSO-APPLICATION-GLOBAL-RECORD.

For responses with a security class higher than zero, the runtime system also checks whether the terminal operator has an acceptable security class. If the user does not have an acceptable security class, the current screen is redisplayed with a message indicating that a different response must be selected.

Note: Process code can move values to the AGR-CURRENT-RESPONSE field, overwriting the response selected by the user. The runtime system does not check security for a response moved to the AGR-CURRENT-RESPONSE field in process code. A process code value is executed if it is valid for the current function.

The response moved to AGR-CURRENT-RESPONSE establishes the next function to be executed. The function is not executed, however, until the runtime system satisfies

►► For more information on defining default control key assignments, see the discussion of the KEYS statement in *CA-IDMS System Generation*.

Default control key assignments are overridden by control key assignments specified for application responses and dialog response processes.

4.3.1 Effects of automatic editing on flow of control

Runtime flow of control is altered when the automatic editing capability of the DC/UCF mapping facility encounters input edit errors on mapin:

Response process selected: If a response process is selected, the outcome depends on whether the **Execute on edit errors** option for the response is selected:

- When it is selected, the response process is executed.
- When it is not selected, the response process is *not* executed. The next event depends on the control key pressed by the user:
 - If the user presses [Clear] or [PA1], the CA-ADS runtime system passes control using the sysgen-defined assignment for the key. This means that it overrides the application-defined assignment (if any) for the key.

Note: Required fields are always marked in error when the user presses [Clear] or any PA key.

- If the user presses any other control key, the runtime system redisplay the map, with edit errors.

►► For more information on execute on edit errors, see Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

System function selected: If a system function, except RETURN or TOP, is selected, the function is executed.

Any other application function: If any other application function, including RETURN or TOP is selected, the map is redisplayed with edit errors.

Considerations: Under certain circumstances, a dialog response process is selected even though the user has selected an application function, as indicated in diagram above. In these cases, the **Execute on edit errors** option of the selected response process determines whether the map with errors is redisplayed. Circumstances under which a response process is selected are as follows:

- The dialog has a response process associated with the ENTER key and the user selects a nonimmediately executable function (POP, POPTOP, RETURN, TOP, or QUIT). The ENTER response process is selected.
- The user selects a nonimmediately executable function and the control key pressed or response name specified by the user is the same as a control key or a response field value associated with a response process. The response process is selected.

4.4 Message prefixes

In CA-ADS, messages can be sent to a terminal in either of two ways:

- The dialog process code can issue a DISPLAY MESSAGE command
- Automatic editing can display a message for every field marked IN ERROR

Specific prefixes can be designated for each message.

Messages issued through DISPLAY MESSAGE command: A prefix can be specified through ADSC in either of two ways:

- In dialog process code in the DISPLAY command
- At the dialog level on the Options and Directives screen

If the message prefixes defined at the dialog level and at the message level conflict, the prefix set at the message level is used. If no prefix is set, 'DC' is used.

Messages issued through automatic editing: A message prefix can be specified through the mapping compiler in either of two ways:

- At the map level on the **General Options** screen
- At the map field level on the **Additional Edit Criteria** screen

The map message prefix is set only at the field level. The map level value entered on the screen is just a default carried to each field during a computation.

If the message prefixes defined at the map level and at the map field level conflict, the prefix set at the map level is used. If no prefix is set, 'DC' is used.

4.5 CA-ADS tasks, run units, and transactions

Tasks and run units opened when accessing a non-SQL defined database are handled automatically during the execution of a CA-ADS application. Tasks and run units for CA-ADS are discussed separately below.

Tasks: A task is a logical unit of work performed by the DC/UCF system that consists of one or more programs.

The CA-ADS runtime system executes as a series of tasks within the DC/UCF environment. The first task begins when the user initiates the runtime system, as discussed in 4.1, “Initiating the CA-ADS runtime system” earlier in this section. Subsequent tasks begin on mapin from the terminal.

A task terminates when the runtime system performs a mapout operation to the terminal with no errors or when the application terminates. When a task terminates, CA-ADS returns control to DC/UCF automatically; the application developer does not code a DC RETURN command.

After a mapin operation, CA-ADS determines whether the response entered by the user is valid. If the response is valid, the task continues and the runtime system resumes processing as directed by the response. If the response is not valid, the task terminates and the runtime system performs a new mapout operation with an error message.

Run units: Communication with the database is established by means of run units. A run unit begins when an application signals its intent to perform database operations and ends when the program releases all database resources from its control. A run unit can consist of any number of CA-IDMS/DB database requests.

CA-ADS can have 0 to 2 run units open at a time. With SQL access, run units are a physical aspect of data access that is hidden, as the SQL model requires. CA-ADS can have a network run unit open and access the database using SQL at the same time.

If a dialog issues non-SQL DML and SQL DML against the same non-SQL defined database at one time, deadlock of the run units is possible.

Establishing a run unit to access the database and extending run units using CA-ADS is discussed below.

Transactions: A database transaction is a unit of recovery within an SQL session.

CA-IDMS/DB begins a database transaction when the dialog submits an SQL statement that results in access to either user data or the dictionary, and ends a transaction when a COMMIT or ROLLBACK is executed or when the SQL session is terminated.

►► For information on transactions within an SQL session, see *CA-IDMS SQL Programming*.

For a list of SQL statements that start and end a database transaction, see *CA-IDMS SQL Reference*.

4.5.1 Run units and database access

During the execution of a CA-ADS application, the following sequence occurs when accessing the database:

1. The run unit begins and READY commands are automatically issued when the CA-ADS runtime system encounters the first database or logical record command that accesses database records.
2. When READY commands are physically coded in process modules, the following considerations apply:
 - The parameters from the last physically coded READY command for an area are used by the runtime system.
 - If no READY command appears in the process code, the default parameters, as defined in the subschema, are used by the runtime system.
3. The run unit that is not extended ends when the CA-ADS runtime system encounters any control command except RETURN in a nested structure.

Before executing the control command, CA-ADS does the following:

- Saves currencies unless additional specifications (NOSAVE, NOFINISH) indicate otherwise
- Issues a FINISH command to release the database areas and write a checkpoint to the CA-IDMS/DB journal

Finishing SQL transactions: An SQL transaction is finished only when the user explicitly terminates it (using the appropriate SQL commands), or when CA-ADS is terminating the task (such as DISPLAY or LEAVE ADS). A network run unit can be closed and re-opened because of a change of subschema causing CA-ADS not to extend a run unit. If CA-ADS has to finish such a run unit, it does **not** finish the SQL transaction.

4.5.2 Extended run units

A run unit is kept open (extended) when a dialog passes control to another dialog, user program, or application function by using an INVOKE, LINK, TRANSFER, or EXECUTE NEXT FUNCTION command.

►► These commands are documented in Chapter 15, “Control Commands.”

A run unit is extended when control passes to any one of the following:

- A user program.

If a run unit is not already open and the LINK command's USING RECORDS list includes SUBSCHEMA-CONTROL, CA-ADS opens and extends a run unit. If the run unit is already open, the run unit is extended.

- A dialog with a premap process and no associated subschema.
- A dialog with a premap process whose schema and subschema are the same as those of the issuing dialog and whose usage modes are equally or less restrictive than those of the linking dialog.

A usage mode is considered more restrictive than another usage mode if either of its two components is more restrictive.

The following table shows the relative restrictiveness of usage modes.

Restrictiveness	Usage mode	Qualifier
Most restrictive	Update	Exclusive
		Protected
	Retrieval	Shared
Least restrictive	Noready	

- Any lower-level dialog, provided that the USING SUBSCHEMA-CONTROL clause of the LINK command is used.

Considerations: The following considerations apply to extended run units:

- **Rollback when a run unit is extended with the LINK command**

The LINK command does not automatically write a checkpoint to the CA-IDMS/DB journal file. This allows a lower level dialog to check for errors and issue a ROLLBACK command if necessary. In this case, the entire extended run unit is rolled back.

If a COMMIT command is included in either dialog, the dialog is rolled back only to the COMMIT checkpoint. In this case, the entire extended run unit is not rolled back.

- **Runtime deadlocks**

- If a user program issues a subschema BIND followed by any database activity, the program can deadlock at runtime. To avoid this situation, a COMMIT command should be coded before a LINK to a user program that issues a BIND or FINISH command.

It may be more efficient to remove subschema BIND and FINISH activities from the user program and allow the extended run unit to handle these functions. In this case, the issuing dialog must pass SUBSCHEMA-CONTROL to the program. Subschema records passed to the program must be bound only if the user program provides its own subschema record buffers. When control returns to CA-ADS, the runtime system automatically rebinds the record buffers.

- If a dialog issues non-SQL DML and SQL DML against the same non-SQL defined database at one time, deadlock of the run units is possible.

- **Extending SQL transactions**

If a dialog links to a lower level dialog after beginning an SQL transaction (where the lower level dialog also issues SQL commands), the developer must either:

1. Issue an SQL COMMIT WORK command before linking to the lower level dialog, **or**
2. Compile the RCMs for the two dialogs into a single access module (AM).

Choice one results in two units of recoverable work; choice two results in a single recoverable unit of work. When neither one or two are done, the SQL request of the lower level dialog fails, because its RCM information is not found in the active AM.

4.6 Dialog Abort Information screen

When a dialog abends at runtime, the CA-ADS runtime system can display a diagnostic screen. The display of the diagnostic screen is enabled and disabled by using the DIAGNOSTIC SCREEN clause of the DC/UCF system generation ADSO statement.

►► For more information on the ADSO statement, refer to *CA-IDMS System Generation*.

If the diagnostic screen is enabled when an abend occurs at runtime, error messages are sent to the system log and the Dialog Abort Information screen is displayed. If the diagnostic screen is not enabled when an abend occurs, error messages are sent to the system log and the DC/UCF prompt ENTER NEXT TASK CODE is displayed with the following message:

```
ERROR OCCURRED DURING PROCESSING. CA-ADS DIALOG ABORTED.
```

The id of the above message is DC466019; the application developer can change the message text by using IDD.

Sample Dialog Abort Information screen

```

CA-ADS RELEASE 15.0          *** DIALOG ABORT INFORMATION ***  ABRT
DC173008 APPLICATION ABORTED. BAD IDMS STATUS RETURNED; STATUS=0306

DATE.....: 99.241          TIME.....: 15:12:29.08          TERMINAL.....: LV81004

ERROR OCCURRED IN DIALOG.....: DIALOG1
                        AT OFFSET.....: 310
                        IN PROCESS.....: DIALOG1-PREMAP          VERSION:      1
                        AT IDD SEQ NO. : 00000200

SEQUENCE
NUMBER:          SOURCE :
00000100 IF FIRST-TIME
00000200 FIND CURRENT EMPLOYEE.
00000300 DISPLAY.

HIT ENTER TO RETURN TO DC OR ENTER NEXT TASK CODE:

```

Field descriptions:

DATE: Specifies the date on which the dialog abended.

TIME: Specifies the time at which the dialog abended.

The date and time aid in locating the snap dump, if any, for the abend in the print log file.

TERMINAL: Specifies the logical terminal at which the abend occurred.

DIALOG: Specifies the name of the aborted dialog.

OFFSET: Specifies the hexadecimal offset for the command that was executing when the abend occurred. The offset is taken from the dialog's fixed dialog block (FDB).

►► For more information on the FDB, see Appendix B, "CA-ADS Dialog and Application Reporter."

PROCESS: Specifies the name of the premap or response process containing the command that caused the abend.

VERSION: Specifies the version number of the process containing the command that caused the abend.

IDD SEQ NUMBER: Specifies the data dictionary sequence number of the source line containing the command that caused the abend. The IDD sequence number is not displayed if the dialog was compiled without diagnostic tables.

►► Diagnostic tables are described in Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

SEQUENCE NUMBER: Specifies the internal command numbers of the source line containing the command that caused the abend and of the source lines immediately preceding and following it.

Internal command numbers are not displayed if the dialog was compiled without diagnostic tables.

►► Diagnostic tables are described in Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

Internal commands for CA-ADS process statements: Internal command numbers are assigned to all CA-ADS process statements in addition to the IDD sequence numbers. IDD numbers may overlap or repeat when code is included from another data dictionary module.

Internal command numbers are assigned sequentially, regardless of the source of the process code. When the abending process command is from an included module, IDD sequence numbers should be used in conjunction with internal command numbers to pinpoint the position of the command.

Internal commands for SQL statements: Internal commands are created by CA-ADS to implement SQL statements. These commands always have the sequence number of the line on which END-EXEC was coded.

SOURCE: Displays the first 70 characters of text of the source line containing the command that caused the abend and of the source lines immediately preceding and following the command. Source lines are not displayed if the dialog was compiled without diagnostic tables.

►► Diagnostic tables are described in Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

The three command lines are displayed only if the date on which the dialog was compiled agrees with the date on which the process was last revised. This prevents the display of source code that has been revised since the dialog was last compiled. Note, however, that the display of process text other than that from which the dialog was compiled could occur under the following circumstances:

- During a single day, the following actions occur:
 1. The process is revised.
 2. The dialog is recompiled.
 3. The process is revised again, but the dialog is not recompiled again.

In this case, the process source does not match the compiled process in the dialog load module.

- During a single run of the DC/UCF system, the following actions occur:
 1. The dialog is recompiled by using the batch dialog compiler.
 2. The dialog's program definition element (PDE) in the system program pool is not updated. (The PDE can be updated by using the NEW COPY option of the DCMT VARY PROGRAM command.)
 3. The dialog is executed.

In this case, the process source matches the compiled process in the dialog load module, but an old version of the dialog that remains in the DC/UCF program pool is being executed.

SYSTEM MESSAGE AREA: Displays the informational and error messages returned by the CA-ADS runtime system.

HIT ENTER TO RETURN TO DC OR ENTER NEXT TASK CODE: Prompts the user for a DC/UCF task code.

If a dialog aborts during an online debugging session, a special version of the diagnostic information screen is displayed.

4.7 Debugging a dialog

To debug a dialog, you can use the CA-ADS trace facility or the CA-IDMS online debugger. Before using either facility, you must compile the dialog with a symbol table.

►► For instructions about using the trace facility and the online debugger, see Appendix H, “Debugging a CA-ADS Dialog.”

4.8 Linking From CA-ADS to CA-OLQ

A user-written CA-ADS application compiled using the application compiler can link to CA-OLQ, pass syntax, and return to the application at the point where it was left. Linking to CA-OLQ and passing syntax is discussed below.

4.8.1 Linking to CA-OLQ

To link to CA-OLQ, perform the following steps:

1. Initialize the UNIVERSAL-COMMUNICATIONS-ELEMENT (UCE) version 2 record.
2. Issue an EXECUTE NEXT FUNCTION control command to initiate an ADSA program function that links to program IDMSOLQS, passing the UCE in the program parameter list.

Example

```
INITIALIZE (UNIVERSAL-COMMUNICATIONS-ELEMENT) .
EXECUTE NEXT FUNCTION.
RETURN.
```

4.8.2 Passing syntax to CA-OLQ

To pass syntax to CA-OLQ, perform the following steps:

1. Initialize the UNIVERSAL-COMMUNICATIONS-ELEMENT version 2 record.
2. Initialize an additional record to hold the CA-OLQ syntax. The record must contain only syntax and must not contain counters or any other values.
3. Move the syntax to the additional record.
4. Issue an EXECUTE NEXT FUNCTION control command to initiate an ADSA program function that links to program IDMSOLQS, passing both the UCE and the syntax record in the program parameter list.

Example

```
INITIALIZE (UNIVERSAL-COMMUNICATIONS-ELEMENT, SYNTAX-RECORD) .
MOVE 'SIGNON SS=EMPSS01 ! SEL * FROM EMPLOYEE ! MEN DISPLAY;'
      TO SYNTAX-FIELD.
EXECUTE NEXT FUNCTION.
RETURN.
```

Note: The exclamation point (!) is the CA-OLQ separator for stacked commands. The semi-colon (;) is the required CA-OLQ command terminator.

The command separator and terminator may differ from site to site, depending on the character set during system generation.

4.9 Linking built-in functions with the runtime system

RHDCEVnn built-in functions can be linked with the CA-ADS or CA-ADS/Batch runtime system. Linking built-in functions with the runtime system reduces the number of times built-in functions are loaded and deleted at runtime.

4.9.1 Linking system-supplied built-in functions

To link system-supplied built-in functions with the runtime system, link edit the function's module with the runtime system module at installation time. The runtime system modules are:

- Module ADSOMAIN for CA-ADS
- Module ADSOBAT2 for CA-ADS/Batch

System-supplied built-in functions are defined in one of four modules. The modules and the built-in functions contained in each module are listed below. To add the component functions to the runtime system, link edit any or all of these modules to the runtime system.

Built-in function modules

Model XDE Module	Logic Module	Built-in functions
JSSEV51	JSSEV01	<ul style="list-style-type: none"> ▪ FIX20 ▪ FIX40 ▪ FIX60 ▪ FIX80 ▪ INITCAP ▪ TOWER ▪ TOUPPER ▪ WORDCAP
JSSEV59	JSSEV09	<ul style="list-style-type: none"> ▪ TODAY ▪ TOMORROW ▪ YESTERDAY

Model XDE Module	Logic Module	Built-in functions
RHDCEV51	RHDCEV01	<ul style="list-style-type: none">▪ CONCATENATE▪ EXTRACT▪ GOODTRAILING▪ INDEX▪ INSERT▪ LEFT-JUSTIFY▪ LIKE▪ REPLACE▪ RIGHT-JUSTIFY▪ STRING-INDEX▪ STRING-LENGTH▪ STRING-REPEAT▪ SUBSTRING▪ TRAILING-TO-ZONED▪ TRANSLATE▪ VERIFY▪ ZONED-TO-TRAILING
RHDCEV52	RHDCEV02	<ul style="list-style-type: none">▪ ABSOLUTE-VALUE▪ INVERT-SIGN▪ MODULO▪ NEXT-INT-EQHI▪ NEXT-INT-EQLO▪ NUMERIC▪ RANDOM-NUMBER▪ SIGN-VALUE

Model XDE Module	Logic Module	Built-in functions
RHDCEV53	RHDCEV03	<ul style="list-style-type: none">■ ARCCOSINE-DEGREES■ ARCCOSINE-RADIANS■ ARCSINE-DEGREES■ ARCSINE-RADIANS■ ARCTAN-DEGREES■ ARCTAN-RADIANS■ COSINE-DEGREES■ COSINE-RADIANS■ LOG-BASE-10■ LOG-BASE-E■ SINE-DEGREES■ SINE-RADIANS■ SQUARE-ROOT■ TANGENT-DEGREES■ TANGENT-RADIANS

Model XDE Module	Logic Module	Built-in functions
RHDCEV59	RHDCEV09	<ul style="list-style-type: none">▪ CDATE▪ CGDATE▪ CJDATE▪ CWEEKDAY▪ DATECHG▪ DATEDIF▪ DATEOFF▪ ECDATE▪ EGDATE▪ EJDATE▪ EWEEKDAY▪ GCDATE▪ GEDATE▪ GJDATE▪ GWEEKDAY▪ JCDATE▪ JEDATE▪ JGDATE▪ JWEEKDAY▪ WEEKDAY

Model XDE Module	Logic Module	Built-in functions
RHDCEV60	RHDCEV10	<ul style="list-style-type: none">▪ CDATEX▪ CGDATEX▪ CJDATEX▪ CWEEKDAYX▪ DATECHGX▪ DATEDIFX▪ DATEOFFX▪ ECDATEX▪ EGDATEX▪ EJDATEX▪ EWEEKDAYX▪ GCDATEX▪ GEDATEX▪ GJDATEX▪ GOODDATE▪ GOODDATEX▪ GWEEKDAYX▪ JCDATEX▪ JEDATEX▪ JGDATEX▪ JWEEKDAYX▪ TODAYX▪ TOMORROWX▪ WEEKDAYX▪ YESTERDAYX

4.9.2 Linking user-written built-in functions

ADSOMAIN and ADSOBAT2 can be linked with any site-specific built-in function modules by using the required module names and entry points shown below.

Module names and entry points

Module name	Entry point
USERBIF1	BIF1EP1
USERBIF2	BIF2EP1
USERBIF3	BIF3EP1
USERBIF4	BIF4EP1
USERBIF5	BIF5EP1

4.10 Managing storage

Various storage management techniques are available to system administrators at DC/UCF sites. The following pages discuss techniques that specifically affect CA-ADS storage usage.

►► For more information about storage management, see *CA-IDMS System Generation*.

4.10.1 Adjusting record compression

Record buffer blocks (RBBs) held for a dialog can be compressed during a pseudo-converse. Record compression increases storage efficiency but causes increased CPU utilization. This option is appropriate only at sites that need to maximize storage usage.

Record compression is only in effect when resources are fixed and the fast mode threshold has not been exceeded.

Record compression can be enabled by using the system generation ADSO statement.

►► For more information, refer to *CA-IDMS System Generation*.

At runtime, the current record compression setting can be changed by using the DCMT VARY ADSO command.

►► For information about this DCMT command, refer to *CA-IDMS System Tasks and Operator Commands*.

4.10.2 Calculating RBB storage

Site administrators can direct the CA-ADS runtime system to calculate the amount of storage required for record buffer blocks (RBBs) instead of using the size specified in the system generation ADSO statement. Calculated storage reduces the amount of wasted space in the storage pool but slightly increases CPU usage. This option is a good choice for storage-constrained systems.

Calculation of runtime RBB storage is enabled in the system generation ADSO statement.

4.10.3 Writing resources to scratch records

Writing resources to scratch records during a pseudo-converse removes the resources from storage pools while the resources are not in use. This strategy is appropriate when storage pool resources are tight.

The following strategies are available to site administrators:

- **Define a fast mode threshold.** The fast mode threshold is the point at which the CA-ADS runtime system writes CA-ADS record buffer blocks (RBBs) and statistics control blocks to the scratch area (DDLDCSCR) across a pseudo-converse. If the total size of the RBBs and statistics control blocks in all storage pools exceed the fast mode threshold, the system writes the RBBs and statistics control blocks to scratch. To define fast mode threshold, specify the threshold and also that resources are relocatable in the system generation ADSO statement.

Resources must be fixed in order for the fast mode threshold to have any effect. When resources are relocatable then RBBs always go to the scratch area.

- **Define a relocatable threshold for one or more storage pools.** The relocatable threshold is the point at which the DC/UCF system writes relocatable storage to the scratch area (DDLDCSCR) across a pseudo-converse.

When this option is in effect, CA-ADS storage is always written to scratch across a pseudo-converse.

Relocatable resources: The following are relocatable resources:

- CA-ADS terminal block (OTB)
- CA-ADS terminal block extension (OTB ext)
- HELP maps
- Menu stack
- Variable dialog blocks (VDBs)

Relocating storage makes more efficient use of the storage pool but increases I/O to the scratch area. You should define a threshold so that the system relocates storage only when the storage pool is heavily used.

System generation statement: Use the system generation ADSO statement to indicate whether resources are relocatable. Use the RELOCATABLE THRESHOLD parameter of the system generation SYSTEM statement to specify the relocatable threshold for storage pool zero. For secondary storage pools, use the RELOCATABLE THRESHOLD parameter of the corresponding system generation STORAGE POOL or XA STORAGE POOL statement.

►► For more information about the ADSO and SYSTEM statements, refer to *CA-IDMS System Generation*.

4.10.4 Using XA storage

Application development tools and CA-ADS applications can be executed in XA storage on any operating system that supports XA functionality. Record buffer blocks (RBBs) and variable dialog blocks (VDBs) can be acquired from XA storage pools. The invoking task for the application determines whether the runtime system can allocate RBBs and VDBs for the entire application from XA storage pools.

Considerations: If XA storage pools are used, the following rules apply:

- If an application links to a 24-bit mode user program, the invoking task must have a location of **BELOW** to insure that storage for the program is allocated from 24-bit storage pools. For example:

```
TASK APPL1 INVOKES ADSORUN1 LOCATION BELOW.
```

- If an application links to 31-bit mode programs exclusively, the invoking task must have a location of **ANY** to take advantage of XA storage. For example:

```
TASK APPL2 INVOKES ADSORUN1 LOCATION ANY.
```

- Tasks invoked after the initial invoking task or after the return from a user program must be defined with a location of **ANY**. For example:

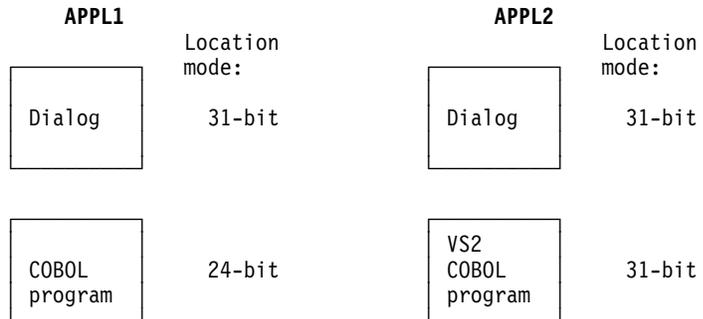
A task invoked after the initial invoking task:

```
TASK ADS2 INVOKES ADSOMAIN LOCATION ANY.
```

A task invoked after the return from a user program:

```
TASK ADS2R INVOKES ADSOMAIN LOCATION ANY.
```

Sample task definitions: The diagram below shows the task definitions for two sample applications.



Task definitions for these applications are:

1. TASK APPL1 INVOKES ADSORUN1 LOCATION BELOW.
2. TASK APPL2 INVOKES ADSORUN1 LOCATION ANY.

Chapter 5. Introduction to Process Language

- 5.1 Overview 5-3
- 5.2 Process modules 5-5
 - 5.2.1 Creating process modules 5-5
 - 5.2.2 Adding process modules to dialogs 5-5
 - 5.2.3 Executing process modules 5-5
- 5.3 Process commands 5-7
 - 5.3.1 Constructing commands 5-7
 - 5.3.2 Coding considerations 5-8
- 5.4 Data types 5-10
 - 5.4.1 Conversion between data types 5-16

5.1 Overview

There are two types of modules that can be associated with a dialog using the CA-ADS dialog compiler:

- Declaration module
- Process module

Declaration module: A **declaration module** is used under the SQL Option to declare cursors and to issue global WHENEVER statements. The statements in a declaration module are **not** executed. They are compiler directives used by the CA-ADS dialog compiler at dialog compilation.

Declaration modules allow you to store declarations you have specified as global to your application.

Unlike the premap and response process modules, the declaration module cannot contain executable CA-ADS commands. This module can contain only DECLARE CURSOR statements and WHENEVER directives.

A WHENEVER directive or DECLARE CURSOR statement is also valid in a premap or response process, but the scope of such a statement is not global.

►► Further explanation of usage for WHENEVER and DECLARE CURSOR can be found in the *CA-IDMS SQL Reference*.

For further considerations regarding the declaration module, see the *CA-IDMS SQL Programming*.

Since declaration modules do not contain executable code, they are not discussed in this chapter.

Process modules: In CA-ADS, **process modules** are defined to handle dialog-specific processing, such as data retrieval, data modifications, and data storage. Each process module consists of one or more **process commands** and **parameters** that qualify the commands.

Data referenced by CA-ADS process commands must be predefined in the data dictionary.

►► Instructions for defining data can be found in *IDD DDDL Reference*.

Instructions for defining data in subschemas can be found in *CA-IDMS Database Administration*.

Data types supported by CA-ADS are presented in 5.4, “Data types” later in this chapter.

5.2 Process modules

A process module is a discrete dialog unit that performs the processing operations required by a given dialog.

5.2.1 Creating process modules

CA-ADS process modules are created and stored in the data dictionary by using IDD. The IDD menu facility provides a series of menus used to define process modules to the data dictionary.

►► For instructions on using the IDD menu facility in the CA-ADS environment, refer to *CA-ADS User Guide*.

The online IDD PROCESS statement can also be used.

►► For IDD PROCESS statement syntax, refer to *IDD DDDL Reference*.

5.2.2 Adding process modules to dialogs

A process module is added to a dialog as either a premap, a response process or a declaration module. The process module is associated with a dialog by using the CA-ADS dialog compiler (ADSC) or the batch dialog compiler (ADSOBCOM). The module is compiled when the dialog is compiled.

►► Instructions for using ADSC to associate process modules with dialogs are provided in *CA-ADS User Guide*.

For information about ADSOBCOM, see Appendix D, "Application and Dialog Utilities."

5.2.3 Executing process modules

Process modules are executed before or after a dialog's map is displayed on the terminal screen (online applications) or are used to transfer input or output data (CA-ADS/Batch):

- A **premap process module**, which is executed before a map is used to transfer data between variable storage and an online terminal (CA-ADS) or files (CA-ADS/Batch).

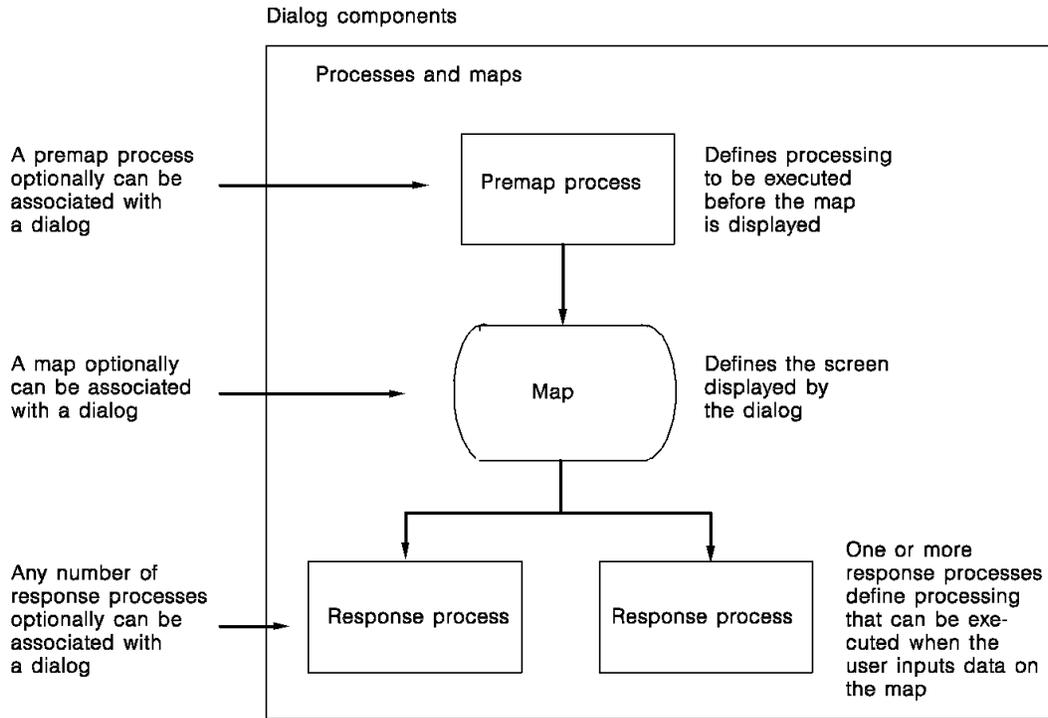
A dialog can have a maximum of one premap process.

- A **response process module**, which is performed after a map is displayed, based on the user's selection of a response.

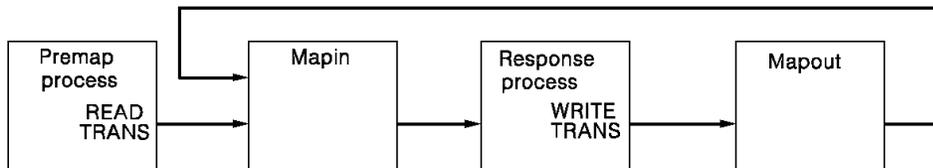
A dialog can have any number of response processes.

CA-ADS premap and response processes: The diagram below shows the execution sequence of CA-ADS premap and response process modules.

A premap process is executed before the dialog's map is displayed to the end user. A response process, based on the user's selection, is executed.



CA-ADS/Batch premap and response processes: The diagram below shows the execution sequence of CA-ADS/Batch premap and response process modules. A premap process is executed at the beginning of the dialog unless the dialog's entry point is its mapin operation. The process executes until it issues a READ/WRITE TRANSACTION command. The response process executes after the mapin operation and continues until it issues a READ/WRITE TRANSACTION command.



5.3 Process commands

CA-ADS premap and response process modules are written using process commands. Process commands are COBOL-like statements.

The way that process commands are constructed and general coding considerations for CA-ADS process commands are discussed below.

►► For coding a declaration module, see *CA-IDMS SQL Programming*.

5.3.1 Constructing commands

Command statements consist of commands and qualifying parameters.

Verbs Specify the operation to perform.: For example, RETURN, COMPUTE, IF, DISPLAY, OBTAIN, and WRITE PRINTER are commands.

Parameters: Qualify commands and specify additional operations to perform. Parameters can be:

- **Keywords**, which are system-defined values. Each keyword must be specified as shown in the documentation. (The required portion of each keyword is shown in capital letters.)

For example, in the RETURN CLEAR statement, CLEAR is a keyword that qualifies the operation of RETURN.

- **Variable terms**, which show where user-defined values can be coded in process command syntax.

For example, in the following command statement, *dialog-name* is a variable term:

RETURN TO dialog-name

Syntax and examples of variable terms are presented in the remaining chapters of this volume. The following table summarizes the types of variable terms that can be used.

Type of variable	Purpose
Arithmetic expression	Specifies a simple or compound arithmetic operation
Built-in function	Specifies evaluation of a value according to a predefined operation
Conditional expression	Specifies test conditions
Constant	Specifies a value to be used in command processing
Error expression	Permits the return of error status codes to a dialog
Variable data field	Supplies the name of a user- or system-supplied data field for use in command processing

- A combination of keywords and/or variable terms.

5.3.2 Coding considerations

Process commands are coded by using syntax specific to each command. The following general coding considerations apply to command statements:

- A command statement can be coded in any column and continue through column 72.
- A statement can be coded on one or more lines. The following considerations apply:
 - No continuation character is required.
 - More than one command can be coded on a single line, with the exception of the INCLUDE command.
 - Extend strings to the next line by coding up to and including column 72 of one line, and continuing in column 1 of the next line.
- The command must always appear first, followed by command parameters, if any.
- Parameters must be separated from each other and from the command by one or more blanks or commas.
- Each statement must be terminated with a period.
- Blank lines can be used to improve readability.

-
- Commas and blanks can be inserted anywhere between command parameters to improve readability, but cannot be used as a null place holder in a list in a command, such as:

wrong ► LINK TO PROGRAM XYZ USING (REC1,,REC2)

wrong ► INITIALIZE RECORDS (REC1,,REC2).

Commas and blanks cannot be used in a built-in function.

- Comments in CA-ADS statements are specified by using an exclamation point followed by the comment text. The following considerations apply:
 - All characters between the exclamation point and the end of the line are considered part of the comment.
 - A comment can be terminated before the end of the line by using a second exclamation point. All characters following a second exclamation point are considered to be part of a command.
- Comments within SQL statements are specified by using two hyphens (--) at the beginning of the comment.

All characters between the hyphens and the end of the line are considered part of the comment.

```
EXEC SQL.  
SELECT * FROM PROD.EMPLOYEE WHERE EMP_ID > 5555;  
--Selecting employees having consultant ids  
END-EXEC.
```

- A statement can include a quoted string of up to 255 characters.
- Quotation marks appearing within a quoted string must be coded as two consecutive single quotation marks.

5.4 Data types

A data type is the internal representation of data. Data referenced by CA-ADS process statements must be predefined in the data dictionary using IDD alone or IDD and the DDL compiler. The data types supported by CA-ADS are described below. Examples of each data type are outlined later in this chapter.

▶▶ See *CA-IDMS SQL Reference* for a discussion of the correlation between CA-ADS data types and SQL data types.

▶▶ See the *CA-IDMS SQL Programming* manual for a discussion of the correlation between CA-ADS data types and SQL data types.

Binary: Binary data fields are 1- to 18-digit signed integer data fields. The left-most bit in a binary field is zero for a positive integer and one for a negative integer. The remainder of the binary field contains the numeric value. A negative value is stored in twos complement form.

The following table describes the characteristics of binary data fields and shows how each type of binary field is defined in the data dictionary.

Binary field	Size	Range	Data dictionary definition
Halfword	2 bytes	-2^{15} to $2^{15}-1$	PICTURE S9(n) USAGE IS COMPUTATIONAL (n is an integer ranging from 1 to 4)
Fullword	4 bytes	-2^{31} to $2^{31}-1$	PICTURE S9(n) USAGE IS COMPUTATIONAL (n is an integer ranging from 5 to 9)
Doubleword	8 bytes	-2^{63} to $2^{63}-1$	PICTURE S9(n) USAGE IS COMPUTATIONAL (n is an integer ranging from 10 to 18)

EBCDIC: EBCDIC data fields are data fields containing any value in the EBCDIC collating sequence (hexadecimal '00' through 'FF').

The following table describes the characteristics of EBCDIC data field and shows how it is defined in the data dictionary.

Size	Maximum length	Data dictionary definition
1 byte per character	32,767 bytes	PICTURE X USAGE IS DISPLAY

Floating point: A floating point data field is a numeric data field whose value is expressed as a mantissa, which represents the number, and an exponent (characteristic), which determines the actual decimal position of the number. The value of a floating point data field is the product of the mantissa, and ten raised to the power of the characteristic. A 1- to 16-digit mantissa can be used.

The following table describes the characteristics of floating point data fields and shows how each field is defined in the data dictionary.

Data Field	Size	Exponent range	Data dictionary definition
Internal short ¹	4 bytes ²	-64 to +63	USAGE IS COMPUTATIONAL-1 (No picture clause)
Internal long ¹	8 bytes ²	-64 to +63	USAGE IS COMPUTATIONAL-2 (No picture clause)
Display ³	1 byte for each character	-64 to +63	PICTURE ±9V99E±99 USAGE IS DISPLAY

Notes:

¹ To display data field values on a map, assign them to a floating point display data field, or, if small enough, to a decimal or binary field.

² The left-most byte contains the sign of the mantissa and the characteristic. The last 3 or 7 bytes contain the binary representation of the mantissa. Either 7 or 17 decimal digits are allowed.

³ Display floating point data fields are in a displayable format. When used in calculations, display floating point fields are converted to equivalent internal floating point values.

Group: Group data fields, including record names, contain subordinate data fields. A group data field references the storage of all subordinate data fields without consideration of their data types. A group data field has no PICTURE or USAGE clauses.

Multibit binary: Multibit data fields are binary data fields. At runtime, the data fields contain either a 0 or 1 for each character.

The following table describes the characteristics of a multibit binary field and shows how the field is defined in the data dictionary.

Data field	Size	Exponent range	Data dictionary definition
Multibit binary	1 bit per character	1- to 32- characters	PICTURE X USAGE IS BIT

Packed decimal: Packed decimal numeric data fields occupy a half byte of storage per digit. The sign of the number (hexadecimal C, for positive, D for negative, and F for no sign) is stored in the four low-order bits of the rightmost byte. S is used only when the field is signed.

If a packed decimal field is defined with an even number of digits, the field is considered to have one extra digit to the left of the decimal point. For example, a packed decimal field with a picture of 9(4)V99 is considered to have a picture of 9(5)V99.

A packed decimal field with a picture of 9(4)V99 or S9(4)V99 occupies a half byte for the sign and a half byte for each digit, totaling 3.5 bytes. Four bytes are reserved for this field, adding an extra digit to the left of the decimal point.

The following table describes the characteristics of a packed decimal data field and shows how the field is defined in the data dictionary.

Data field	Size	Range	Data dictionary definition
Packed decimal	1/2 byte per digit	1 to 18 digits	PICTURE S9V99 USAGE IS COMPUTATIONAL-3

Zoned decimal: Zoned decimal numeric data fields occupy one byte of storage per digit. The sign of the number (hexadecimal C for positive, D for negative, F for unsigned positive) is stored in the four high-order bits of the rightmost digit. S is used only when the field is signed.

The following table describes the characteristics of a zoned decimal data field and how the field is defined in the data dictionary.

Data field	Size	Range	Data dictionary definition
Zoned decimal	1 byte per digit	1 to 18 digits	PICTURE S9V99 USAGE IS DISPLAY

Examples of data types: The following table illustrates the definition, use, and internal representation of the different data types.

Type	Description ¹	Command
Group	EMPLOYEE ← Group item EMP-ID 9(4). EMP-NAME PIC X(10).	MOVE 'abcde' TO EMPLOYEE.
	Internal representation:²	
EBCDIC	FIELD-1 PIC X(10).	MOVE 'abcde' TO EMPLOYEE.
	Internal representation:	
Zoned decimal	AMT-1 PIC 9(4).	MOVE 4505 TO AMT-1.
	Internal representation:	
	AMT-1 PIC S9(4).	MOVE 4505 TO AMT-1.
	Internal representation:	
Packed decimal	AMT-1 PIC 9(4) USAGE COMP-3.	MOVE 4505 TO AMT-1.
	Internal representation:	
	AMT-1 PIC S9(4) USAGE COMP-3.	MOVE 4505 TO AMT-1.
	Internal representation:	

Type	Description ¹	Command										
Binary	AMT-1 PIC S9(8) USAGE COMP.	MOVE 4505 TO AMT-1.										
	Internal representation:	<table border="1"> <tr> <td>0000</td><td>0000</td><td>0000</td><td>0000</td><td>0001</td><td>0001</td><td>1001</td><td>1001</td> </tr> </table> byte	0000	0000	0000	0000	0001	0001	1001	1001		
0000	0000	0000	0000	0001	0001	1001	1001					
	AMT-1 PIC S9(8) USAGE COMP.	MOVE -4505 TO AMT-1.										
	Internal representation:	<table border="1"> <tr> <td>1111</td><td>1111</td><td>1111</td><td>1111</td><td>1110</td><td>1110</td><td>0110</td><td>0110</td> </tr> </table> byte	1111	1111	1111	1111	1110	1110	0110	0110		
1111	1111	1111	1111	1110	1110	0110	0110					
Multibit binary	FIELD-1 PIC X(10) USAGE BIT.	MOVE B'10011100' TO FIELD-1.										
	Internal representation:	<table border="1"> <tr> <td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td> </tr> </table> bit	1	0	0	1	1	1	0	0	0	0
1	0	0	1	1	1	0	0	0	0			
Floating point	AMT-1 USAGE COMP-1. (Internal short)	MOVE -45.05E02 TO AMT-1. (Value will be stored as -4505E04)										
	Internal representation:	<table border="1"> <tr> <td>C4</td><td>11</td><td>99</td><td>00</td> </tr> </table> byte	C4	11	99	00						
C4	11	99	00									
	AMT-1 PIC S9.9999 E-99.	MOVE -45.05 E 02 TO AMT-1.										
	Internal representation:	<table border="1"> <tr> <td>-</td><td>4</td><td>.</td><td>5</td><td>0</td><td>5</td><td>0</td><td>E</td><td>0</td><td>3</td> </tr> </table> byte	-	4	.	5	0	5	0	E	0	3
-	4	.	5	0	5	0	E	0	3			

Notes:

¹ If no USAGE clause is provided, the default usage is DISPLAY.

² A blank space = a blank (X'40') in internal representations.

5.4.1 Conversion between data types

CA-ADS automatically performs data type conversion in the following cases:

- In an **assignment command**, conversion is performed if the target field is a different data type than the source field.
- In an **arithmetic command**, conversion is performed if the target field is a different data type than the result of the command.
- In an **arithmetic expression**, all operands are converted to signed packed decimal fields or, if required, to internal floating point fields before the arithmetic operation is performed.
- In any command in which **numeric literals** are used, fixed point numeric literals are stored internally as packed decimal fields, and floating point numeric literals are stored internally as internal short or long floating point fields.

Data type conversions: The following table shows the permissible data type conversions in arithmetic and assignment commands and in arithmetic expressions. Source data types are presented down the left-hand side. Target data types are presented across the top. Permissible conversions are indicated by a YES in the box formed by the intersection of the applicable source and target data types.

SOURCE	TARGET						
	Group	EBCDIC	Binary*	Decimal**	Multi-bit binary	Internal float pt.	Display float pt.
Group	YES ¹	YES ¹	YES ²	YES ²	YES ³	YES ²	NO
EBCDIC	YES ¹	YES ¹	YES ²	YES ²	YES ³	YES ²	NO
Binary*	YES ¹	YES ⁴	YES	YES	YES ⁵	YES	NO
Decimal**	YES ¹	YES ⁴	YES	YES	YES ⁵	YES	YES
Multibit binary	YES ⁶	YES ⁶	YES ⁷	YES ⁷	YES ¹⁰	YES ⁷	NO
Internal float pt.***	YES ¹	YES ⁹	YES	YES	YES ⁵	YES	YES ⁸
Display float pt.***	YES ¹	YES ¹	YES	YES	NO	YES	YES

Notes:

* Binary includes halfword, fullword, and doubleword binary.

** Decimal includes zoned and packed decimal.

*** Internal floating point includes internal short and long floating point.

¹ Source moved to target without conversion. Target is blank-filled or truncated on right, if necessary.

² Number begins at leftmost numeric digit and includes all numeric digits up to the first nonnumeric character (or end of data field). A negative sign can immediately precede the number. A decimal point can immediately precede or be embedded in the number. Embedded commas are ignored.

³ Bits in source moved to bits in target without conversion. Target is binary zero filled or truncated on right, if necessary.

⁴ If CA-ADS moves are in effect, decimal portion and leading zeros are dropped, a negative sign, if any, is placed in front of the number, and the result is left justified in the target field, with leading blanks.

⁵ If COBOL moves are in effect, the decimal portion (without the decimal point) and leading zeros are maintained, the negative sign if any, is dropped, and the result is left justified in the target, with blank filling or truncation on right, if necessary.

►► For information on requesting CA-ADS or COBOL moves, see 3.3.2, “Options and Directives screen.”

⁶ Decimal component of the number is dropped, forced positive, and converted to a binary fullword. Bits are moved left to right. Target is binary, zero filled, or truncated on right, if necessary.

⁷ Each bit value 0 or 1 is converted to the character 0 or 1, as appropriate. Target is blank filled or truncated on right, if necessary.

⁸ Source bits are right justified in a fullword. The resulting fullword value is forced positive by moving 0 to the leftmost bit, and is moved to the target with any required data conversion.

⁹ The maximum output length is 23 bytes (mantissa sign, 17-digit mantissa, decimal point, character E, characteristic sign, and 2-digit characteristic). The minimum output length is 6 bytes (mantissa sign, 1-digit mantissa, character E, characteristic sign, and 2-digit characteristic).

¹⁰ The mantissa is converted to zoned decimal format and moved to the target. The negative sign and decimal point, if any, are dropped. The characteristic is not moved. Target is blank filled or truncated on right, if necessary.

¹¹ Target is binary zero filled or truncated on right, if necessary.

Chapter 6. Arithmetic Expressions

- 6.1 Overview 6-3
- 6.2 Syntax 6-4
- 6.3 Evaluation of arithmetic expressions 6-5
- 6.4 Coding considerations 6-6

6.1 Overview

An arithmetic expression is a variable term that can be a simple or compound arithmetic operation. An arithmetic expression can be used as a variable wherever the command syntax specifies arithmetic-expression

The elements allowed in an arithmetic expression are summarized in the table below. Arithmetic expressions are composed of operands, binary operations, and unary operations. The elements of each entity are listed below.

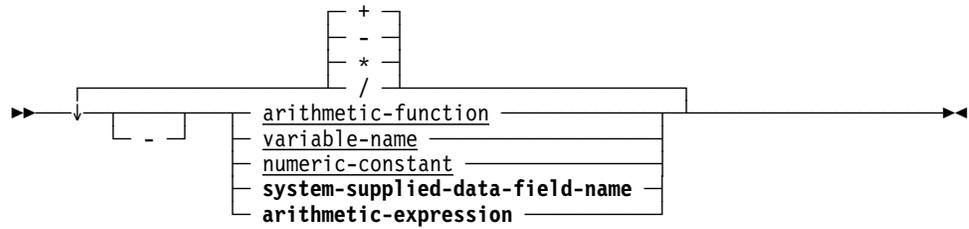
Arithmetic expression elements

Operands	Binary operators	Unary operators
Variable data fields	Addition [+]	Plus [+]
Numeric constants	Subtraction [-]	Minus [-]
Built-in functions	Multiplication [*]	
	Division [/]	

Considerations

- Any number of parentheses can be included in the expression to indicate order of evaluation.
- Parentheses can be nested.

6.2 Syntax



Parameters

-

The unary minus operator. It reverses the sign of the operand that follows it.

arithmetic-function

►► For a list of arithmetic built-in functions, see Chapter 7, “Built-in Functions.”

variable

A user-defined variable data field.

The named variable must contain a number and can be any of the following:

- A field on a map
- A numeric variable
- An element in a group
- An element in an array

numeric-constant

A number.

system-supplied-data-field-name

See “System-supplied data field names” in Chapter 11, “Variable Data Fields”

arithmetic-expression

An arithmetic expression. Use parentheses to control the order in which operations are to be performed.

+ - * /

The arithmetic operators:

Operator	What it does
+	Addition
-	Subtraction
*	Multiplication
/	Division

6.3 Evaluation of arithmetic expressions

Arithmetic expressions are evaluated according to the following rules:

- Expressions within parentheses are evaluated first. Within nested parentheses, evaluation proceeds from the innermost to the outermost set of parentheses.
- If the order of evaluation of an expression or of an embedded expression is not specified explicitly by parentheses, the following order of evaluation is implied:
 1. Unary plus, unary minus, and built-in functions, from left to right
 2. Multiplication and division, from left to right
 3. Addition and subtraction, from left to right

Variable data fields specified in an arithmetic expression are not changed during the evaluation of the expression. All intermediate results in an expression are stored in separate internal data fields.

Example: The following example illustrates the order of evaluation of arithmetic expressions in process commands:

```
MOVE -(4 - VALUE1 / (ABS(VALUE2) + -5 / VALUE3) + VALUE4)
      TO RESULT.
```

The expression is evaluated in the following order:

1. The absolute value of VALUE2 is calculated.
2. Unary minus is applied to 5.
3. The result of step 2 is divided by VALUE3.
4. The result of step 3 is added to the result of step 1.
5. VALUE1 is divided by the result of step 4.
6. The result of step 5 is subtracted from 4.
7. The result of step 6 is added to VALUE4.
8. Unary minus is applied to the result of step 7.

The result of the expression is moved to RESULT.

6.4 Coding considerations

The following considerations apply to coding arithmetic expressions:

- An arithmetic expression must begin with a left parenthesis, a unary operator, or an operand.
- An arithmetic expression must end with a right parenthesis or an operand.
- An arithmetic expression does not require a binary operation.
- Each left parenthesis must be followed later in the expression by a corresponding right parenthesis.
- Operands and binary operators must be separated by at least one space from the operand or operator that follows. Parentheses do not require surrounding spaces.
- Operands can be followed by a right parenthesis, any binary operator, or can be the end of the expression.
- Any binary operator can be followed by an operand, a unary operator, or a left parenthesis.
- A unary operator can be followed by an operand or a left parenthesis.
- A left parenthesis can be followed by an operand, a unary operator, or another left parenthesis.
- A right parenthesis can be followed by any binary operator, another right parenthesis, or can be the end of the expression.

Chapter 7. Built-in Functions

7.1 Overview	7-3
7.1.1 Invocation names	7-3
7.1.2 Built-in function values	7-4
7.1.3 Coding parameters	7-4
7.2 User-defined built-in functions	7-5
7.3 System-supplied functions	7-6
7.3.1 Arithmetic functions	7-6
7.3.2 Date functions	7-6
7.3.3 String functions	7-7
7.3.4 Trailing-sign functions	7-8
7.3.5 Trigonometric functions	7-9
7.4 ABSOLUTE-VALUE	7-11
7.5 ARC COSINE	7-12
7.6 ARC SINE	7-13
7.7 ARC TANGENT	7-14
7.8 CONCATENATE	7-15
7.9 COSINE	7-16
7.10 DATECHG	7-17
7.11 DATEDIF	7-20
7.12 DATEOFF	7-21
7.13 EXTRACT	7-23
7.14 FIX	7-24
7.15 GOODDATE	7-25
7.16 GOODTRAILING	7-26
7.17 INITCAP	7-27
7.18 INSERT	7-28
7.19 INVERT-SIGN	7-30
7.20 LEFT-JUSTIFY	7-31
7.21 LIKE	7-32
7.22 LOGARITHM	7-34
7.23 MODULO	7-35
7.24 NEXT-INT-EQHI	7-36
7.25 NEXT-INT-EQLO	7-37
7.26 NUMERIC	7-38
7.27 RANDOM-NUMBER	7-40
7.28 REPLACE	7-42
7.29 RIGHT-JUSTIFY	7-44
7.30 SIGN-VALUE	7-45
7.31 SINE	7-46
7.32 SQUARE-ROOT	7-47
7.33 STRING-INDEX	7-48
7.34 STRING-LENGTH	7-49
7.35 STRING-REPEAT	7-50
7.36 SUBSTRING	7-51
7.37 TANGENT	7-53
7.38 TODAY	7-54
7.39 TOLOWER	7-55

7.40	TOMORROW	7-56
7.41	TOUPPER	7-57
7.42	TRAILING-TO-ZONED	7-58
7.43	TRANSLATE	7-59
7.44	VERIFY	7-61
7.45	WEEKDAY	7-62
7.46	WORDCAP	7-65
7.47	YESTERDAY	7-66
7.48	ZONED-TO-TRAILING	7-67

7.1 Overview

Built-in functions evaluate expressions according to predefined operations and return results that can be used in command processing. Built-in functions use a specified list of parameters, which are not changed by the execution of the function.

A built-in function can be used wherever the syntax for a variable expression specifies an arithmetic expression, the name of a user-defined data field, a user supplied numeric constant, a literal in quotes, or a string-variable.

Built-in functions supported: CA-ADS supports the following types of built-in functions:

- **System-supplied functions** that perform predefined arithmetic, date, string, and trigonometric operations. The built-in function names given in this manual are default invocation names that can be changed.
 - ▶▶ For more information, see F.4, “Changing invocation names.”

- **User-defined functions** that perform site-specific functions defined by the installation.
 - ▶▶ For a description of how to create a user-defined function, see F.5, “Creating user-defined built-in functions.”

User-defined and system-defined functions are described below, after a discussion of general considerations that apply to both types of built-in functions.

7.1.1 Invocation names

A built-in function is invoked by means of a unique invocation name, such as CONCATENATE, CONCAT, or CON for the concatenate function.

Note: Built-in function names are keywords. If an invocation name is the same name as a data field known to a dialog, an error occurs because CA-ADS interprets the function invocation name as a subscripted reference to the data field.

An invocation name can be changed by modifying the internal table of invocation names (the master function table).

▶▶ For more information, see F.4, “Changing invocation names.”

7.1.2 Built-in function values

Values are supplied to a built-in function according to **parameters** that are coded along with the function's invocation name. Parameters can be either string values or numeric values, as follows:

- A **string value** should be coded as an EBCDIC variable data field, a nonnumeric literal, or a built-in function that returns a string value. A value in a string built-in function cannot be zero in length and cannot be filled with only spaces.
- A **numeric value** should be coded as an arithmetic expression, a numeric variable data field, a numeric literal, or a built-in function that returns a numeric value.

Some built-in function parameters have restrictions on the values they can contain. If an invalid value is specified at runtime, the dialog aborts. For example, the value specified in a square root function must be positive.

If a parameter is specified with a different data type than expected, CA-ADS attempts to make the appropriate conversion at runtime. The dialog aborts if the conversion cannot be made.

►► For permissible conversions, see the datatype conversion table under "Conversion between data types" in Chapter 5, "Introduction to Process Language."

7.1.3 Coding parameters

Parameters are coded within parentheses and separated by commas.

Each parameter must be coded in a specific position relative to the other parameters. When an optional parameter is not included in a parameter list, it must be replaced by the @ character unless no further parameters follow the optional parameter.

7.2 User-defined built-in functions

User-defined built-in functions perform functions that are defined by individual sites.

▶▶ For a description of how to create user-defined functions, see Appendix F, “Built-in Function Support.”

7.3 System-supplied functions

CA-ADS system-supplied functions perform predefined arithmetic, date, string, and trigonometric functions. System-supplied functions are summarized in the tables that follow. Detailed discussions for each particular function appear later in this chapter, arranged alphabetically by function name.

7.3.1 Arithmetic functions

Arithmetic built-in functions (with the exception of NUMERIC) perform arithmetic operations on numeric values and return numeric values as results.

Function	Keyword	What it does
Absolute value	ABSOLUTE-VALUE	Returns the absolute value of a numeric value
Logarithm (base 10)	LOG-BASE-10	Returns the common logarithm of a numeric value
Logarithm (base E)	LOG-BASE-E	Returns the natural logarithm of a numeric value
Modulo	MODULO	Returns the modulus (remainder) of one specified numeric value divided by another
Next integer equal or higher	NEXT-INT-EQHI	Returns the smallest integer that is equal to or greater than a specified numeric value
Next integer equal or lower	NEXT-INT-EQLO	Returns the largest integer that is equal to or lower than a specified numeric value
Numeric	NUMERIC	Returns TRUE or FALSE to indicate whether a field is numeric
Random number	RANDOM-NUMBER	Returns a pseudo-random number based on a seed numeric value
Sign inversion	INVERT-SIGN	Returns the value of a numeric value multiplied by -1
Sign value	SIGN-VALUE	Returns a +1, 0, or -1, depending on whether a numeric value is positive, zero, or negative
Square root	SQUARE-ROOT	Returns the square root of a numeric value

7.3.2 Date functions

Date built-in functions perform date processing in eight formats:

- **Gregorian**— The first format is *yymmdd*, where *yy* represents a year, *mm* a month, and *dd* a day.

The second Gregorian format is *yyyymmdd*, where *yyyy* represents a year in any century, *mm* a month, and *dd* a day.

- **Calendar**— The first format is *mmddy*, where *yy* represents a year, *mm* a month, and *dd* a day.

The second Calendar format is *mmddyyyy*, where *yyyy* represents a year in *any century*, *mm* a month, and *dd* a day.

- **European**— The first format is *ddmmyy*, where *yy* represents a year, *mm* a month, and *dd* a day.

The second European format is *ddmmyyyy*, where *yyyy* represents a year in *any century*, *mm* a month, and *dd* a day.

- **Julian**— The first format is *yyddd*, where *ddd* is a day in the year from 1 to 365 (366 for leap years).

The second Julian format is *yyyyddd*, where *yyyy* represents a year in *any century*, and *ddd* is a day in the year from 1 to 365 (366 for leap years).

Function	Keyword	What it does
Date change	DATECHG	Returns Gregorian, calendar, European, or Julian date conversions
Date difference	DATEDIF	Returns the number of days between two specified dates
Date offset	DATEOFF	Returns the date resulting from adding a specified number of days to a date
Good date	GOODDATE	Returns TRUE or FALSE to indicate whether a date is valid for the date type
Today's date	TODAY	Returns today's date in the specified format
Tomorrow's date	TOMORROW	Returns tomorrow's date in the specified format
Weekday	WEEKDAY	Returns the weekday of a specified date
Yesterday's date	YESTERDAY	Returns yesterday's date in the specified format

7.3.3 String functions

String built-in functions perform operations on string values and return either string or numeric values.

Function	Keyword	What it does
Concatenate	CONCATENATE	Returns the concatenation of a specified list of string values
Extract	EXTRACT	Returns the string that results from removing leading and trailing spaces from a string value
Fixed-length string	FIX	Converts a string to a fixed-length character variable

Function	Keyword	What it does
Index	STRING-INDEX	Returns the starting position of a specified string within a string value
Initial cap	INITCAP	Capitalizes the first letter of a string
Insert	INSERT	Returns the string that results from inserting a specified string into a string value starting at a specified position
Left justify	LEFT-JUSTIFY	Returns the string that results from left justifying a string value
Length	STRING-LENGTH	Returns the length of a string value
Like	LIKE	Returns TRUE or FALSE to indicate whether a source string matches a given pattern string
Lowercase	TOLOWER	Converts a string to lowercase characters
Repeat	STRING-REPEAT	Returns the string that results from repeating a string value a specified number of times
Replace	REPLACE	Returns a string that results from replacing, in a string value, each occurrence of a specified string by another specified string
Right justify	RIGHT-JUSTIFY	Returns the string that results from right justifying a string value
Substring	SUBSTRING	Returns the substring of a string value, starting from a specified position, and continuing for a specified length
Uppercase	TOUPPER	Converts a string to uppercase characters
Translate	TRANSLATE	Returns the string that results from translating characters in a string value that also occur in a selection string, to corresponding characters in a substitution string
Verify	VERIFY	Returns the position of the first character in a string value that does not occur in a second specified string
Word cap	WORDCAP	Capitalizes the first character in each word in a string

7.3.4 Trailing-sign functions

Trailing-sign built-in functions support conversion between trailing sign and zoned decimal representations.

Function	Keyword	What it does
Good trailing sign	GOODTRAILING	Returns TRUE or FALSE to indicate whether a target field is a valid trailing sign numeric field
Trailing to zoned	TRAILING-TO-ZONED	Returns a zoned numeric from a COBOL trailing sign numeric
Zoned to trailing	ZONED-TO-TRAILING	Returns a COBOL trailing sign numeric from a zoned numeric

7.3.5 Trigonometric functions

Trigonometric built-in functions perform trigonometric operations on numeric values that represent angles in either degrees or radians, and return numeric values that are the results of the operations.

Function	Keyword	What it does
Arc cosine (degrees)	ARCCOSDEG	Returns the arc cosine of a numeric value that represents an angle in degrees
Arc cosine (radians)	ARCCOSRAD	Returns the arc cosine of a numeric value that represents an angle in radians
Arc sine (degrees)	ARCSINDEG	Returns the arc sine of a numeric value that represents an angle in degrees
Arc sine (radians)	ARCSINRAD	Returns the arc sine of a numeric value that represents an angle in radians
Arc tangent (degrees)	ARCTANDEG	Returns the arc tangent of a numeric value that represents an angle in degrees
Arc tangent (radians)	ARCTANRAD	Returns the arc tangent of a numeric value that represents an angle in radians
Cosine (degrees)	COSINE-DEGREES	Returns the cosine of a numeric value that represents an angle in degrees
Cosine (radians)	COSINE-RADIANS	Returns the cosine of a numeric value that represents an angle in radians
Sine (degrees)	SINE-DEGREES	Returns the sine of a numeric value that represents an angle in degrees
Sine (radians)	SINE-RADIANS	Returns the sine of a numeric value that represents an angle in radians
Tangent (degrees)	TANGENT-DEGREES	Returns the tangent of a numeric value that represents an angle in degrees

7.3 System-supplied functions

Function	Keyword	What it does
Tangent (radians)	TANGENT-RADIANS	Returns the tangent of a numeric value that represents an angle in radians

7.4 ABSOLUTE-VALUE

Purpose: Returns the absolute value of a numeric value.

Syntax

► ABSOLUTE-VALUE (value) ◄

ABS-val

Parameters

value

Specifies the numeric value whose absolute value is calculated.

Value can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Example: In the example below, the absolute value function is used to specify the absolute value of a calculated length in a substring function:

Initial values:
 EMP-NAME: 'JOE SMITH'
 WK-LENGTH: -3

Statement:
 MOVE SUB(EMP-NAME,1,ABS(WK-LENGTH)) TO WK-FNAME.

Returned value from ABS function: 3

Returned string from SUB function: 'JOE'

7.5 ARC COSINE

Purpose: Returns the arc cosine of a numeric value that represents an angle in either degrees or radians.

Syntax: Arc cosine (degrees):

► ARCCOSINE-DEGREES (value)
 └─ ARCCOSDEG
 └─ ACOSD

Arc cosine (radians):

► ARCCOSINE-RADIANS (value)
 └─ ARCCOSRAD
 └─ ACOSR

Parameters

ARCCOSINE-DEGREES

Returns an arc cosine value in degrees.

ARCCOSINE-RADIANS

Returns an arc cosine value in radians.

value

Specifies the numeric value representing the angle, in degrees or radians, whose arc cosine is calculated.

Value can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Value must be a value ranging from -1 to +1.

Example: In the following example, the arc cosine (degrees) of -0.5 is calculated and moved to WK-RESULT (PIC S999V9999):

```
MOVE ACOSD(-0.5) TO WK-RESULT.
```

Returned value: 120.

7.7 ARC TANGENT

Purpose: Returns the arc tangent of a numeric value that represents an angle in either degrees or radians.

Syntax: Arc tangent (degrees):

► [ARCTAN-DEGREES | ARCTANDEG | ATAND] (value) →

Arc tangent (radians):

► [ARCTAN-RADIANS | ARCTANRAD | ATANR] (value) →

Parameters

ARCTAN-DEGREES

Returns an arc tangent value in degrees.

ARCTAN-RADIANS

Returns an arc tangent value in radians.

value

Specifies the numeric value representing the angle, in degrees or radians, whose arc tangent is calculated.

Value can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Example: In the following example, the arc tangent (degrees) of 1.7321 is calculated and moved to WK-RESULT (PIC S999V9999):

```
MOVE ATAND(1.7321) TO WK-RESULT.
```

Return value: 60.0007

7.9 COSINE

Purpose: Returns the cosine of a numeric value that represents an angle in either degrees or radians.

Syntax: Cosine (degrees):

► $\left[\begin{array}{l} \text{COSINE-DEGREES} \\ \text{COSDeg} \end{array} \right] (\text{value})$ —————►

Cosine (radians):

► $\left[\begin{array}{l} \text{COSINE-RADIANS} \\ \text{COSRad} \end{array} \right] (\text{value})$ —————►

Parameters

COSINE-DEGREES

Returns a cosine value in degrees.

COSINE-RADIANS

Returns a cosine value in radians.

value

Specifies the numeric value representing the angle, in degrees or radians, whose cosine is calculated.

Value can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Example: In the following example, the cosine (degrees) of 60 is calculated and moved to WK-RESULT (PIC S999V9999):

```
MOVE COSD(60) TO WK-RESULT.
```

Return value: 0.5

7.10 DATECHG

Purpose: Returns the conversion of a specified date from one format (Gregorian, calendar, European, or Julian) to another.

Date change functions can be coded two ways, as shown in the following syntax diagrams.

Syntax: Format 1:

```

  >> [ DATECHG ] ( date, input-date-format, output-date-format ) <<<
      [ DATECHGX ]
  
```

Format 2:

```

  >> [ GCDATE ] ( date ) <<<
      [ GCDATEX ]
      [ GEDATE ]
      [ GEDATEX ]
      [ GJDATE ]
      [ GJDATEX ]
      [ CGDATE ]
      [ CGDATEX ]
      [ CEDATE ]
      [ CEDATEX ]
      [ CJDATE ]
      [ CJDATEX ]
      [ EGDATE ]
      [ EGDATEX ]
      [ ECDATE ]
      [ ECDATEX ]
      [ EJDATE ]
      [ EJDATEX ]
      [ JGDATE ]
      [ JGDATEX ]
      [ JCDATE ]
      [ JCDATEX ]
      [ JEDATE ]
      [ JEDATEX ]
  
```

Parameters: Format 1:

DATECHG/DATECHGX

Converts the input date value to the specified output date format. DATECHGX operates on dates that contain the century portion of the year.

date

A numeric value that specifies the input date.

Date can be:

- The name of a user-defined numeric variable data field
- A user-supplied numeric literal

input-date-format

Specifies the format of *date*.

Input-date-format can be:

- The date format, enclosed in quotation marks
- The name of a user-defined variable data field containing the date format

output-date-format

Specifies the format to which the input date *date* is converted.

Output-date-format can be:

- The output date format, enclosed in quotation marks
- The name of a user-defined variable data field containing the output date format

Input-date-format and *output-date-format* can be:

- **C** for calendar
- **E** for European
- **G** for Gregorian
- **J** for Julian

Format 2:

Using format 2, the first character of each function name identifies the format of the input date. The second character identifies the format to which the date is converted, as follows:

- **C** specifies calendar
- **E** specifies European
- **G** specifies Gregorian
- **J** specifies Julian

For example, the GCDATE function converts from **G**regorian to **C**alendar format.

Function names ending with **X** operate on values that contain the century portion of the year.

date

A numeric value that specifies the input date.

Examples: Using format 1 (DATECHG)

In this example, the DATECHG format of the date change function is used to convert January 28, 1958 from Gregorian to calendar format:

Statement:

```
MOVE DATECHG(580128,'G','C') TO WK-RESULT.
```

Returned value: 012858

Similarly, the DATECHGX function converts a date containing the century. Here, WK-RESULT must contain an 8 character result:

Statement:
MOVE DATECHGX(19580128,'G','C') TO WK-RESULT.

Returned value: 01281958

Using format 2 (GCDATE ...)

In this example, the GCDATE format is used to convert January 28, 1958 from Gregorian to calendar format:

Statement:
MOVE GCDATE (580128) TO WK-RESULT.

Returned value: 012858

In this example, GCDATEX is used to convert September 12, 1929 from Gregorian to calendar format. The result contains the century portion of the year:

Statement:
MOVE GCDATEX(19290912) TO WK-RESULT.

Returned value: 09121929

7.11 DATEDIF

Purpose: Returns the number of days between two specified dates.

Syntax

► DATEDIF (gregorian-date-1, gregorian-date-2) DATEDIFX ◀

Parameters

DATEDIF/DATEDIFX

Invokes the date difference function. DATEDIFX operates on values containing the century portion of the date.

gregorian-date1

Specifies the date, in Gregorian format, from which the second date is subtracted.

gregorian-date2

Specifies the date, also in Gregorian format, that is subtracted from the first date.

Gregorian-date1 and *gregorian-date2* can be:

- Names of user-defined variable data fields
- User-supplied numeric literals
- For two-digit years, the twentieth century is assumed unless year is 68 or less, in which case, the twenty-first century is assumed

Examples: Example 1

In the example below, the date difference function is used to find the number of days between January 28, 1978 and August 11, 1975:

Statement:
MOVE DATEDIF(780128,750811) TO WK-RESULT.

Returned value: 901

Note that if the dates were supplied in reverse order, the value -901 would have been returned. **Example 2**

In this example, the date difference function is used to find the number of days between January 6, 2000 and December 25, 1999, specifying the century portion of the year:

Statement:
MOVE DATEDIFX(20000106,19991225) TO WK-RESULT.

Returned value: 12

Again, if the dates were supplied in reverse order, the value -901 would have been returned.

7.12 DATEOFF

Purpose: Returns the date resulting from adding a specified number of days to a specified date.

Syntax

Parameters

DATEOFF/DATEOFFX

Invokes the date offset function. DATEOFFX operates on values that contain the century portion of the year.

gregorian-date

Specifies the date, in Gregorian format, to which the offset is added.

Gregorian-date can be:

- The name of a user-defined variable data field
- A user-supplied numeric literal

offset

Specifies the offset, in days, that is added to the specified date. *Offset* can be:

- The name of a user-defined variable data field
- A user-supplied numeric literal
- A built-in function that returns a numeric value

Offset can be negative.

Usage: DATEOFF assumes the twentieth century if the year is greater than 68, and assumes the twenty-first century if between 0 and 68. DATEOFFX allows a computation to be made in any century.

DATEOFFX assumes a continuous algorithm using the modern Gregorian calendar. It does not contain tables for historical aberrations.

Anytime a signed literal is used with DATEOFF, it should be enclosed within single quotes like this:

```
MOVE DATEOFF(911119, '-1') TO EXP-DATE
```

Examples: Example 1

In the example below, the date offset function is used to find the date that results from adding four days to January 28, 1978:

Statement:
MOVE DATEOFF(780128,4) TO WK-RESULT.

Returned value: 780201

Example 2

In this example, the date offset function is used to find the date that results from adding five days to December 28, 1999. *Gregorian-date* contains the century portion of the year, as does the returned date.

Statement:
MOVE DATEOFFX(19991228,5) TO WK-RESULT.

Returned value: 20000102

7.13 EXTRACT

Purpose: Returns the string that results from removing leading and trailing spaces from a string value.

Syntax

►— EXTRACT (string) —————►

Parameters

string

Specifies the string value on which the extract function is performed.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

Usage: When a field contains only spaces, EXTRACT returns one space. In this example:

```
FNAME="JANA      "
MID="          "
LNAME="SEDLAKOVA      "
CONCAT(EXT(FNAME), ' ', EXT(MID), ' ', EXT(LNAME))
```

Extract returns the following value:

```
"JANA  SEDLAKOVA"
```

Example: In the example below, the extract function is used to remove leading and trailing spaces from the string contained in EMP-LNAME:

```
Initial value:
EMP-LNAME: '   GAR FIELD   '
```

```
Statement:
MOVE EXTRACT(EMP-LNAME) TO WK-EXTRACTED-NAME.
```

```
Returned string:
'GAR FIELD'
```

Other examples of the extract function are provided in "CONCATENATE" and in "STRING-LENGTH" elsewhere in this chapter.

7.14 FIX

Purpose: Returns a fixed-length string of 20, 40, 60, or 80 characters.

Multiple detail lines can be produced using this string function.

Syntax

```

▶ [ FIX20 | FIX40 | FIX60 | FIX80 ] ( string )

```

Parameters

string

Specifies the string value on which the fix function is performed.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

Example: In the example below, the fix function is used to produce a formatted address list:

Statement:

```

MOVE FIX40(CONCAT(EXT(EMP-FIRST-NAME), ' ', EXT(EMP-LAST-NAME)))
  TO WK-FIX-NAME.
MOVE FIX40(EMP-STREET) TO WK-FIX-ADDR1.
MOVE FIX40(CONCAT(EXT(EMP-CITY), ' ', EXT(EMP-STATE), ' ', EXT(EMP-ZIP)))
  TO WK-FIX-ADDR2.

```

Returned string:

```

'JOHN RUPEE           '
'114 WEST INDIA ST   '
'METHUEN, MA 02312  '

```

7.15 GOODDATE

Purpose: Returns TRUE or FALSE to indicate whether a date is valid for the date type.

Syntax

Parameters

GOODDATE/GOODDATEX

Invokes the good date function. Use GOODDATEX to test dates that contain the century portion of the year.

date

A numeric value that specifies the input date.

Date can be:

- The name of a user-defined variable data field
- A user-supplied numeric literal

Note: If you are specifying a Julian date, you must specify a leading zero in the string that the process passes for Julian dates. The leading zero does not apply to non-Julian dates.

date-format

Specifies the date format for which GOODDATE or GOODDATEX tests *date*.

Date-format can be a string enclosed in quotation marks or a user-defined variable data field containing one of the following:

- **C** for calendar
- **E** for European
- **G** for Gregorian
- **J** for Julian

Example: In this example, GOODDATE tests whether the date type in the user-defined variable, MYDATE, is of date format calendar:

```

IF (GOODDATE(MYDATE, 'C')) THEN
    CALL DATECONV.
ELSE
    CALL DATERROR.

```

7.16 GOODTRAILING

Purpose: Returns TRUE or FALSE to indicate whether the value passed is a valid trailing sign field.

Syntax:

► `GOODTRAILING` (value) ◄

`GOODTRL`

Parameters

value

Specifies the numeric value whose type is tested.

Value can be:

- The name of a user-defined variable data field
- A user-supplied numeric literal

Below are values of type trailing sign:

100076+
2-

Example: In this example, the good trailing function is used to test MYNUMBER before attempting to convert it from trailing sign representation to zoned numeric.

```
IF (GOODTRL(MYNUMBER)) THEN  
    TRAILING-TO-ZONED(MYNUMBER) .  
ELSE  
    CALL NUMERROR.
```

7.17 INITCAP

Purpose: Returns the string that results when the first letter in the specified source string is capitalized and all other characters in the string are converted to lowercase.

Syntax:

▶— INITCAP (string) —▶

Parameters

string

Specifies the string whose first letter is to be capitalized.

String can be:

- A string literal enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

Example: In the example below, the initial cap function is used on the employee's last name:

Initial value:
EMP-LNAME: 'O'HEARN

Statement:
MOVE INITCAP(EMP-LNAME) TO WK-STRING.

Returned string:
'O'hearn

7.18 INSERT

Purpose: Returns the string that results from a specified string being inserted into a string value starting at a specified position.

Syntax:

►— INsert (string, insertion-string, starting-position) —————►

Parameters

string

Specifies the string into which *insertion-string* is inserted.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

insertion-string

Specifies the string that is inserted into *string*.

Insertion-string can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

starting-position

Specifies the numeric position at which insertion will begin.

Starting-position can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Starting-position is in a range from 1 to the length of *string* plus 1.

Usage:

Considerations

- If *starting-position* is 1 or less, insertion starts at the beginning of the string value.
- If *starting-position* is greater than the length of *string*, insertion starts at the end of the string value.

Example: In the example below, the INSERT function is used with the SUBSTRING function to insert the first six letters of the string contained in EMP-LNAME (PIC X(20)) into the string '**', starting at position 2:

Initial value:
EMP-LNAME: 'PARKINSON'

Statement:
MOVE INSERT('*','*',SUBS(EMP-LNAME,1,6),2) TO WK-STRING.

Returned string:
'*PARKIN*'

7.19 INVERT-SIGN

Purpose: Returns the specified numeric value with the opposite sign:

- A positive numeric value becomes negative.
- A negative numeric value becomes positive.

Syntax:

Parameters

value

Specifies the numeric value whose sign inversion value is calculated.

Value can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Example: In the example below, the sign inversion function is used to form the negative of a value if the transaction code is 'DB':

Initial values:

```
TRANS-CODE: 'DB'
WK-AMT:    453.29
```

Statements:

```
IF TRANS-CODE EQ 'DB'
  THEN
    MOVE INVERT-SIGN(WK-AMT) TO WK-AMT.
```

Returned value: -453.29

7.20 LEFT-JUSTIFY

Purpose: Returns the string that results from removing leading blanks from the left side of a string value, shifting the remainder of the string value to the left side, then filling the right side with as many blanks as were removed from the left side.

Syntax:

```

▶— LEFT-JUSTIFY ( string ) —————▶
   |
   |— LEFT-just
   |— LEFJUS

```

Parameters

string

Specifies the string value on which the left justify function is performed.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

Example: In the example below, the left justify function is used to left justify EMP-LNAME (PIC X(20)):

Initial value:
EMP-LNAME: ' SMITH '

Statement:
MOVE LEFT-JUSTIFY(EMP-LNAME) TO EMP-LNAME.

Returned string:
'SMITH '

7.21 LIKE

Purpose: Returns TRUE or FALSE when comparing a source string value with a supplied string.

Syntax:

►— LIKE (string, search-string [,escape-character]) —►

Parameters

string

Specifies the source string value being tested.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

search-string

Specifies the string used for testing *string*.

Search-string can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

Search-string is compared with *string*, one character at a time, starting with the leftmost character in each string.

All characters in the search string, except the mask characters listed below, must match the contents of *string* exactly. The mask characters are:

- **_ (underscore)**— Matches any single, non-blank character in the source string.
- **% (percent sign)**— Matches by any number of consecutive characters (zero or greater) in the source string

escape-character

Specifies a 1-character escape character that allows the current LIKE expression to search for the underscore, percent sign, and the escape character itself as an actual character.

Escape-character can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

Examples: Example 1: Testing for an embedded string

In the example below, the string contained in the field ADDRESS is evaluated for an occurrence of BOSTON within the string:

```
IF LIKE (ADDRESS, '%BOSTON%')
THEN
  DISPLAY.
```

Example 2: Testing for an embedded 4-character string starting with 'C'

In the example below, the string contained in the field PNAME is evaluated for an occurrence of a 4-character string starting with 'C':

```
IF LIKE (PNAME, '%C__ ')
THEN
  DISPLAY.
```

Example 3: Examples using an escape character

- Does AGR-NEXT-FUNCTION = '% ' ?
 IF LIKE (AGR-NEXT-FUNCTION, '%*', '*')
 This gives the same result as
 IF AGR-NEXT-FUNCTION = '% '
- Does AGR-NEXT-FUNCTION = contain a '%' ?
 IF LIKE (AGR-NEXT-FUNCTION, '%*%', '*')
 This gives the same result as
 IF AGR-NEXT-FUNCTION CONTAINS '%'
- Does AGR-NEXT-FUNCTION end with a '%' ?
 IF LIKE (AGR-NEXT-FUNCTION, '%*%', '*')
- Does AGR-NEXT-FUNCTION contain a '%A*' ?
 IF LIKE (AGR-NEXT-FUNCTION, '%*%A**%', '*')

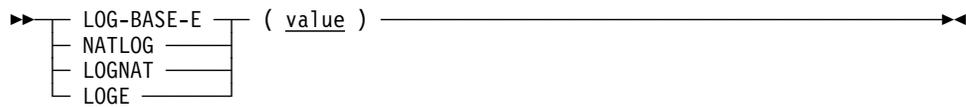
7.22 LOGARITHM

Purpose: Returns the common (base 10) or natural (base E) logarithm of a numeric value.

Syntax: Logarithm (base 10)



Logarithm (base E)



Parameters

value

Specifies the numeric value whose logarithm is calculated. *Value* can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Value must be greater than zero.

Example: In the example below, the logarithm function is used to calculate the base-10 logarithm of a numeric value:

Initial value:
WK-VALUE: 100

Statement:
MOVE LOG-BASE-10(WK-VALUE) TO WK-LOG-EQUIVALENT.

Returned value: 2

7.23 MODULO

Purpose: Returns the modulus (remainder) of one numeric value divided by another.

Syntax:

►►— MODULO (dividend, divisor) —————►►

Parameters

dividend

Specifies the numeric value that is divided by *divisor*.

divisor

Specifies the numeric value that is divided into *dividend*.

Dividend and *divisor* can be:

- Arithmetic expressions
- Names of user-defined variable data fields
- User-supplied numeric literals

Example: In the example below, the modulo function is used to find the remainder resulting from the division of two numeric values:

Initial values:
WK-VALUE1: 43
WK-VALUE2: 10

Statement:
MOVE MODULO(WK-VALUE1,WK-VALUE2) TO WK-REMAINDER.

Returned value: 3

7.24 NEXT-INT-EQHI

Purpose: Returns the smallest integer that is equal to or greater than a numeric value.

Syntax:



Parameters

value

Specifies the numeric value whose next integer equal or higher is calculated.

Value can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Example: In the example below, the next integer equal or higher function is used to raise a balance due amount to the next higher dollar value:

Initial value:
WK-BAL-DUE: 453.29

Statement:
MOVE NEXT-INT-EQHI(WK-BAL-DUE) TO WK-NEW-BAL.

Returned value: 454

7.25 NEXT-INT-EQLO

Purpose: Returns the largest integer that is equal to or less than a numeric value.

Syntax:



Parameters

value

Specifies the numeric value whose next integer equal or lower is calculated.

Value can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Example: In the example below, the next integer equal or lower function is used with the square root function to determine whether a number is the exact square of an integer value:

Initial value:

WK-VALUE: 65

Statements:

```
IF NEXIL(SQRT(WK-VALUE)) NE SQRT(WK-VALUE)
  THEN
  DISPLAY TEXT 'VALUE IS NOT AN EXACT SQUARE'.
```

Returned value from square root functions: 8.0632

Returned value from next integer function: 8

7.26 NUMERIC

Purpose: Returns TRUE or FALSE to indicate whether an alphanumeric field is a valid candidate for a MOVE to a numeric field or can be used in a computation without a data exception occurring.

Syntax:

►—— NUMERIC (value) —————►

Parameters

value

An alphanumeric value tested by the function.

Value can be:

- A user-supplied string literal
- The name of a user-defined variable data field containing the string
- A user-supplied string literal

Usage: For EBCDIC or group values, NUMERIC checks the field in isolation, without regard to possible target fields of a move or computation. For example, '999999' will test as a numeric field (TRUE), but an error would occur if this were moved to a field with the picture of 9(4) COMP-3.

NUMERIC does not support validation of floating point numbers.

Because CA-ADS and EVAL do not check the DECIMAL POINT IS clause of the OLM SYSGEN statement, NUMERIC does not either. Therefore, a period (.) and a comma (,) will always be the decimal point and the thousands separator respectively.

The types of fields tested for numeric and the tests applied to those fields are:

Field data type	Test NUMERIC applies
Binary	Always returns a TRUE value.
Packed decimal	Follows the IBM standard for what a packed field should contain; and additionally checks for a maximum field length of 16 bytes.
Zoned decimal	Follows the IBM standard for what a zoned decimal field should contain; and additionally checks for a maximum field length of 31 bytes.

Field data type	Test NUMERIC applies
EBCDIC or group values	<p>One of the following must be true:</p> <ul style="list-style-type: none"> There are 0 or more leading spaces The number starts with a plus or minus sign, or a decimal point, or a number from 0 to 9 A decimal point or number immediately follows a plus or minus sign There must be at least one digit in the number There may be no characters other than a decimal point embedded in the number There are 0 or more trailing spaces After a digit is encountered, commas are ignored
All other types	Returns a FALSE value.

In general, a single number embedded in an EBCDIC field that may contain a leading sign is considered NUMERIC.

The table below shows valid and invalid examples of NUMERIC values:

Valid examples	Invalid examples
3	.
4.4	++4
+6	.5.
.5	- . 6
-9	

Example: In the example below, NUMERIC tests whether MYALPHANUM contains a valid number:

```
IF (NUMERIC(MYALPHANUM)) THEN
    CALL NUMCALC.
ELSE
    CALL NUMERROR.
```

Initial value of MYALPHANUM: 123
Statement evaluates TRUE.

Initial value of MYALPHANUM: M123
Statement evaluates FALSE.

7.27 RANDOM-NUMBER

Purpose: Returns a pseudo-random number based on a seed numeric value. The returned random number is greater than zero and less than 1, and has a length of 9 decimal places.

Syntax:

► `RANDOM-NUMBER` (random-number-seed)

 └─┬─┘
 └─┬─┘
 RANdom

Parameters

random-number-seed

Specifies the numeric variable data field containing the seed value from which the pseudo-random number is calculated.

Random-number-seed cannot be zero.

Usage: To obtain random numbers:

1. **Set the initial random number seed value** at execution time to some varying value, such as TIME. The random seed value must not be zero.

If the result is set to a fixed value, each execution of the dialog will result in the generation of the same series of pseudo-random numbers.
2. **Move the pseudo-random number** returned by the random number function to the seed variable data field. The number returned becomes the next seed value. In this way, the random number function can generate a nonrepeating sequence of 536,870,912 numbers.
3. **Define the seed value** with a picture of 9(9) and move the result of the function to a variable with a picture of V9(9).

The result can be moved back to the seed variable by using the result as a redefinition of the seed value, as follows:

```
03 SEED-VALUE PICTURE 9(9).
03 RESULT-VALUE REDEFINES SEED-VALUE PICTURE V9(9).
```

Example: In the example below, the random number function is used to generate a sequence of ten pseudo-random numbers:

Field descriptions:

```
03 SEED-VALUE PICTURE 9(9).  
03 RESULT-VALUE REDEFINES SEED-VALUE PICTURE V9(9).  
03 RANDOM-TABLE PICTURE V9(9) OCCURS 10 TIMES.
```

Statements:

```
MOVE TIME TO SEED-VALUE.  
MOVE 1 TO WK-COUNT.  
WHILE WK-COUNT LE 10  
  REPEAT.  
    MOVE RANDOM(SEED-VALUE) TO RESULT-VALUE.  
    MOVE RESULT-VALUE TO RANDOM-TABLE(WK-COUNT).  
    ADD 1 TO WK-COUNT.  
  END.
```

7.28 REPLACE

Purpose: Returns a string that results from replacing, in a string value, each occurrence of a specified search string with a specified replacement string.

Syntax:

► REPlace (string, search-string [,replacement-string]) ◀

Parameters

string

Specifies the string value on which the replace function is performed.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

search-string

Specifies the string that the replace function searches for within the string value.

Search-string can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

replacement-string

Specifies the string that replaces each occurrence of *search-string* in the string value.

Replacement-string can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

If *replacement-string* is not specified, each occurrence of *search-string* in the string value is deleted.

Usage: The replacement string can be a different length than the search string; if this is the case, the target string value is adjusted appropriately for each replacement.

The resulting string value cannot be greater than 1,024 characters. Excess characters are truncated.

Example: In the example below, the replace function is used to replace all occurrences of BB with XXX in the string 'AABBCCBBBDD':

Statement:

```
MOVE REPLACE('AABBCCBBBDD','BB','XXX') TO WK-STRING.
```

Returned string:

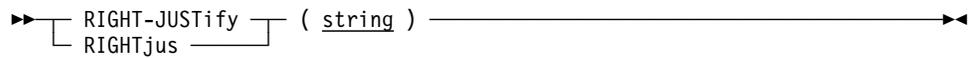
```
'AAXXCCXXBDD'
```

A further example of the replace function is provided in "SUBSTRING" later in this chapter.

7.29 RIGHT-JUSTIFY

Purpose: Returns the string that results from removing blanks on the right side of a string value, shifting the remainder of the string value to the right side, then filling the left side with as many blanks as were removed from the right side.

Syntax:

► `RIGHT-JUSTify` (string) 

Parameters

string

Specifies the string value that is right justified.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

Example: In the example below, the right justify function is used to right justify EMP-LNAME (PIC X(20)):

Initial value:
EMP-LNAME: ' SMITH '

Statement:
MOVE RIGHT-JUSTIFY(EMP-LNAME) TO EMP-LNAME.

Returned string:
' SMITH'

7.30 SIGN-VALUE

Purpose: Returns a +1, 0, or -1, depending on whether the specified numeric value is positive, zero, or negative, respectively.

Syntax:

► $\left[\begin{array}{l} \text{SIGN-VALue} \\ \text{SIGV} \end{array} \right] (\text{value}) \longleftarrow \longrightarrow$

Parameters

value

Specifies the numeric value whose sign is determined.

Value can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Example: In the example below, the sign value function is used to move a zero to a transaction code field if an amount is negative, and a 1 to the field if the amount is zero or positive. On mapout, the transaction code field can be decoded to CR or DB:

Initial value:

WK-AMT: -453.29

Statements:

```
MOVE SIGN-VALUE(WK-AMT) + 1 TO TRANS-CODE.
IF TRANS-CODE EQ 2
  THEN
    MOVE 1 TO TRANS-CODE.
```

Returned value from function: -1

Result of MOVE expression: 0

7.31 SINE

Purpose: Returns the sine of a numeric value that represents an angle in either degrees or radians.

Syntax: Sine (degrees):

► SINE-DEGREES (value)

├ SINEDEG

└ SIND

Sine (radians):

► SINE-RADIANS (value)

├ SINERAD

└ SINR

Parameters

SINE-DEGREES

Returns the sine value in degrees.

SINE-RADIANS

Returns the sine value in radians.

value

Specifies the numeric value representing the angle, in degrees or radians, whose sine is calculated.

Value can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Example: In the following example, the sine (degrees) of -60 is calculated and moved to WK-RESULT (PIC S999V9999):

```
MOVE SIND(-60) TO WK-RESULT.
```

Return value: -0.8660

7.32 SQUARE-ROOT

Purpose: Returns the square root of a numeric value.

Syntax:

► $\left[\begin{array}{l} \text{SQUARE-ROOT} \\ \text{SQRT} \end{array} \right] (\text{value}) \longleftarrow \longrightarrow$

Parameters

value

Specifies the numeric value whose square root is calculated. *Value* can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Value cannot be a negative number.

Example: In the example below, the square root function is used to calculate the square root of a number:

Initial value:

WK-VALUE: 256

Statement:

MOVE SQUARE-ROOT(WK-VALUE) TO WK-RESULT.

Returned value: 16

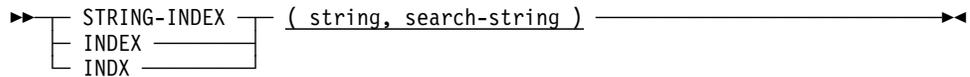
Another example of the square root function is provided in "NEXT-INT-EQLO" earlier in this chapter.

7.33 STRING-INDEX

Purpose: Returns the starting position of a specified string within a string value.

If the specified string is not found, a zero is returned.

Syntax:



Parameters

string

Specifies the string that is searched.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

search-string

Specifies the string that the index function searches for within *string*.

Search-string can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

Search-string cannot be longer than *string*.

Example: In the example below, the index function is used to test whether a product code contains the string 'ABC':

Initial value:
PROD-CODE: '12AB43 ABC3254'

Statements:
IF INDX(PROD-CODE, 'ABC') EQ 0
THEN
 DISPLAY TEXT 'INVALID PRODUCT CODE'.

Returned value from function: 8

Since the string 'ABC' appears in the product code (starting at character position 8), the condition is false.

7.34 STRING-LENGTH

Purpose: Returns the length of a string value.

Syntax

► `STRING-LENGTH` (string) 

Parameters

string

Specifies the string value whose length is determined.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

Example: In the example below, the length of a name contained in EMP-LNAME (PIC X(20)) is determined. To calculate the length of a string value, excluding leading and trailing spaces, the length function is used in conjunction with the extract function, as follows:

Initial value:
EMP-LNAME: 'SMITH'

Statement:
MOVE SLENGTH(EXTRACT(EMP-LNAME)) TO WK-NAME-LENGTH.

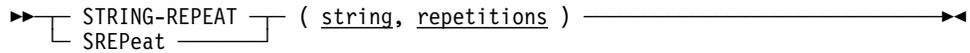
Returned string from extract function:
'SMITH'

Returned value from length function: 5

7.35 STRING-REPEAT

Purpose: Returns the string that results from repeating a string value a specified number of times.

Syntax

►  (string, repetitions)

Parameters

string

Specifies the string value that is repeated.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

repetitions

Specifies the numeric value representing the number of times that the string value is to be repeated.

Repetitions can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Example: In the example below, the repeat function is used to repeat the constant 'NAME' two times:

Statement:
MOVE SREPEAT('NAME',2) TO WK-TARGET.

Returned string:
'NAMENAME'

7.36 SUBSTRING

Purpose: Returns the substring of a string value, starting from a specified position and continuing for a specified length.

Syntax:

Parameters

string

Specifies the string value from which the substring is taken.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

starting-position

Specifies the numeric starting position of the substring within the string value.

Starting-position can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Starting-position must be positive and not greater than the length of *string*.

length

Specifies the numeric length of the substring within the string value.

Length can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

The sum of *starting-position* and *length*, minus 1, cannot be greater than the length of *string*.

If *length* is not specified, the substring is taken from the specified starting position to the end of the string value.

Examples: Example 1: Extracting a substring

In the example below, the substring function is used to extract a substring of EMP-LNAME (PIC X(20)), starting at position 4 and continuing for a length of 3:

```
Initial value:
EMP-LNAME: 'SMITH          '

Statement:
MOVE SUBSTR(EMP-LNAME,4,3) TO WK-NAME.

Returned string:
'TH '
```

Example 2: Replacing a leading zero

In the next example, the substring function is used in conjunction with the verify and concatenate functions to replace each leading zero in a number stored in WK-AMT (PIC X(10)) with an asterisk (*):

```
Initial value:
WK-AMT: '000500.43 '

Statements:
MOVE VERIFY(WK-AMT,'0') TO WK-START-POSITION.
IF WK-START-POSITION GT 1
THEN
MOVE CON(REP(SUBS(WK-AMT,1,WK-START-POSITION - 1),'0','*'),
SUBS(WK-AMT,WK-START-POSITION)) TO WK-EDITED.

Returned value from verify function:          4
Returned string from first substring function: '000'
Returned string from replace function:       '***'
Returned string from second substring function: '500.43 '
Returned string from concatenate function:    '***500.43 '
```

The string '***500.43 ', with a length of ten characters, is moved to the field WK-EDITED. Note that the MOVE VERIFY command in the above example locates the position of the first nonzero character in WK-AMT.

Another example of the substring function is provided in "INSERT" earlier in this chapter.

7.37 TANGENT

Purpose: Returns the tangent of a numeric value that represents an angle in either degrees or radians.

Syntax: Tangent (degrees):

►► $\left[\begin{array}{l} \text{TANGENT-DEGREES} \\ \text{TANDeg} \end{array} \right] (\text{value})$ ►►

Tangent (radians):

►► $\left[\begin{array}{l} \text{TANGENT-RADIANS} \\ \text{TANRad} \end{array} \right] (\text{value})$ ►►

Parameters

TANGENT-DEGREES

Returns the tangent value in degrees.

TANGENT-RADIANS

Returns the tangent value in radians.

value

Specifies the numeric value representing the angle, in degrees or radians, whose tangent is calculated.

Value can be:

- An arithmetic expression
- The name of a user-defined variable data field
- A user-supplied numeric literal

Value cannot equal values such as -270, +270, -90, or +90 in the tangent (degrees) function, and cannot equal values such as $-\pi/2$ or $+\pi/2$ in the tangent (radians) function.

Usage

- **For the tangent (degrees) function,** *value* cannot be a value equal to the following expression, where *n* is any integer:

$$(\underline{n} * 180) + 90$$

- **For the tangent (radians) function,** *value* cannot be a value equal to the following expression:

$$(\underline{n} * \pi) + \pi/2$$

Example: In the following example, the tangent (degrees) of 60 is calculated and moved to WK-RESULT (PIC S999V9999):

```
MOVE TAND(60) TO WK-RESULT.
```

Returned value: 1.7321

7.38 TODAY

Purpose: Returns today's date in the format requested.

Syntax

► `TODAY` | `TODAYX` (`date-format`)

Parameters

TODAY/TODAYX

Invokes the today function. `TODAYX` returns a date that contains the century portion of the year.

date-format

Specifies the output date format. *Date-format* can be:

- The date format, enclosed in quotation marks
- The name of a user-defined variable data field containing the date format

Date-format can be:

- **C** for calendar
- **E** for European
- **G** for Gregorian
- **J** for Julian

Examples: Example 1

In the example below, the today function is used to display today's date in the calendar format (where today is March 17, 1989):

Statement:
`MOVE TODAY('C') TO WK-RESULT.`

Returned value: 031789

Example 2

In this example, the today function is used to return today's date in the calendar format (where today is October 30, 1990). The returned date contains the century portion of the year:

Statement:
`MOVE TODAYX('C') TO WK-RESULT.`

Returned value: 10301990

7.39 TOLOWER

Purpose: Returns the string that results from converting all characters to lowercase.

Syntax:

►— TOLOWER (string) —————►

Parameters

string

Specifies the string value on which the lowercase function is performed.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

Example: In the example below, the lowercase function is used to convert all characters in the last name to lowercase:

Initial value:
EMP-LNAME: 'LANCHESTER'

Statement:
MOVE TOLOWER(EMP-LNAME) TO WK-EMP-LNAME.

Returned string:
'lanchester'

7.40 TOMORROW

Purpose: Returns tomorrow's date in the format requested.

Syntax

► `TOMORROW` `TOMORROWX` (date-format)

Parameters

TOMORROW/TOMORROWX

Invokes the tomorrow function. `TOMORROWX` returns a value that contains the century portion of the year.

date-format

Specifies the output date format. *Date-format* can be:

- A date format, enclosed in quotation marks
- The name of a user-defined variable data field containing the date format

Date-format can be:

- **C** for calendar
- **E** for European
- **G** for Gregorian
- **J** for Julian

Examples: Example 1

In the example below, the tomorrow function is used to display tomorrow's date in the calendar format (where today is March 17, 1989):

Statement:
`MOVE TOMORROW('C') TO WK-RESULT.`

Returned value: 031889

Example 2

In this example, the tomorrow function is used to return tomorrow's date in the calendar format (where today is October 30, 1990). The returned date contains the century portion of the year:

Statement:
`MOVE TOMORROWX('C') TO WK-RESULT.`

Returned value: 10311990

7.41 TOUPPER

Purpose: Returns the string that results from converting all characters to uppercase.

Syntax

▶— TOUPPER (string) —▶

Parameters

string

Specifies the string value on which the uppercase function is performed.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

Example: In the example below, the uppercase function is used to convert all characters in the last name to uppercase:

Initial value:
EMP-LNAME: 'Lanchester'

Statement:
MOVE TOUPPER(EMP-LNAME) TO WK-EMP-LNAME.

Returned string:
'LANCHESTER'

7.42 TRAILING-TO-ZONED

Purpose: Returns a zoned numeric from a COBOL trailing sign numeric.

Syntax:

►► TRAILING-TO-ZONED (value) ◄◄
 TRAILZN

Parameters

value

Specifies the COBOL trailing sign numeric value on which the trailing to zoned function is performed.

Value can be:

- The name of a user-defined variable data field in trailing sign format
- A user-supplied numeric literal

Example: In the example below, the trailing to zoned function is used to convert the value of MYNUMBER to a zoned numeric:

Initial value:
MYNUMBER: 123-

Statement:
MOVE TRAILZN(MYNUMBER) TO WK-PART-CODE.

Returned value:
WK-PART-CODE: 123 negative (hex 'F1F2D3')

7.43 TRANSLATE

Purpose: Returns the string that results from translating characters in a string value.

The characters are translated to corresponding characters that are specified in a substitution string:

- Characters in a selection string correspond by position to characters in a substitution string.
- Each character in the string value specified in the selection string is translated to the corresponding character contained in the substitution string.

Syntax:

```
►— TRANSLate ( string, substitution-string ————— selection-string ) —◄
```

Parameters

string

Specifies the string value on which the translate function is performed.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

substitution-string

Specifies the substitution string.

Substitution-string can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the substitution string

selection-string

Specifies the selection string. Characters in *selection-string* will be replaced by corresponding characters in *substitution-string*.

Selection-string can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the selection string

Usage:

Considerations

- If *selection-string* is longer than *substitution-string*, the excess characters in *selection-string* correspond to blanks.

- If *selection-string* specifies the same character more than once, the translate function uses the first occurrence of the character.
- If *selection-string* is not specified, the 256-character EBCDIC table is used, consisting of hexadecimals 00 through FF.

Example: In the example below, the translate function is used to translate all occurrences in PART-CODE (PIC X(20)) of the characters A, B, C, and D (*selection-string*), to W, blank, Y, and Z (*substitution-string*), respectively:

Initial value:

```
PART-CODE: 'B53A22B1E50D40C94  '
```

Statement:

```
MOVE TRANS(PART-CODE,'W YZ','ABCD') TO WK-PART-CODE.
```

Returned string:

```
' 53W22 1E50Z40Y94  '
```

7.44 VERIFY

Purpose: Returns the position of the first character in a string value that does not occur in a verification string.

If every character in the input string value occurs in the verification string, a zero is returned.

Syntax:

►— VERIFY (string, verification-string) —►

Parameters

string

Specifies the string value on which the verify function is performed.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

verification-string

Specifies the string value against whose characters the string value's characters are verified.

Verification-string can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

Example: In the example below, the verify function is used to verify that WK-NUMBER (PIC X(10)) contains only numeric values or blanks:

```
Statement:  
  IF VER(WK-NUMBER, '0123456789 ') NE 0  
  THEN  
    DISPLAY TEXT 'INVALID SPECIFICATION FOR NUMERIC FIELD'.
```

Another example of the verify function is provided in "SUBSTRING" earlier in this chapter.

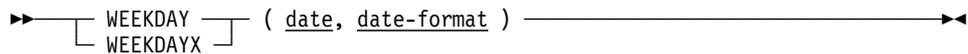
7.45 WEEKDAY

Purpose: Returns the weekday (Monday, Tuesday, etc.) of a specified date.

Weekday functions can be coded in two ways, as shown in the syntax diagrams below.

Syntax

Format 1::



Format 2:



Parameters

Format 1:

WEEKDAY/WEEKDAYX

Invokes the weekday function. **WEEKDAYX** operates on dates that contain the century portion of the year.

date

A numeric value that specifies the input date. *Date* can be:

- The date, enclosed in quotation marks
- The name of a user-defined variable data field containing the date

date-format

Specifies the format of the date specified by *date*. *Date-format* can be:

- The date format, enclosed in quotation marks
- The name of a user-defined variable data field containing the date format

Date-format can be:

- **C** for calendar
- **E** for European
- **G** for Gregorian
- **J** for Julian

Format 2:

GWEEKDAY/GWEEKDAYX
CWEEKDAY/CWEEKDAYX
EWEEKDAY/EWEEKDAYX
JWEEKDAY/JWEEKDAYX

The invocation names of the alternate formats of the weekday function. The prefix C, E, G, or J of an invocation name identifies the format of the date specified by *date* (calendar, European, Gregorian, or Julian.)

Invocation names ending in X operate on dates that contain the century portion of the year.

date

A numeric value that specifies the input date. *Date* can be:

- The name of a user-defined variable data field
- A user-supplied numeric literal

Examples

Example 1 (Format 1): In the example below, the weekday function is used to determine on which weekday January 28, 1958 fell. The date is provided in calendar format:

Statement:

```
MOVE WEEKDAY(012858,'C') TO WK-RESULT.
```

Returned value: 'TUESDAY'

Example 2 (Format 1)

This example returns the weekday for a date that contains the century portion of the year:

Statement:

```
MOVE WEEKDAYX(01281958,'C') TO WK-RESULT.
```

Returned value: 'TUESDAY'

Example 3 (Format 2)

This is equivalent to Example 2:

Statement:

```
MOVE CWEEKDAY(01281958) TO WK-RESULT.
```

Returned value: 'TUESDAY'

7.46 WORDCAP

Purpose: Returns the string that results when the first letter of each word in the specified source string is capitalized and all other characters in the string are converted to lowercase.

Syntax

►— WORDCAP (string) —————►◀

Syntax Rule

string

Specifies the string to be converted.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

The first letter in each word is capitalized and all other characters are converted to lowercase.

Example: In the example below, the word cap function is used on the employee's name:

Initial value:

```
EMP-LNAME: 'O'HEARN
```

Statement:

```
MOVE WORDCAP(EMP-LNAME) TO WK-STRING.
```

Returned string:

```
'O'Hearn
```

7.47 YESTERDAY

Purpose: Returns yesterday's date in the format requested.

Syntax

►►

YESTERDAY	(<u>date-format</u>)	◄◄
YESTERDAYX		

Parameters

YESTERDAY/YESTERDAYX

Invokes the yesterday function. YESTERDAYX returns a date that contains the century portion of the year.

date-format

Specifies the output date format. *Date-format* can be expressed using:

- The date format, enclosed in single quotation marks
- The name of a user-defined variable data field that contains the date format

Date-format can be:

- **C** for calendar
- **E** for European
- **G** for Gregorian
- **J** for Julian

Examples: Example 1

In the example below, the yesterday function is used to display yesterday's date in the calendar format (where today is March 17, 1997).

Statement:
MOVE YESTERDAY('C') TO WK-RESULT.

Returned value: 031697

Example 2

This example uses YESTERDAYX to return a date containing the century:

Statement:
MOVE YESTERDAYX('C') TO WK-RESULT.

Returned value: 03161997

7.48 ZONED-TO-TRAILING

Purpose: Returns a COBOL trailing sign numeric from a zoned numeric.

Syntax:

Parameters

value

Specifies the zoned numeric value on which the zoned to trailing function is performed.

Value is the name of a user-defined variable data field in zoned numeric format.

Example: In the example below, the zoned to trailing function is used to convert the value of MYNUMBER to a COBOL trailing sign numeric:

Initial value:

MYNUMBER: 123 negative (hex 'F1F2D3')

Statement:

MOVE ZNTRAIL(MYNUMBER) TO WK-PART-CODE.

Returned value:

WK-PART-CODE: 123-

Chapter 8. Conditional Expressions

- 8.1 Overview 8-3
- 8.2 General considerations 8-4
 - 8.2.1 Syntax for conditional expressions 8-4
- 8.3 Batch-control event condition 8-6
- 8.4 Command status condition 8-7
- 8.5 Comparison condition 8-10
- 8.6 Cursor position condition 8-12
- 8.7 Dialog execution status condition 8-14
- 8.8 Environment status condition 8-16
- 8.9 Level-88 condition 8-17
- 8.10 Map field status condition 8-18
- 8.11 Map paging status conditions 8-22
- 8.12 Set status condition 8-25
- 8.13 Arithmetic and assignment command status condition 8-27

8.1 Overview

A conditional expression specifies test conditions in an IF or WHILE command. The outcome of a conditional test determines the processing that occurs.

A conditional expression can be used as a variable wherever the command syntax specifies conditional-expression.

The table below summarizes the test conditions that can be used in conditional expressions. Each condition is described separately in this section.

Summary of test conditions

Condition	Purpose
Batch control event	Determines the occurrence of runtime events (batch input only)
Command status	Tests for the presence of a status code in a dialog's error-status field
Comparison	Compares two values
Cursor position	Determines if the cursor is located in a specified field after a mapin operation
Dialog execution status	Determines if a dialog is executing for the first time
Environment status	Determines the environment in which the application is executed
Level-88 condition name	Determines if a variable data field value is equal to the value of the associated level-88 condition name
Map field status	Determines if a map's data field are changed or in error
Map paging status	Determines the runtime events of a map paging session
Set status	Determines member record occurrences or if a record is a member of a specific set
Assignment condition	Tests for an arithmetic or assignment exception

8.2 General considerations

Contents: Conditional expressions can contain:

- A single test condition
 - Two or more test conditions combined with the logical operators AND and OR
 - The logical operator NOT to specify the opposite of the condition
- NOT can precede a single condition or a compound condition enclosed in parentheses.

Evaluation of operators: Operators in a conditional expression are evaluated one at a time, from left to right, in the following order of precedence:

- Unary plus or minus
- Multiplication or division
- Addition or subtraction
- MATCHES or CONTAINS keywords
- EQ, NE, GT, LT, GE, LE operators
- NOT
- AND
- OR

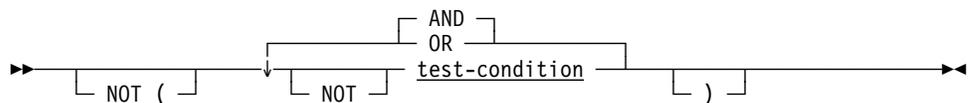
The default order of precedence can be overridden by using parentheses. The expression in the innermost parentheses is evaluated first.

Significant tests in conditional expressions should be coded to the left for greater runtime efficiency.

8.2.1 Syntax for conditional expressions

The conditional expression syntax shown below applies when the command syntax specifies *conditional-expression*.

Syntax



Parameters

NOT

Specifies that the opposite of a condition fulfills the test requirements.

The opposite of the entire conditional expression can be specified by enclosing the expression in parentheses and preceding it with NOT.

test-condition

Specifies the condition being tested and can include parentheses

AND

Specifies the expression is true only if the outcome of both test conditions is true.

OR

Specifies the expression is true if the outcome of either one or both test conditions is true.

8.3 Batch-control event condition

Purpose: (CA-ADS/Batch only) Tests the occurrence of runtime events, such as end-of-file or physical input errors, specific to batch input.

The event status is initialized at the beginning of application execution and the outcome of each test is false.

Syntax



Parameters

\$END-OF-FILE

Tests whether the most recent input file read operation results in an end-of-file condition.

\$IOERRor

Tests whether the most recent input file read operation results in a physical input error.

A physical error on a write operation causes the application to abort.

Example: In the example below, execution of a group of commands continues until an end-of-file condition occurs:

```
WHILE NOT $EOF
  REPEAT.
  .
  .
  WRITE TRANSACTION.
  END.
LEAVE APPLICATION.
```

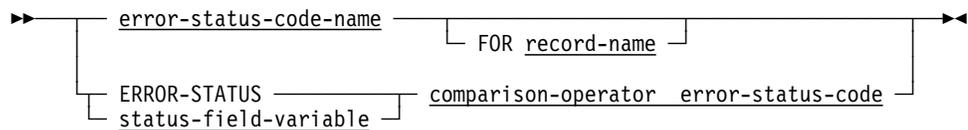
8.4 Command status condition

Purpose: Tests a dialog's error-status field for the presence of a specified status code, following the execution of a process command that involves database, queue, or scratch activity, or a WRITE PRINTER utility command.

The command status is checked by testing the error-status field for a specified status code or by testing a level-88 condition name. Level-88 condition names and status field names other than ERROR-STATUS must be defined in the dialog's status definition record.

►► For a discussion of status definition records, see Chapter 10, "Error Handling."

Syntax



Parameters

error-status-code-name:

The name of a level-88 condition defined in the dialog's status definition record.

FOR record-name:

Specifies that the test applies to the last database command involving the named record.

Record-name must be known to the dialog's subschema.

ERROR-STATUS

Represents the value contained in the internal error-status field for the dialog.

status-field-variable

Specifies the name of a user-supplied data field that contains the error-status field for the dialog.

Status-field-variable must be defined in the dialog's status definition record.

comparison-operator:

The comparison operators are:

Operator	Synonym	Meaning
EQ	=	Equal
NE		Not equal to
GT	>	Greater than
LT	<	Less than
GE		Greater than or equal to
LE		Less than or equal to

error-status-code:

Specifies the status code to which the value in *status-field-variable* is compared.

Error-status-code is:

- the name of a variable data field that contains the status code
- the code itself (optionally enclosed in single quotation marks)
- an expression, including a built-in function, that returns the status code

Examples: Example 1: Testing for a database record status

The command status condition in the following IF statement is true when the dialog's error-status field contains the status code 0326:

```
IF DB-REC-NOT-FOUND THEN ...
```

DB-REC-NOT-FOUND must be defined in the dialog's status definition record.

Example 2: Testing for the end-of-set

The command status condition in the following IF statement is true when the dialog's error-status field does not contain the status code 0307:

```
IF NOT DB-END-OF-SET THEN ...
```

DB-END-OF-SET must be defined in the dialog's status definition record.

Example 3: Testing for the status of a database record

The command status condition in the following IF statement is true when the dialog's error-status field contains the status code 0000 following execution of the most recent command involving a CUSTOMER record:

```
IF DB-STATUS-OK FOR CUSTOMER THEN ...
```

DB-STATUS-OK must be defined in the dialog's status definition record.

Example 4: Testing for a dialog's error status

The command status condition in the following IF statement is true when the dialog's error-status field contains the status code 0307:

```
IF ERROR-STATUS IS '0307'...
```

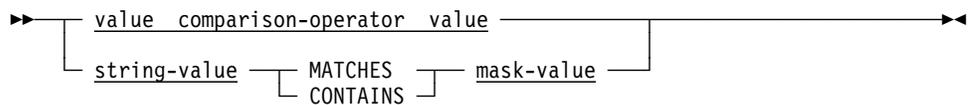
8.5 Comparison condition

Purpose: Compares two values.

Each value can be a variable data field, an arithmetic expression, a built-in function, or a numeric, nonnumeric, multi-bit binary, or figurative constant.

A comparison condition also compares two EBCDIC, DBCS, or unsigned zoned decimal character strings to determine if the first string matches or contains the second string.

Syntax



Parameters

value:

Identifies the operands being compared. *Value* is specified according to the rules presented in Chapter 5, "Introduction to Process Language."

comparison-operator:

The comparison operators are:

Operator	Synonym	Meaning
EQ	=	Equal
NE		Not equal to
GT	>	Greater than
LT	<	Less than
GE		Greater than or equal to
LE		Less than or equal to

string-value:

Either the name of an elementary DBCDIC, DBCS, or unsigned zoned decimal data field that contains the character string being compared, or the string itself enclosed in single quotation marks.

MATCHES

Compares the left operand to the right operand, one character at a time, beginning with the leftmost character in each operand.

The length of the string that is compared is set to the length of the shorter of the two operands. If a character in the left operand does not match the corresponding character in the right operand, the outcome of the comparison is false.

CONTAINS

Searches the left operand for an occurrence of the right operand.

The length of the right operand must be less than or equal to the length of the left operand. If the right operand is not entirely contained in the left operand, the outcome of the comparison is false.

mask-value:

Either the name of a variable data field that contains the mask value or the value itself enclosed in single quotation marks.

Usage:

Considerations: Special mask characters in *mask-value* match characters in *value* according to the following conventions:

- @ -- Matches any alphabetic character
- # -- Matches any numeric character
- * -- Matches any character

Any other character in *mask-value* matches only itself in *value*.

Examples: Example 1: Using a simple comparison

The comparison condition in the following IF statement is true when the value in the SALES field is greater than or equal to 5000:

```
IF SALES GE 5000 ...
```

Example 2: Using a compound comparison

The comparison condition in the following IF statement is true when the value in the CODE field is not equal to X3 and the value in the QTY field is less than 15:

```
IF CODE NE 'X3' AND QTY LT 15 ...
```

Example 3: Searching for a given string occurrence

The comparison condition in the following IF statement is true when the character string TOM occurs in the character string contained in the NAME field:

```
IF NAME CONTAINS 'TOM' ...
```

Example 4: Using a comparison to a given string value

The comparison condition in the following IF statement is true when the character string contained in the PART-ID field matches the mask value **@398:

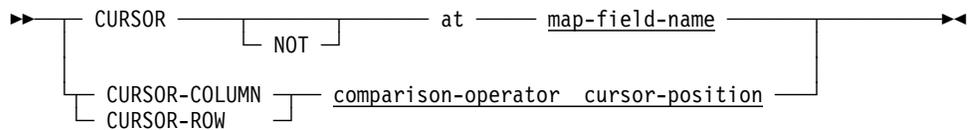
```
IF PART-ID MATCHES '**@398' ...
```

8.6 Cursor position condition

Purpose: Determines whether the cursor is located in a specified field following a mapin operation.

The named map field can be tested for the presence of the cursor, or a comparison of the cursor column or row position to a specified value can be made following mapin.

Syntax:



Parameters

NOT

Specifies the condition is true only when the cursor is not located in the named map field.

map-field-name:

Tests the named map field for the presence of the cursor.

CURSOR-COLUMN

Specifies that the comparison with *curson-position* is made using the value in the CURSOR-COLUMN.

CURSOR-ROW

Specifies that the comparison with *curson-position* is made using the value in the CURSOR-ROW field.

comparison-operator:

The comparison operators are:

Operator	Synonym	Meaning
EQ	=	Equal
NE		Not equal to
GT	>	Greater than
LT	<	Less than
GE		Greater than or equal to
LE		Less than or equal to

cursor-position:

Specifies the value being compared to the value in the CURSOR-COLUMN or CURSOR-ROW field. The specified value should correspond to a possible cursor column or row position on the terminal in use.

Cursor-position is a value variable, arithmetic expression, or numeric constant that is specified according to the rules presented in this manual.

8.7 Dialog execution status condition

Purpose: Determines whether a dialog is executing for the first time in an application thread.

Syntax:

►—— FIRST-TIME —————►

Usage

Dialog execution status test outcomes: When a dialog executes for the first time, the CA-ADS runtime system sets the execution status to FIRST-TIME and the outcome of the execution status test is true. The outcome of a subsequent test depends on the control command that precedes the test, as shown in the table below.

Control command	Status test outcome
DISPLAY	False.
EXECUTE NEXT FUNCTION	Depends on the control command (TRANSFER, INVOKE, LINK, or RETURN) associated with the selected application response.
INVOKE	False for the dialog issuing the INVOKE command.
LEAVE	Not applicable. The application is no longer operative.
LINK	Unchanged for the dialog issuing the LINK command.
RETURN	Not applicable. The dialog is no longer operative in the application thread. If the dialog issuing the RETURN command is invoked or linked to again, the dialog execution status is reset to FIRST-TIME.
TRANSFER	Not applicable. The dialog is no longer operative in the application thread. If a dialog transfers to itself, the dialog execution status is reset to FIRST-TIME.
READ	False.
WRITE	False.
CONTINUE	False.

►► For detailed descriptions of the control commands, see Chapter 15, “Control Commands.”

Example: The example below shows the use of the dialog execution status condition:

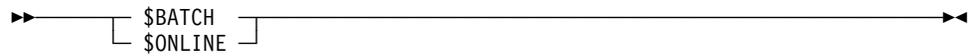
```
IF FIRST-TIME
THEN
  MOVE 1 TO COUNTER.
ELSE
  ADD 1 TO COUNTER.
```

8.8 Environment status condition

Purpose: Determines the application's environment.

Status conditions can be tested in both online and batch environments.

Syntax:



Parameters

\$BATCH

Is true when the dialog is executing in the batch environment.

\$ONLINE

Is true when the dialog is executing in the online environment.

Example: In the following example, different types of processing are performed, depending on the runtime environment.

```
IF $ONLINE
  THEN
    DISPLAY.
  ELSE
    WRITE TRANSACTION.
```

8.9 Level-88 condition

Purpose: Determines whether the value contained in a variable data field is equal to a value associated with a level-88 condition name defined for that field. CA-ADS checks for:

- Single or multiple values
- Single or multiple ranges of values
- Any combination of values and ranges of values

Syntax:

▶▶—— condition-name ——▶▶

Parameters

condition-name:

Specifies the condition being tested.

Condition-name must be defined as a level-88 condition name in a data dictionary or subschema record used by the dialog.

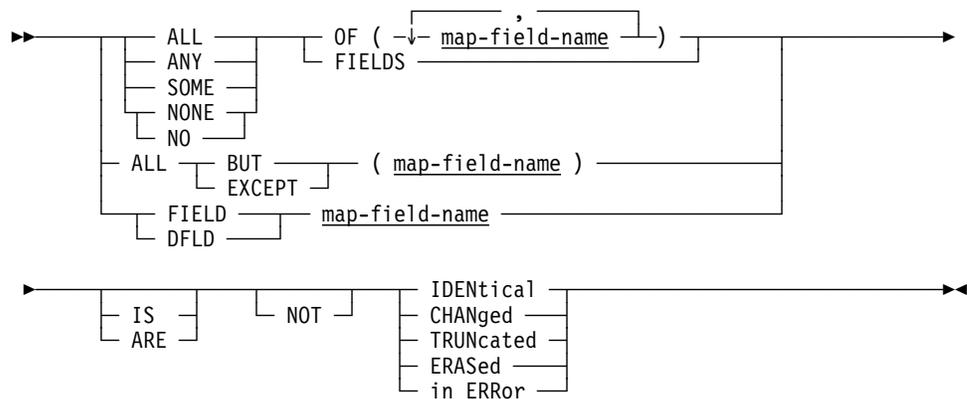
8.10 Map field status condition

Purpose: Determines if one or more of a map's data fields are changed, identical, truncated, erased, or in error.

A map field status condition applies to the status of the tested map data fields at the time of the most recent mapin operation. The IN ERROR status condition also applies to the status of the map fields following a map modification command that specifies EDIT IS ERROR/CORRECT.

Map field status tests cannot be used to test the condition of system-supplied \$MESSAGE, \$RESPONSE, and \$PAGE fields.

Syntax:



Parameters

ALL

The outcome of the test must be true for every specified field.

ANY

The outcome of the test must be true for one or more of the specified fields.

NONE

The outcome of the test must be true for none of the specified fields.

NO can be used in place of NONE.

SOME

The outcome of the test must be true for at least one but not all of the specified fields.

OF (map-field-name)

Specifies a data field in the dialog's map.

One or more fields, up to the number of fields defined for the map, can be specified inside the parentheses.

FIELDS

Specifies all data fields in the dialog's map.

ALL BUT map-field-name:

Specifies that all map fields are to be tested, except for the fields specified by *map-field-name*

EXCEPT can be used instead of ALL BUT.

Map-field-name specifies a data field in the dialog's map.

One or more fields can be specified, up to the number of fields defined for the map, inside the parentheses.

FIELD map-field-name:

Explicitly names one map field for which the outcome of the test must be true.

Map-field-name must be a data field known to the dialog's map.

DFLD can be used in place of FIELD.

NOT

Specifies that a test is for the opposite of the specified status.

IDENTICAL

At the time of the most recent mapin from the terminal, the contents of the mapped-in field are compared with the original contents of the dialog's record buffer.

The condition is true if:

- The field's modified data tag (MDT) is off. On mapin, the MDT is off if the user did not type any characters in the field.

Note: The MDT can also be set at mapout, depending on the map's definition and any MODIFY MAP commands issued before mapout.

- The field's MDT is on, but each character in the input data is exactly the same (including capitalization) as data that was originally mapped out for the field.

CHANGED

At the time of the most recent mapin from the terminal, the field's modified data tag (MDT) is checked to determine if the end user has changed the field.

When erase EOF is pressed at the beginning of a field, the MDT is set; however, the changed condition is only true if you have specified a pad character.

The condition is true if:

- The MDT is on for the field. The MDT is on if any characters are typed in the field during mapin. This is true even if the characters are the same as those that are mapped out.

Note: The MDT can also be set at mapout, depending on the map's definition and any MODIFY MAP commands issued before mapout.

TRUNcated

At the time of the most recent mapin from the terminal, CA-ADS truncates excess data entered in the specified map fields.

ERASed

At the time of the most recent mapin from the terminal, the terminal operator erased all data in specified map fields.

in ERRor

At the time of the most recent mapin from the terminal, specified map fields contain erroneous data or were given the EDIT IS ERROR attribute in a map modification command.

Note: You do not have to wait for mapin. You can set fields and immediately test them.

Automatic editing affects the use of the IN ERROR status condition as follows:

- If automatic editing is enabled and EXECUTE ON EDIT ERRORS is YES, fields that contain erroneous data are set in error and control is returned to the response process. Error tests can be made by using the IN ERROR status condition.

Note: The above does not apply for pageable map detail areas.

- If automatic editing is enabled and EXECUTE ON EDIT ERRORS is NO, CA-ADS returns control to the mapout operation, displays specified error messages, and waits for the user to enter valid data.

Note: The above does not apply for pageable map detail areas.

- If automatic editing is not enabled for the map, fields that contain erroneous data are not automatically set in error. To make use of the IN ERROR status condition, the fields in error must be flagged by using the MODIFY MAP command.

EXECUTE ON EDIT ERRORS is specified on the Process Modules screen.

►► For information about this screen, see Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

For a description of the MODIFY MAP command, see Chapter 17, “Map Commands.”

Note: Map fields in error are not mapped in. Variable storage contains the values of the fields prior to the last mapout operation.

Usage:

Pageable map considerations

- Conditions set for a data field are cumulative. If a map field is changed, identical, truncated, erased, and/or in error at any time during a pseudo-converse, the field is considered changed, identical, truncated, erased, or in error when control transfers to a response process.

- A test on a detail area map field applies to the detail occurrence referenced by the most recent pageable map command following the last pseudo-converse.
- After a PUT DETAIL command, the outcome of all tests on detail area map fields is false.

Examples: Example 1: Testing for field changes

The map field status condition in the following IF statement is true when the user modifies any map field:

```
IF ANY FIELD IS CHANGED ...
```

Example 2: Testing for modified data

The map field status condition in the following IF statement is true if the input data is identical to data initially displayed on the map. In this example, the user is asked to specify another department if no change is made to the department id or name:

```
IF FIELD DEPT-ID-0410 IS IDENTICAL
   AND FIELD DEPT-NAME-0410 IS IDENTICAL
   THEN
   DISPLAY MSG TEXT
   'PLEASE SPECIFY NEXT DEPARTMENT'.
```

Example 3: Testing for field truncation

The map field status condition in the following IF statement is true when excess data entered in the CUST-CITY field has been truncated during the mapin operation:

```
IF FIELD CUST-CITY IS TRUNCATED ...
```

Example 4: Testing for erased data

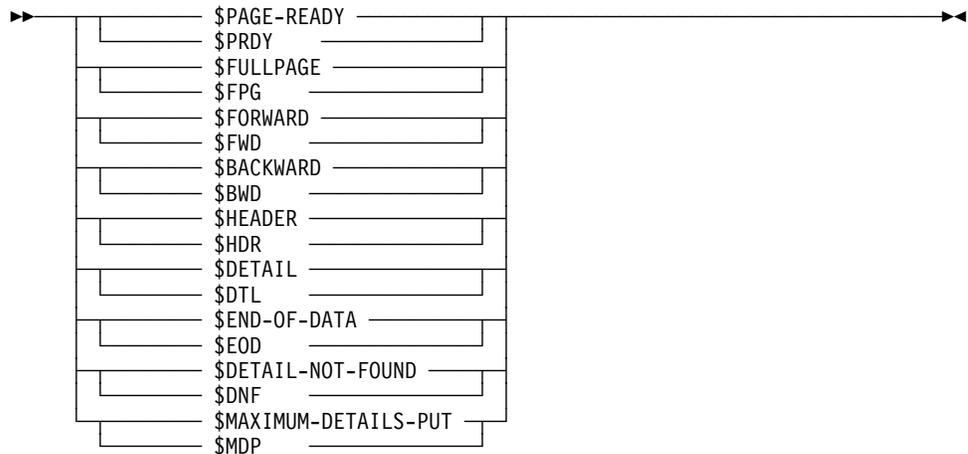
The map field status condition in the following IF statement is true when the user erases all data in the CUST-NAME, CUST-ADDR1, and CUST-CITY fields:

```
IF ALL OF (CUST-NAME, CUST-ADDR1, CUST-CITY) ARE ERASED ...
```

8.11 Map paging status conditions

Purpose: Determines the occurrence of runtime events associated with a pageable map.

Syntax:



Parameters

\$PAGE-READY

Tests whether the runtime system has written a full map page to scratch.

\$PAGE-READY is set to true for each map page built in a given map paging session before the page is displayed.

\$PAGE-READY is reset and the outcome of the test is false as soon as the next detail occurrence is written to a scratch record.

If \$PAGE-READY is used, the **Auto display** option must be chosen for the dialog. This setting is made using the Map Specification screen.

►► For information about this screen, see Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

\$PRDY can be used in place of \$PAGE-READY.

\$FULLPAGE

Tests whether the runtime system has displayed the first map page to the user as a result of a PUT DETAIL command.

\$FULLPAGE is reset and the outcome of the test is false when a DISPLAY command without the CONTINUE keyword is issued.

\$FPG can be used in place of \$FULLPAGE.

\$FORWARD

Tests whether the user has pressed the control key associated with paging forward.

`$FORWARD` is reset and the outcome of the test is false when a new page is displayed at the user's screen.

`$FWD` can be used in place of `$FORWARD`.

\$BACKWARD

Tests whether the terminal operator has pressed the control key associated with paging backward.

`$BACKWARD` is reset and the outcome of the test is false when a new page is displayed at the user's screen.

`$BWD` can be used in place of `$BACKWARD`.

\$HEADER

Tests whether a modified data tag (MDT) was set for any header or footer area map fields following the most recent mapin operation from the terminal.

`$HEADER` is reset and the outcome of the test is false when a `DISPLAY` command without the `CONTINUE` keyword is issued.

`$HDR` can be used in place of `$HEADER`.

\$DETAIL

Tests whether the most recent `GET DETAIL` command with the `FIRST` or `NEXT` keyword has retrieved a modified detail occurrence.

`$DETAIL` is reset and the outcome of the test is false when a `DISPLAY` command without the `CONTINUE` keyword is issued.

`$DTL` can be used in place of `$DETAIL`.

\$END-OF-DATA

Tests whether the most recent `GET DETAIL` command with the `FIRST` or `NEXT` keyword has encountered an end-of-data condition while attempting to retrieve a modified detail occurrence.

An end-of-data condition results when the runtime system reaches the physical end of detail occurrences without finding a modified detail occurrence.

`$END-OF-DATA` is reset and the outcome of the test is false when a `DISPLAY` command without the `CONTINUE` keyword is issued.

`$EOD` can be used in place of `$END-OF-DATA`.

\$DETAIL-NOT-FOUND

Tests whether the most recent `GET DETAIL` command with the `KEY IS` specification has encountered a detail-not-found condition while attempting to retrieve a modified detail occurrence.

A detail-not-found condition results if no detail occurrence with the specified key exists or if the existing detail occurrence is not a modified detail occurrence.

`$DETAIL-NOT-FOUND` is reset and the outcome of the test is false when a `DISPLAY` command without the `CONTINUE` keyword is issued.

\$DNF can be used in place of \$DETAIL-NOT-FOUND.

\$MAXIMUM-DETAILS-PUT

Tests whether storage is unavailable to hold new detail occurrences.

\$MAXIMUM-DETAILS-PUT (MDP) is set when a PUT DETAIL command fails to create a detail occurrence due to lack of storage.

The runtime system allocates storage for detail occurrences based on the system generation OLM statement.

\$MAXIMUM-DETAILS-PUT is reset across a pseudoconverse even though the condition still exists.

You can use \$MDP in place of \$MAXIMUM-DETAILS-PUT.

Usage: Map paging status conditions can be used in one or more dialogs associated with the same pageable map.

At the beginning of a map paging session, the map paging status conditions are initialized and the outcome of each test is false.

►► For information about map paging sessions and the syntax for the map paging commands mentioned below, see Chapter 17, “Map Commands.”

Example: The example below defines a premap process that builds the first page of a pageable map. The page is displayed with a message as soon as the page is built. The \$PAGE-READY condition is used to determine when the page is built:

```
OBTAIN FIRST EMPLOYEE WITHIN DEPT-EMPLOYEE.  
WHILE NOT $PAGE-READY  
  AND NOT DB-END-OF-SET  
  REPEAT.  
    MOVE EMP-ID TO WK-EMP-ID.  
    MOVE EMP-LNAME TO WK-EMP-LNAME.  
    MOVE EMP-START-DATE TO WK-EMP-START-DATE.  
    ACCEPT DB-KEY INTO WK-KEY FROM CURRENCY.  
    PUT NEW DETAIL KEY WK-KEY.  
    OBTAIN NEXT EMPLOYEE WITHIN DEPT-EMPLOYEE.  
  END.  
DISPLAY MSG TEXT  
  'FOR MORE INFORMATION, ENTER AN EMPLOYEE'S ID'.
```

Subsequent pages for this pageable map are built, as needed, by the map's response process (not shown).

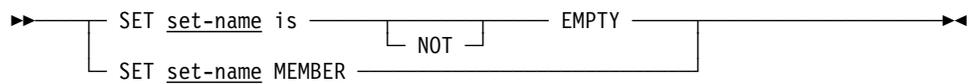
8.12 Set status condition

Purpose: Tests a set for the presence of member record occurrences or determine whether a record is a member of a specified set.

Note: The set status condition is not allowed for sets whose members are stored in native VSAM data sets.

For a discussion of native VSAM data sets in the CA-ADS environment, see Chapter 15, "Control Commands."

Syntax:



Parameters

set-name is EMPTY

Tests the current occurrence of the named set for the presence of member records. The outcome of the test is true only when the specified set has no members.

Set-name must be known to the dialog's subschema.

NOT

Specifies that the set has one or more members for the test to be true.

set-name MEMBER

Tests the current record of run unit to determine whether it participates as a member in any occurrence of the named set.

Set-name must be known to the dialog's subschema.

Examples: Example 1: Testing for set member records

The following statements establish a current occurrence of the CUSTOMER-ORDER set and then test to determine whether the set has any member records:

```

FIND CALC CUSTOMER.
IF SET CUSTOMER-ORDER EMPTY
THEN
  .
  .
  .
  
```

Example 2: Testing for a specific member of a set

The following statements establish a POLICY record as current of run unit and then test to determine whether the record is a member of any occurrence of the AGENCY-POLICY set:

```
OBTAIN CALC POLICY.  
IF SET AGENCY-POLICY MEMBER  
THEN  
  .  
  .  
  .
```

8.13 Arithmetic and assignment command status condition

Purpose: Tests the results of the previous assignment command.

Syntax

▶▶	ANY-DATA-ERROR - .	▶▶
	BAD-DATA-TYPE	
	UNSUPPORTED-DATA-CONVERSION	
	NO-NUMBER-EBCDIC/NUMERIC-CONVERSION	
	INCORRECT-FIELD-LENGTH	
	INVALID-SUBSCRIPT-VALUE	
	DATE-FORMAT-ERROR	
	SPECIFICATION-EXCEPTION	
	DATA-EXCEPTION	
	FIXED-POINT-OVERFLOW-EXCEPTION	
	FIXED-POINT-DIVIDE-EXCEPTION	
	DECIMAL-OVERFLOW-EXCEPTION	
	DECIMAL-DIVIDE-EXCEPTION	
	FLOATING-POINT-DIVIDE-EXCEPTION	
	EXPONENT-OVERFLOW-EXCEPTION	
	EXPONENT-UNDERFLOW-EXCEPTION	
	SIGNIFICANCE-EXCEPTION	

Example: The following example shows how the ALLOWING clause can be used to prevent application abends. The specified MOVE command moves a numeric field from an eight-byte field to a four-byte field. The application must be prepared to handle any error condition that might arise.

```
MOVE big-num TO little-num ALLOWING-ANY-DATA-ERROR.
IF DECIMAL-OVERFLOW-EXCEPTION
  DISPLAY ERROR MESSAGE TEXT 'SOURCE DATA TOO LARGE'.
IF ANY-DATA-ERROR
  DISPLAY ERROR MESSAGE TEXT 'INVALID DATA VALUE'.
```


Chapter 9. Constants

- 9.1 Overview 9-3
- 9.2 Figurative constants 9-4
- 9.3 Graphic literals 9-6
- 9.4 Multibit binary constants 9-7
- 9.5 Nonnumeric literals 9-8
- 9.6 Numeric literals 9-9

9.1 Overview

Constants are data items that are not subject to change during the execution of a dialog. Constants include the following:

- Figurative constants
- Graphic literals
- Multibit binary constants
- Nonnumeric literals
- Numeric literals

Usage: The VALUE IS clause for fields in records used by a CA-ADS dialog can be defined using any of the figurative constants listed in the syntax diagram. Note that the only allowable figurative constants in the VALUE IS clause for fields defined as numeric constants are ZEROS and ZEROES.

Restriction HIGH-VALUES, LOW-VALUES, and QUOTES cannot be used as a source field in a MOVE statement, or as an operand in a comparison expression if the corresponding target field is a data type other than group, EBCDIC, or UNSIGNED ZONE DECIMAL.

Examples: Example 1: Moving zero to a numeric field

```
MOVE ZERO TO COUNTER.
```

Example 2: Filling a field with binary zeros

```
MOVE ALL 'XO' TO HUGS-AND-KISSES.
```

Example 3: Comparing the contents of a field

```
IF EMP-NAME EQ SPACES
  THEN
    DISPLAY TEXT 'ENTER EMPLOYEE NAME'.
```

9.3 Graphic literals

Purpose: A graphic literal, also known as a G-literal, is a special type of double-byte character set (DBCS) string used when working with non-EBCDIC alphabets, such as the Japanese Kanji alphabet, the Korean Han-gul alphabet, or Chinese characters.

Usage: The graphic literal allows DBCS characters to be moved or compared to database or map record elements when shift codes are not part of the actual data.

This type of constant starts with the EBCDIC character G, followed by a single quotation character, a shiftout [SO], one or more DBCS characters, a shiftin [SI], and a closing single quotation character:

```
G' [SO]DBCS-characters[SI] '
```

The number of characters expressed depends on the hardware supporting DBCS. Maximum size is 255 bytes.

►► For information about defining data to handle DBCS characters in the data dictionary, refer to *IDD DDDL Reference*.

For information about defining maps that handle DBCS characters, see *CA-IDMS Mapping Facility*.

Example: An example of a graphic literal used in a process command is shown below:

```
IF MAP-REC-DBCS EQ G' [SO]DBCS-characters[SI] '  
  THEN  
    RETURN.
```

9.4 Multibit binary constants

Purpose: A multibit binary constant is a 1- to 32-character string that can contain only the values 1 and 0. The string is enclosed in single quotation marks with the first quotation mark immediately preceded by the character B.

```
B'110101'
```

Usage: A multibit binary constant can be used as a comparison for a data field and can be used to store a value in a data field.

The data field must be an elementary data field defined with the USAGE IS BIT clause. If the data field is an occurring field within a group, all other data fields in the group must be defined with USAGE IS BIT.

In general, groups in record structures are of type EBCDIC. Multibit binary is an exception. Even at the group level, multibit binary (MBB) fields should be reference for MOVEs or comparisons with B'...' fields. Specifically, a MBB group field would be initialized to all zeroes by moving B'000...' to the MBB field, or by redefining the MBB field with an EBCDIC field and moving LOW-VALUES to the redefined EBCDIC field.

Examples: Example 1: Data field definition

```
02 MASK-VALUE PIC X(7) USAGE IS BIT.
02 MASK-VALUE-OCCURRENCE REDEFINES MASK-VALUE
                           PIC X USAGE IS BIT OCCURS 7 TIMES.
```

Example 2: Process command

```
MOVE B'1001000' TO MASK-VALUE
WHILE MASK-VALUE EQ B'1001000'
  REPEAT.
  .
  .
  .
  IF DB-END-OF-SET
  THEN
    MOVE B'0' TO MASK-VALUE-OCCURRENCE (4).
END.
```

9.5 Nonnumeric literals

Purpose: A nonnumeric literal is a string of any allowable EBCDIC or DBCS characters. The nonnumeric literal must be enclosed in single quotation marks.

Nonnumeric literals can be used whenever the process command syntax specifies that the literal be in quotes.

Usage: A single quotation mark in the string is coded as two single quotation marks (' ').

Examples: Example 1: Digits and characters used as nonnumeric literals

The digits 0307 and the characters END OF SET CONDITION in the following example are nonnumeric literals:

```
IF ERROR-STATUS EQ '0307'  
  THEN  
    DISPLAY TEXT 'END OF SET CONDITION'.
```

Example 2: Single quotation marks within a string

The apostrophe contained in the following nonnumeric literal is coded with two single quotation marks:

```
MOVE 'JOHN KERR''S BROTHER' TO EMP-RELATIONSHIP.
```

9.6 Numeric literals

Purpose: A numeric literal is a numeric value that can be expressed as a fixed-point or floating-point constant.

First Usage:

Fixed-point numeric literals: A fixed-point numeric literal is a 1- to 16-digit number with an optional decimal point. The decimal point cannot be in the first or last position of the constant. If the constant does not contain a decimal point, it is an integer.

Fixed-point numeric literals are treated internally as packed decimal numbers and can be used whenever the process command syntax specifies a user-supplied numeric literal.

A fixed-point numeric literal can be signed or unsigned. A unary plus (+) or unary minus (-) can immediately precede the first digit or can be separated from the digit by one or more spaces. The numeric literal is positive if no sign is provided.

Examples: Example 1: Fixed-point numeric literal as a value for comparison

The following example compares the value in the field VALUE-2 to the fixed-point numeric literal -13.65:

```
IF VALUE-2 EQ -13.65
  THEN
  .
  .
  .
```

Example 2: An integer as a fixed-point numeric literal

The following example moves the integer 31456 to the field VALUE-1:

```
MOVE 31456 TO VALUE-1.
```

Second Usage:

Floating-point numeric literals: A floating-point numeric literal is a numeric literal whose value is expressed as a mantissa, which represents the number, followed by an exponent (characteristic), which determines the actual decimal position of the number.

All floating-point numeric literals are treated internally as internal short or long floating-point numbers, depending on the size of the mantissa. Floating-point numeric literals can be used whenever the process command syntax specifies an arithmetic expression, the name of a user-defined data field, or a user-supplied numeric constant.

Format of a floating-point numeric literal:

- The **mantissa**, coded first, is a 1- to 16-digit number with an optional decimal point. The decimal point can be placed anywhere in the number, including in the first or last position. If no decimal point is included, it is considered to be in the last position. For example:

1.2564E3

In this example, 1.2564 is the mantissa.

- The **character E** immediately follows the mantissa. For example:

1.2564E3

- The **characteristic**, a 1- or 2-digit integer preceded by an optional plus (+) or minus (-) sign immediately follows the character E. If no sign is included, it is assumed to be a plus sign. For example:

1.2564E3

In this example, 3 is the characteristic.

The value of the floating-point constant is the product of the mantissa, and ten raised to the power of the characteristic.

A floating-point numeric literal can be signed or unsigned. A unary plus (+) or unary minus (-) can immediately precede the first digit or can be separated from the digit by one or more spaces. If no sign is provided, the numeric literal is positive.

Examples: The following examples show the floating-point numeric literals and their fixed point equivalents.

Floating-point numeric literal	Fixed-point equivalent
1.2574E3	1257.4
1.2574E-3	0.0012574
-1.2574E20	-12574000000000000000

Chapter 10. Error Handling

- 10.1 Overview 10-3
- 10.2 The autostatus facility 10-4
- 10.3 Error expressions 10-6
- 10.4 The ALLOWING clause 10-7
- 10.5 Status definition records 10-9

10.1 Overview

Errors encountered while accessing the database, or involving queue or scratch activity are handled differently depending on whether or not SQL commands are used.

SQL commands: When CA-IDMS/DB executes an SQL statement, it returns information about the status of statement execution to a data structure called the SQLCA. The dialog contains logic to handle exceptional conditions resulting from statement execution. This logic takes the form of checking SQLCA information through the use of a conditional statement or through the use of the `WHENEVER SQLERROR` or `WHENEVER SQLWARNING` statement. In either situation, control is always returned to the dialog.

►► For more information on conditional statements and the `WHENEVER SQLERROR` statement processed during an SQL session, refer to *CA-IDMS SQL Programming*.

Non-SQL commands: When CA-IDMS/DB executes a non-SQL process command that involves database, queue, or scratch activity, or a `WRITE PRINTER` utility command, CA-ADS returns a 4-byte status code to an internal error-status field for the issuing dialog. A subsequent process command statement can test for the presence of a specified status code. Based on the outcome of the test, further processing can be done.

Handling errors in the non-SQL environment involves the use of the following:

- The **autostatus facility**, which handles errors generated by command processing
- **Error expressions**, which specify allowable status codes that can be returned
- The **status definition record**, which allows level-88 condition names to be associated with status codes

The autostatus facility, error expressions, and the status definition record are discussed separately in this section.

►► For information about using automatic editing and error-handling facilities to evaluate input data, see *CA-IDMS Mapping Facility*.

10.2 The autostatus facility

Autostatus is a runtime facility that enables CA-ADS to return specific status codes to an issuing dialog. When autostatus is in use, CA-ADS returns only certain status codes to the issuing dialog. The autostatus facility is not appropriate for use when data is defined in logical records and accessed using logical record commands.

►► For information about database access strategies, see Chapter 16, “Database Access Commands.”

Status codes returned by the autostatus facility: If command processing results in a status code not allowed by autostatus, dialog execution terminates abnormally. To allow the dialog to receive other status codes, specify all allowable status codes in an error expression. Error expressions are described later in this section.

Status codes allowed by autostatus are listed below.

Status code	Meaning
0000	The request was executed successfully.
0307	An end-of-set condition was encountered.
0326	The requested record cannot be found.
1707	An end-of-index condition was encountered.
1726	The requested index record cannot be found.
4303	The requested scratch area cannot be found.
4305	The requested scratch record cannot be found.
4317	A request to replace a scratch record was executed successfully.
4404	The requested queue id cannot be found.
4405	The requested queue record cannot be found.
5149	NOWAIT was specified in a KEEP LONGTERM request, and a wait is required.

Enabling autostatus: Autostatus is enabled on a dialog-by-dialog basis during dialog compilation by specifying the **Autostatus** option on the Options and Directives screen.

►► For more information about the Options and Directives screen, see Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

The availability of autostatus is controlled by the autostatus clause of the system generation ADSO statement. The ADSO AUTOSTATUS clause specifies:

- Whether the **Autostatus** option is selected, by default, on the Options and Directives screen
 - Whether this default setting can be overridden during dialog compilation
- For information about the AUTOSTATUS clause of the system generation ADSO statement, refer to *CA-IDMS System Generation*.

10.3 Error expressions

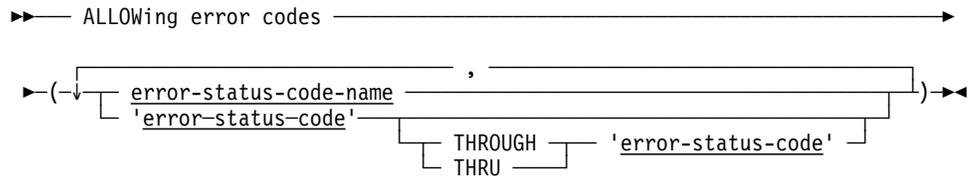
An error expression is a clause that consists of one or more allowable status codes, a range of status codes, or one or more level-88 status definition record condition names that can be returned to a dialog. In command syntax, error expressions are indicated as error-expression.

An error expression is allowed only if a dialog is compiled with the **Autostatus** option selected. If a dialog is compiled without the **Autostatus** option, the expression is flagged as an error during process compilation.

10.4 The ALLOWING clause

Purpose: The ALLOWING clause is used to allow the dialog to receive status codes not allowed by the autostatus facility.

Syntax:



Parameters

error-status-code-name

Specifies a level-88 condition name defined in the dialog's status definition record.

Multiple status code and condition name specifications must be separated by commas or blanks.

'error-status-code'

A 4-digit number enclosed in single quotation marks that identifies a status code applicable to the process command.

THROUGH 'error-status-code'

Specifies a status code or range of status codes.

THRU can be used in place of THROUGH.

Usage: An error expression is coded in the form of an ALLOWING clause in any of the following commands:

- All database record commands except for ACCEPT STATISTICS, COMMIT, READY, and ROLLBACK
- All logical record commands except for ON
- All queue and scratch management commands
- The WRITE PRINTER utility command

An ALLOWING clause overrides the autostatus facility. The values normally allowed by autostatus, with the exception of 0000, are returned only if explicitly named.

Nonzero status codes returned to a dialog are checked against the specified values. If the status code matches any of the specified values, processing continues. If the status code does not match any of the specified values, the CA-ADS runtime system terminates the application thread.

The ALLOWING clause is useful to check for deadlock conditions.

Examples: The examples below illustrate the ALLOWING clause in two database access commands.

Example 1: Specification of a range of allowable error codes

```
MODIFY ORDOR ALLOWING ERROR CODES ('0801' THRU '0850').
```

Example 2: Specification of a site-defined level-88 status code

```
FIND CUST-NUM ALLOWING (ANY-ERROR).
```

ANY-ERROR is a level-88 condition name in a site-defined status definition record. See the discussion of status definition records that follows this example.

10.5 Status definition records

Overview: Status codes can be tested using a system-supplied status definition record or by using a site-defined definition record. A status definition record associates status codes with level-88 condition names. The condition names can be coded in error expressions in place of 4-character status codes.

The status definition record is specified by the STATUS clause of the system generation ADSO statement. The STATUS clause specifies:

- The name of the default status definition record available to dialogs at dialog compilation time
- Whether this default status definition record can be overridden during dialog compilation

►► For further information on the system generation ADSO statement, refer to *CA-IDMS System Generation*.

A status definition record is associated with a dialog during dialog compilation. However, a buffer for this record is not allocated at runtime.

►► For further information on associating a status definition record with a dialog, see "Options and Directives screen" in Chapter 3, "CA-ADS Dialog Compiler (ADSC)."

System-supplied status definition record: CA-ADS supplies the ADSO-STAT-DEF-REC status definition record. ADSO-STAT-DEF-REC defines level-88 record elements for the status codes most commonly tested.

Record definition

```
ADSO-STAT-DEF-REC.
02  ERROR-STATUS          PIC 9(4).
    88  DB-STATUS-OK      VALUE '0000'.
    88  DB-END-OF-SET     VALUE '0307'.
    88  DB-REC-NOT-FOUND  VALUE '0326'.
    88  DB-END-OF-INDEX   VALUE '1707'.
    88  DB-INDEX-NOT-FOUND VALUE '1726'.
    88  SCRATCH-AREA-NOT-FOUND VALUE '4303'.
    88  SCRATCH-REC-NOT-FOUND VALUE '4305'.
    88  SCRATCH-REC-REPLACED VALUE '4317'.
    88  QUEUE-ID-NOT-FOUND VALUE '4404'.
    88  QUEUE-REC-NOT-FOUND VALUE '4405'.
    88  DB-ANY-ERROR     VALUE '0001'.
                                THRU '9999'.
```

Tests that specify error-status code names can include only those condition names that are defined in the status definition record associated with the dialog.

Examples: Example 1: Testing with the 4-byte status code

The following example tests for an error using the 4-byte status code 0307:

```
IF ERROR-STATUS IS '0307'  
THEN  
    CALL SUBA.  
ELSE  
    CALL SUBZ.
```

Example 2: Testing with a status definition record

The following example uses a status definition record level-88 element to test for the same error as in example 1 above:

```
IF DB-END-OF-SET  
THEN  
    CALL SUBA.  
ELSE  
    CALL SUBZ.  
  
OBTAIN CALC DEPARTMENT.  
OBTAIN CALC OFFICE.  
IF DB-REC-NOT-FOUND FOR DEPARTMENT  
    THEN CALL SUBA.  
IF DB-REC-NOT-FOUND FOR OFFICE  
    THEN CALL SUBB.
```

Site-defined status definition record: The system-defined status definition record ADSO-STAT-DEF-REC can be modified or replaced with one or more site-specific status definition records by using the IDD DDDL compiler.

Considerations

- The record definition must include a 1- to 32-character level-01 record name and one or more level-88 condition names that refer to status codes returned by database record, logical record, or queue and scratch management commands, or by the WRITE PRINTER utility command. Tests that specify error-status code names can include only those condition names that are defined in the status definition record.
- The record definition can include no more than one level-02 elementary field description that represents the value of the most recent status code returned to the dialog. Such a field is not referenced directly. CA-ADS uses the internal 4-byte unsigned zoned decimal error-status field that contains the most recently returned status code.

Examples: Example 1: Defining a site-specific status definition record

In this example, the first record defines the field CODE-FIELD, which contains two level-88 condition names. The second record contains only level-88 record elements. Record definitions are shown below:

```

01 ADSO-ONE-STAT-REC
   02 CODE-FIELD          PIC X(4).
      88 OKAY              VALUE '0000'.
      88 NOT-SO-GOOD      VALUE '0001' THRU '9999'.

01 ADSO-TWO-STAT-DEF
   88 DB-STATUS-OKAY      VALUE '0000'.
   88 DB-END-OF-SET       VALUE '0307'.
   88 NO-RECORD           VALUE '0326'.
   88 MODIFY-PROBLEM      VALUE '0800' THRU '0899'.

```

Example 2: Testing for the return of specific error codes

In this example, ADSO-TWO-STAT-DEF (defined in example 1) is used to test for the return of error-status codes 0800 through 0899:

```

MODIFY CUST
IF MODIFY-PROBLEM
THEN
.
.
.

```

Example 3: Testing for subschema record error status

In this example, ADSO-ONE-STAT-REC (defined in example 1) is used to test the latest error status returned for subschema records DEPARTMENT and OFFICE:

```

OBTAIN CALC DEPARTMENT.
OBTAIN CALC OFFICE.
IF NOT-SO-GOOD FOR DEPARTMENT
  THEN CALL SUBA.
IF NOT-SO-GOOD FOR OFFICE
  THEN CALL SUBB.

```

►► For instructions on using the RECORD statement to add, modify, or delete status definition records, refer to *IDD DDDL Reference*.

Chapter 11. Variable Data Fields

- 11.1 Overview 11-3
- 11.2 User-defined data field names 11-4
- 11.3 System-supplied data field names 11-6
- 11.4 Entity names 11-12

11.1 Overview

Variable data fields are data items whose values can change during the execution of a dialog.

Types of variable data fields: Variable data fields can be user-defined or system-defined. Each of these types of variable data fields is discussed separately below.

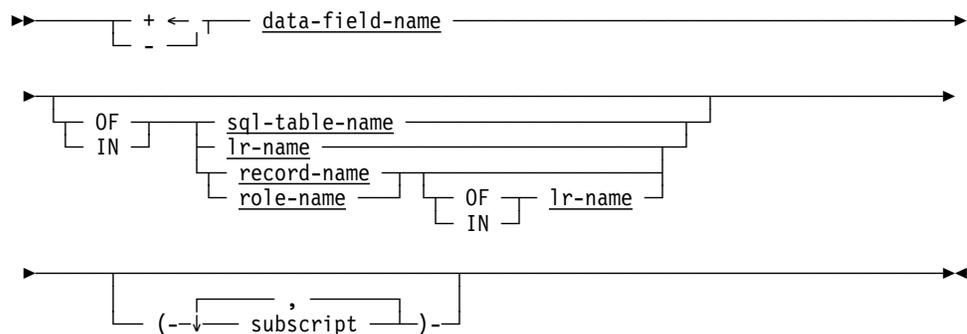
Syntax references: The appearance of *variable* in CA-ADS process language syntax denotes the validity of either a user-defined or a system-defined data field.

11.2 User-defined data field names

Purpose: User-defined data field names specify variable data fields in subschema records, map work records, or dialog work records.

User-defined data fields can be used as both source and target fields in process commands.

Syntax:



Parameters

+/-

Specifies the unary operator to precede a numeric data field.

A plus sign (+) does not change the sign of the data field.

A minus sign (-) multiplies the data field by -1.

+ is the default when neither + or - is specified.

A unary operator can be used when the data field is specified as part of an arithmetic expression.

data-field-name

Specifies the 1- to 32-character name of a variable data field.

Data-field-name must begin with an alphabetic, national (@, #, and \$), or numeric character. This field can specify a record or role name where logically appropriate. The named record or role is treated as a group field.

OF

Introduces *record-name*, *role-name*, or *lr-name*.

IN can be used in place of OF.

sql-table-name

Specifies the name of the SQL table that contains the fields referenced by *data-field-name*, when the SQL schema name has been entered in the ADSC Records and Tables screen.

►► For information about the Records and Tables screen, see 3.3.5, “Records and Tables screen.”

record-name

Specifies the name of the record that contains the fields referenced by *data-field-name*.

role-name

Specifies the name of the role that contains the fields referenced by *data-field-name*.

lr-name

Specifies the name of the logical record that contains the fields referenced by *data-field-name*.

Record-name, *role-name*, or *lr-name* is required if the named field is not unique among the records and roles known to the dialog. The reference to the data field must be unambiguous. For example, if the named field participates in a role, then reference to the field always requires qualification by record or role name. Further qualification of *record-name* or *role-name* with *lr-name* may also be necessary.

subscript

Specifies an arithmetic expression, variable data field, or numeric literal that indicates the value of each subscript required to reference a specific occurrence of the field that is referenced by *data-field-name*.

Subscript applies only if the named field is defined as a multiply-occurring field.

Example: The following example illustrates a data field name used with a MOVE command to specify a nonunique subscripted field:

```
MOVE CUSTOMER-NUMBER OF CUST-ACC-REC (3) TO CUSTORDR.
```


DB-NAME/NODE-NAME

Establish the database name and Distributed Database System (DDS) node name used for database commands at runtime. DB-NAME and NODE-NAME allow access to multiple databases under a DC/UCF system. When used, DB-NAME and NODE-NAME must be set before the first database command is issued for the run unit.

When dialog execution begins, DB-NAME and NODE-NAME are initialized to spaces. Database names and DDS node names that are moved to these fields within a process are propagated downward to all lower level dialogs. In this way, a dialog can access a database other than the subschema default.

DB-NAME and NODE-NAME can also be specified when the runtime system is initiated,

▶▶ For more information, see 4.1, “Initiating the CA-ADS runtime system.”

agr-data-field

Represents a data field provided in the ADSO-APPLICATION-GLOBAL-RECORD.

This record supplies runtime information in applications created by the CA-ADS application compiler (ADSA).

▶▶ For the names and definitions of these fields, see Appendix A, “System Records.”

amr-data-field

Represents a data field provided in the ADSO-APPLICATION-MENU-RECORD.

The ADSO-APPLICATION-MENU-RECORD contains information used to build menus in applications created by the CA-ADS application compiler.

▶▶ For the names and definitions of these fields, see Appendix A, “System Records.”

\$RESPONSE

References the 32-character \$RESPONSE field of a map.

Any value moved to \$RESPONSE appears in the map's \$RESPONSE field when the map is displayed. On mapin, the value in the \$RESPONSE field is considered by the runtime system in its selection of a dialog response process or an application function.

▶▶ For more information, see Chapter 4, “CA-ADS Runtime System.”

Once a response is selected, the \$RESPONSE field is cleared.

▶▶ For more information on the \$RESPONSE map field, refer to *CA-IDMS Mapping Facility*.

\$PAGE

References the \$PAGE field of a map.

\$PAGE determines the page displayed when a pageable map is mapped out to the terminal. Values are assigned to \$PAGE, as follows:

- At the beginning of a map paging session, \$PAGE is initialized to zero.
- Arithmetic and assignment process commands can modify \$PAGE.
- When a map is displayed, if \$PAGE is greater than the map's highest page number or less than its lowest page number, \$PAGE is set to the highest or lowest page number.

Note: The lowest page number can be greater than zero when backpaging is not allowed, as described in Chapter 17, "Map Commands."

- If the user presses a control key associated with paging forward or backward, \$PAGE is incremented or decremented by 1, unless it is already equal to the highest or lowest page number.
- If the user modifies the \$PAGE field displayed on the screen and presses a control key other than the paging forward key or paging backward key, and other than [Clear], [PA1], [PA2], or [PA3] (which do not transmit data), \$PAGE is assigned the modified value. If the new value is higher than the highest page number or lower than the lowest page number, \$PAGE is set to the highest or lowest page number.
- A GET DETAIL process command assigns \$PAGE the page number of the retrieved detail occurrence. If no detail occurrence is retrieved, \$PAGE is not changed.

►► For more information on the \$PAGE map field, refer to *CA-IDMS Mapping Facility*.

For more information on map paging sessions, see Chapter 17, "Map Commands."

LENGTH

Represents the halfword binary value equal to the number of characters entered into the named map field.

map-field-name

The name of a data field used by the dialog's map, enclosed in parentheses.

CURSOR-ROW

Represents the halfword binary value equal to the cursor row position on the dialog's map following the mapin operation.

CURSOR-COLUMN

Represents the halfword binary value equal to the cursor column position on the dialog's map following the mapin operation.

ERROR-STATUS

Represents the 4-byte EBCDIC value equal to the most recent status code returned to the dialog.

JULIAN

References a signed packed decimal field that contains the current date in the format *yyddd*.

JULIAN is updated before the execution of each premap and response process.

JULIANX

References a signed packed decimal field that contains the current date in the format *yyyyddd*.

JULIANX is updated before the execution of each premap and response process.

DATE

References an unsigned zoned decimal field that contains the current date in the format *yymmdd*.

DATE is updated before the execution of each premap and response process.

DATEX

References an unsigned zoned decimal field that contains the current date in the format *yyyymmdd*.

DATEX is updated before the execution of each premap and response process.

TIME

References an unsigned zoned decimal field that contains the time in the format *hhmmss*.

TIME is updated before the execution of each premap and response process.

\$ERROR-COUNT

(CA-ADS/Batch only) Contains the number of input records that have been written to the suspense file for the current dialog.

\$ERRCNT can be used in place of \$ERROR-COUNT.

Note: Data cannot be moved into \$ERROR-COUNT.

If a record is written to the suspense file but a suspense file was not allocated for the dialog, nothing is written, but \$ERROR-COUNT is still incremented.

\$INPUT-COUNT

(CA-ADS/Batch only) Contains the number of input records read for the current dialog.

\$INCNT can be used in place of \$INPUT-COUNT.

Note: Data cannot be moved into \$INPUT-COUNT.

\$OUTPUT-COUNT

(CA-ADS/Batch only) Contains the number of output records written for the current dialog.

\$OUTCNT can be used in place of \$OUTPUT-COUNT.

Note: Data cannot be moved into \$OUTPUT-COUNT.

Usage: System-supplied data fields are provided automatically for use by a dialog, except for fields in the ADSO-APPLICATION-GLOBAL-RECORD and the ADSO-APPLICATION-MENU-RECORD. To use these system record fields in a dialog, the records must be associated with the dialog.

►► For more information, see Appendix A, “System Records.”

All system-supplied data fields can be used as source fields in process commands. The following fields can also be used as target fields:

- DIRECT-DBKEY
- DB-NAME
- NODE-NAME
- Fields in ADSO-APPLICATION-GLOBAL-RECORD
- Fields in ADSO-APPLICATION-MENU-RECORD
- \$RESPONSE
- \$PAGE

System-supplied data fields for batch processing (\$ERROR-COUNT, \$INPUT-COUNT, \$OUTPUT-COUNT) count the suspense, input, and output file records read or written by each dialog. A field is set to zero when the file it describes is opened. If an input file is opened, closed, then reopened, the field is reset to zero when the file is reopened. Data from these fields can be moved to other data fields, but data cannot be moved into the system-supplied fields.

Examples: Example 1: Using the DB-NAME and NODE-NAME fields

This example uses the DB-NAME and NODE-NAME fields to establish IDMSNWKZ as an alternative database for a dialog. SYSTEM99 is named as the DDS node that controls IDMSNWKZ. All lower level dialogs access IDMSNWKZ and SYSTEM99 unless another database is established in a lower level dialog:

```
MOVE 'IDMSNWKZ' TO DB-NAME.  
MOVE 'SYSTEM99' TO NODE-NAME.
```

Note: The database and DDS node cannot be changed at a lower level if the processing is part of an extended run unit.

For more information, see Chapter 4, “CA-ADS Runtime System.”

Example 2: Displaying a value in the \$RESPONSE field

This example causes ADD to appear in the \$RESPONSE field of a displayed map:

```
MOVE 'ADD' TO $RESPONSE.  
DISPLAY.
```

Example 3: Using \$PAGE to display a specified map page

This example displays page 10 of a pageable map:

```
MOVE 10 TO $PAGE.  
DISPLAY.
```

11.4 Entity names

Purpose: Identifies the names of entities.

Usage: The names of entities, such as database records, logical records, sets, areas, queue ids, scratch ids, subroutines, dialogs, user programs, or message identifiers are user supplied.

Entity names are used whenever the command syntax specifies the type of entity followed by *-name* or *-id*, such as *record-name*, *set-name*, and *message-id*.

Specific entity names are described where they occur in the syntax for individual commands.

Chapter 12. Introduction to Process Commands

- 12.1 Overview 12-3
- 12.2 Summary of process commands 12-4
- 12.3 INCLUDE 12-8

12.1 Overview

CA-ADS process commands are COBOL-like statements used to construct processing routines for dialogs. These processing routines are stored in the dictionary as process modules. Process commands can perform activities such as:

- Calculate values and move data
- Define and call subroutines
- Access and update database values
- Modify maps and handle pageable maps
- Manage queue and scratch records

A summary of CA-ADS process commands is presented below. Details of commands in each command category are given in the remaining chapters of this volume and the next volume.

Information about the INCLUDE directive, which inserts one process module into another at compile time, is discussed in "Including common routines in process modules" later in this section.

Note: For information on how to interpret syntax diagrams, refer to "Understanding syntax diagrams" in the preface of this manual.

12.2 Summary of process commands

The table below summarizes the purpose of each process command. The commands are categorized according to the activities they perform.

Category	Keywords	Purpose
Arithmetic and assignment commands	ADD	Calculates the sum of two values
	COMPUTE	Evaluates an arithmetic expression
	DIVIDE	Calculates the quotient of two values
	MOVE	Moves a value to a target field
	MULTIPLY	Calculates the product of two values
	SUBTRACT	Calculates the difference between two values
Conditional commands	EXIT	Terminates a WHILE command
	IF	Performs conditional execution
	NEXT	Terminates an IF command
	WHILE	Iterates a loop based on a condition
Control commands	CONTINUE	Terminates a current process and executes a dialog's premap process
	DISPLAY	Displays a dialog's map or reexecutes a dialog's premap process
	EXECUTE NEXT FUNCTION	Directs the flow of control in an application defined by the application compiler
	INVOKE	Passes control to a lower level dialog
	LEAVE	Terminates a CA-ADS application
	LINK	Passes control to a lower level dialog or to a user program with inline return expected

Category	Keywords	Purpose
	READ TRANSACTION	Terminates a current process, performs a mapin operation, and selects the next function or response process to be executed
	RETURN	Returns control to a higher level dialog
	TRANSER	Passes control to a dialog at the same level
	WRITE TRANSACTION	Terminates a current process, performs a mapout operation, and passes control within an application (batch only)
Database commands	ACCEPT	Retrieves database keys page group information and database access statistics for navigationally accessed records
	BIND PROCEDURE	Establishes communication between a dialog and a DBA-written procedure
	COMMIT	Writes checkpoints to the journal file and releases record locks for navigationally accessed records
	CONNECT	Connects records in navigationally accessed sets
	DISCONNECT	Disconnects records from navigationally accessed sets
	ERASE	Erases database records
	FIND	Locates navigationally accessed records in the database
	GET	Retrieves navigationally accessed records from the database
	KEEP	Places locks on navigationally accessed records
	MODIFY	Modifies database records
	OBTAIN	Locates and retrieves database records

12.2 Summary of process commands

Category	Keywords	Purpose
	ON	Performs conditional execution based on the outcome of LRF command execution
	READY	Specifies an area usage mode for navigational database access
	RETURN DB-KEY	Retrieves index entries associated with navigationally accessed database records
	ROLLBACK	Requests recovery operations for navigationally accessed records
	STORE	Stores database records
Map commands	ATTRIBUTES	Alters map field attributes (an alternative format to MODIFY MAP)
	CLOSE	Closes a dialog's input or output file maps (batch only)
	GET DETAIL	Retrieves a modified detail occurrence of a pageable map
	MODIFY MAP	Alters the options specified for the dialog's map
	PUT DETAIL	Creates or modifies a detail occurrence of a pageable map
Queue and scratch management commands	DELETE QUEUE	Deletes queue records
	GET QUEUE	Retrieves queue records
	PUT QUEUE	Stores queue records
	DELETE SCRATCH	Deletes scratch records
	GET SCRATCH	Retrieves scratch records
	PUT SCRATCH	Stores scratch records
Subroutine commands	CALL	Passes control to a predefined subroutine
	DEFINE	Defines a subroutine
	GOBACK	Returns control from a subroutine

Category	Keywords	Purpose
Utility commands	ABORT	Causes the runtime system to abort the application
	ACCEPT	Retrieves runtime status information associated with the current dialog
	INITIALIZE RECORDS	Reinitializes a dialog's record buffers
	SNAP	Requests a snapshot dump of the areas in memory associated with CA-ADS
	WRITE PRINTER	Transmits data from a dialog to a CA-IDMS/DC or CA-IDMS/UCF queue for subsequent printing
	WRITE TO LOG/OPERATOR	Sends a message to the log file or to the operator's console (batch only)

SQL statements: In addition to the database commands listed above, CA-ADS also supports embedded SQL statements.

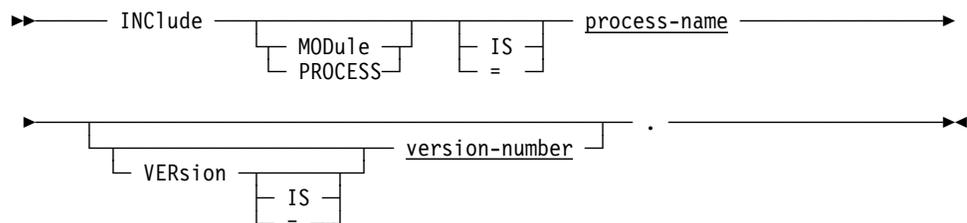
▶▶ See *CA-IDMS SQL Reference* for syntax for conditional statements (WHENEVER) and database commands.

▶▶ See *CA-IDMS SQL Programming* for more information on using SQL with CA-ADS.

12.3 INCLUDE

Purpose: Inserts stored process source code into another process at compile time.

Syntax



Parameters

MODu1e is process-name

Causes the source code of the named process module to be inserted logically in the current process source code at compile time. The process module itself is not changed. At runtime, CA-ADS executes the process as if the included code were coded in the process itself.

Process-name must name a module occurrence in the data dictionary. The module is defined with an IDD DDDL ADD PROCESS statement or an ADD MODULE statement with the attribute LANGUAGE IS PROCESS.

VERsion is version-number

Indicates the version number associated with the included process module. If not specified, *version-number* defaults to the default version number set in the dictionary.

Usage

Considerations

- The included source code must be stored in the data dictionary as a process module.
- INCLUDE commands can be nested.
- INCLUDE cannot be used recursively. An INCLUDE statement cannot reference a process that is already in the nested INCLUDE structure.
- A process cannot include itself. For example, if process A includes process B and process B includes process C, then process C cannot include process A, B, or C.
- The INCLUDE statement must be contained entirely on one process code line.
- Any other process commands entered on the same line as the INCLUDE statement must precede INCLUDE.
- An INCLUDE statement can be followed by comments on the same line.

Example: The example below illustrates the use of the INCLUDE command:

Process: CUST-NUM-CHECK

```
MOVE CUST-NUM TO A.  
INCLUDE MODULE VALUE-CHECK.  
RETURN.
```

Process: VALUE-CHECK

```
IF A = 1  
THEN  
    DISPLAY.  
ELSE  
    LINK TO 'LINKDIAL'.
```

If the application developer specifies CUST-NUM-CHECK as a premap or response process using the CA-ADS dialog compiler (ADSC), on the Process Modules screen, the following process source code will logically be compiled:

```
MOVE CUST-NUM TO A.  
IF A = 1  
THEN  
    DISPLAY.  
ELSE  
    LINK TO 'LINKDIAL'.  
RETURN.
```


Chapter 13. Arithmetic and Assignment Commands

- 13.1 Overview 13-3
- 13.2 General considerations 13-4
 - 13.2.1 Numeric fields 13-4
 - 13.2.2 EBCDIC and DBCS fields 13-4
 - 13.2.3 Arithmetic and assignment command status condition 13-5
- 13.3 Arithmetic commands 13-6
 - 13.3.1 ADD 13-6
 - 13.3.2 COMPUTE 13-7
 - 13.3.3 DIVIDE 13-8
 - 13.3.4 MULTIPLY 13-10
 - 13.3.5 SUBTRACT 13-11
- 13.4 Assignment command 13-13
 - 13.4.1 MOVE 13-14

13.1 Overview

CA-ADS arithmetic and assignment commands are used to perform calculations and move data. Values used in arithmetic or assignment commands can include built-in functions.

►► See Chapter 7, “Built-in Functions” for more information.

Arithmetic and assignment commands: The arithmetic and assignment commands are listed in the table below. Each command is presented in alphabetical order after the general considerations that follow the table.

Command	Purpose
ADD	Calculates the sum of two values and places the result in a variable data field
COMPUTE	Evaluates an arithmetic expression and places the result in a variable data field
DIVIDE	Calculates the quotient of two values, places the result in a variable data field, and optionally places the remainder in another variable data field
MOVE	Moves a value to a variable data field
MULTIPLY	Calculates the product of two values and places the result in a variable data field
SUBTRACT	Calculates the difference between two values and places the result in a variable data field

13.2 General considerations

General considerations are given below for arithmetic and assignment operations that involve source and target fields of different lengths. Considerations for numeric fields are presented first, followed by considerations for EBCDIC and DBCS fields.

13.2.1 Numeric fields

A value moved between numeric source and target fields is decimal-point aligned in the target field. Differences between the source-field value and the target field are handled as follows:

- **Differences to the left of the decimal point:**
 - If the portion of the source-field value to the left of the decimal point is **shorter** than the corresponding portion of the target field, the leftmost positions in the target field are filled with zeros.
 - If the portion of the source-field value to the left of the decimal point is **longer** than the corresponding portion of the target field, the operation cannot be executed and CA-ADS terminates the application thread abnormally.
- **Differences to the right of the decimal point:**
 - If the portion of the source-field value to the right of the decimal point is **shorter** than the corresponding portion of the target field, the rightmost positions in the target field are filled with zeros.
 - If the portion of the source-field value to the right of the decimal point is **longer** than the corresponding portion of the target field, the value is either rounded to or truncated at the rightmost decimal position in the target field, depending on whether the **ROUNDED** or **TRUNCATED** specification applies.

13.2.2 EBCDIC and DBCS fields

A nonnumeric value moved between EBCDIC fields or DBCS fields is left justified in the target field. The following considerations apply:

- If the source-field value is **shorter** than the target field, the remaining positions in the target field are filled with blanks.
 - If the source-field value is **longer** than the target field, the rightmost characters are truncated.
- For special information about the **MOVE** command and EBCDIC fields, see 13.4, “Assignment command” later in this section.

13.2.3 Arithmetic and assignment command status condition

ADS supports error handling for assignment and arithmetic commands. This allows an application to handle errors such as data exception or decimal overflow rather than forcing ADS to abort the dialog execution. An ALLOWING clause specifies which error condition a dialog is prepared to handle.

Assignment command status condition: The following condition names can be specified as assignment command status conditions in ALLOWING clauses:

- ANY-DATA-ERROR
- BAD-DATA-TYPE
- UNSUPPORTED-DATA-CONVERSION
- NO-NUMBER-EBCDIC/NUMERIC CONVERSION
- INCORRECT-FIELD-LENGTH
- INVALID-SUBSCRIPT-VALUE
- DATE-FORMAT-ERROR
- SPECIFICATION-EXCEPTION
- DATA-EXCEPTION
- FIXED-POINT-OVERFLOW-EXCEPTION
- FIXED-POINT-DIVIDE-EXCEPTION
- DECIMAL-OVERFLOW-EXCEPTION
- DECIMAL-DIVIDE-EXCEPTION
- FLOATING-POINT-DIVIDE-EXCEPTION
- EXPONENT-OVERFLOW-EXCEPTION
- EXPONENT-UNDERFLOW-EXCEPTION
- SIGNIFICANCE-EXCEPTION

Specifying ANY-DATA-ERROR allows a dialog to retain control following any error condition. The data after an exception condition is encountered will be returned as if the command had never been attempted. The meaning of the exception conditions are defined in the IBM *Principles of Operations Manual*.

Example: The following example shows how the ALLOWING clause can be used to prevent application abends. The specified MOVE command moves a numeric field from an eight-byte field to a four-byte field. The application must be prepared to handle any error condition that might arise.

```
MOVE big-num TO little-num ALLOWING ANY-DATA-ERROR.  
IF DECIMAL-OVERFLOW-EXCEPTION  
  DISPLAY ERROR MESSAGE TEXT 'SOURCE DATA TOO LARGE'.  
IF ANY-DATA-ERROR  
  DISPLAY ERROR MESSAGE TEXT 'INVALID DATA VALUE'.
```

13.3 Arithmetic commands

Arithmetic commands assign values to variable data fields based on the results of a simple addition, subtraction, multiplication, or division operation or a compound operation involving multiple arithmetic functions.

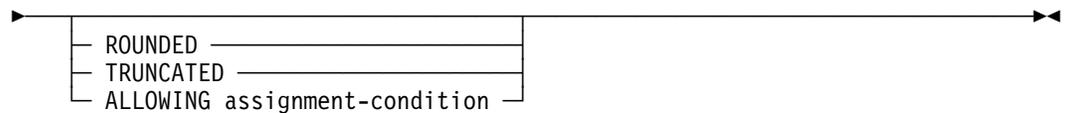
13.3.1 ADD

Purpose: Calculates the sum of two values.

Syntax

►► — ADD **arithmetic-expression** to variable — **options** — . —►►

Expansion of options



Parameters

arithmetic-expression

Specifies the value being added to the value in *variable*.

►► For information on arithmetic-expression, see Chapter 6, “Arithmetic Expressions.”

to variable

Specifies the field that contains the value to which arithmetic-expression is added. Following execution of the command, *variable* contains the result of the ADD operation.

ROUNDED

Rounds the result of the addition to the number of decimal positions found in *variable*.

TRUNCATED

Truncates the result of the addition to the number of decimal positions found in *variable*.

The default specification is **ROUNDED** if **COBOL moves are enabled** is not selected and **TRUNCATED** if the option is selected.

►► For more information, see 13.4, “Assignment command” later in this chapter.

ALLOWING

Specifies which error conditions that would normally abend should cause control to be returned to the dialog for error handling. The list of allowable assignment-condition names can be found in the section entitled “Arithmetic and Assignment Command Status Condition.”

Usage: The ADD command is used to perform addition. A variable data field value, a numeric literal, or the result of an arithmetic expression is added to a data field value. The result is placed in the data field that contains the right operand.

Example: The example below uses the ADD command to add the value 1 to the contents of the variable data field COUNTER.

ADD 1 TO COUNTER.

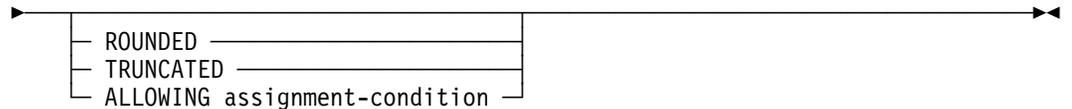
13.3.2 COMPUTE

Purpose: Evaluates an arithmetic expression. The result of the evaluation is placed in a variable data field.

Syntax:

►— COMPUTE variable — **options** — = — arithmetic-expression — . —►

Expansion of options



Parameters

variable

Specifies the name of a variable data field that contains the result of the COMPUTE operation.

ROUNDED

Rounds the result of the computation to the number of decimal positions found in *variable*.

TRUNCATED

Truncates the result of the computation to the number of decimal positions found in *variable*.

The default specification is **ROUNDED** if **COBOL moves are enabled** is not selected and **TRUNCATED** if the option is selected.

►► For more information, see 13.4, “Assignment command” later in this chapter.

ALLOWING

Specifies which error conditions that would normally abend should cause control to be returned to the dialog for error handling. The list of allowable assignment-condition names can be found in the section entitled “Arithmetic and Assignment Command Status Condition.”

arithmetic-expression

Specifies the arithmetic expression being evaluated for the value contained in *variable*.

►► For information on arithmetic-expression, see Chapter 6, “Arithmetic Expressions”

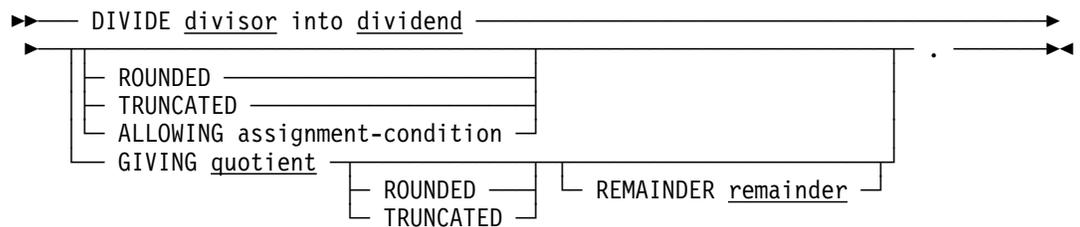
Example: The example below uses the COMPUTE command to calculate commission as a percentage of sales plus a percentage of sales above quota. The result is truncated.

```
COMPUTE COMMISSION TRUNCATED =
    0.10 * SALES + 0.03 * (SALES - QUOTA).
```

13.3.3 DIVIDE

Purpose: Calculates the quotient of two values.

Syntax:



Parameters

divisor

Specifies the divisor in the divide operation. *Divisor* cannot be longer than eight bytes. *Divisor* can be an arithmetic expression, a numeric literal, or a user-defined variable.

into dividend

Specifies the dividend in the divide operation. *Dividend* can be a user-defined variable.

ROUNDED

(Coded immediately after *dividend*) Rounds the result of the division to the number of decimal positions found in *dividend*.

TRUNCATED

(Coded immediately after *dividend*) Truncates the result of the division to the number of decimal positions found in *dividend*.

The default specification is **ROUNDED** if **COBOL moves are enabled** is not selected and **TRUNCATED** if the option is selected.

►► For more information, see 13.4, "Assignment command" later in this section.

ALLOWING

Specifies which error conditions that would normally abend should cause control to be returned to the dialog for error handling. The list of allowable assignment-condition names can be found in the section entitled "Arithmetic and Assignment Command Status Condition."

GIVING quotient

Specifies the user-defined variable that receives the quotient of the **DIVIDE** operation.

ROUNDED

(Coded after the GIVING parameter) Rounds the result of the division to the number of decimal positions found in *quotient*.

TRUNCATED

(Coded after the GIVING parameter) Truncates the result of the division to the number of decimal positions found in *quotient*.

The default specification is TRUNCATED if the REMAINDER parameter is specified.

If the REMAINDER parameter is not specified and if **COBOL moves are enabled** is not selected, the default specification is ROUNDED. If **COBOL moves are enabled** is selected, the default specification is TRUNCATED.

►► The effect of the **COBOL moves are enabled** option is described under 13.4, “Assignment command” later in this section.

REMAINDER remainder

Specifies the field that receives the remainder of a DIVIDE operation. The remainder is calculated by subtracting the product of the truncated quotient and the divisor from the dividend.

If *quotient* and *remainder* refer to the same data field, the data field at the end of the DIVIDE command contains the quotient. The remainder is ignored.

Usage:

Definition: The DIVIDE command is used to perform division. A variable data field value, a numeric literal, or the result of an arithmetic expression, which represents the divisor, is divided into a variable data field value, which represents the dividend.

The result of the division (the quotient) can be stored in the dividend data field or in a designated quotient data field. If the result is stored in a quotient data field, a data field to hold the remainder can also be specified.

Considerations

- If the GIVING parameter is not specified, *dividend* contains the result of the DIVIDE operation.

If *dividend* is to contain the result of the divide operation, ROUNDED or TRUNCATED can be specified immediately after *dividend*. The GIVING and REMAINDER parameters, however, cannot be specified.

If *dividend* is not to contain the result, ROUNDED or TRUNCATED cannot be specified immediately after *dividend*. The GIVING parameter must be specified. The GIVING parameter can be followed optionally by ROUNDED or TRUNCATED and the REMAINDER parameter.

- The truncated quotient contains as many positions to the right of the decimal point as does *quotient*. If the ROUNDED keyword is used, the quotient is rounded after the remainder is calculated.

Examples: The examples below illustrate the use of the DIVIDE command to divide the value in the TOT-SALES field by the value in the NUM-ORDERS field.

Example 1: Simple division

In this example, the quotient is placed in TOT-SALES:

```
DIVIDE NUM-ORDERS INTO TOT-SALES.
```

Example 2: Obtaining a truncated quotient with a remainder

In this example, the quotient is truncated and placed in TOT-SALES-Q. The remainder is placed in TOT-SALES-R:

```
DIVIDE NUM-ORDERS INTO TOT-SALES
      GIVING TOT-SALES-Q TRUNCATED REMAINDER TOT-SALES-R.
```

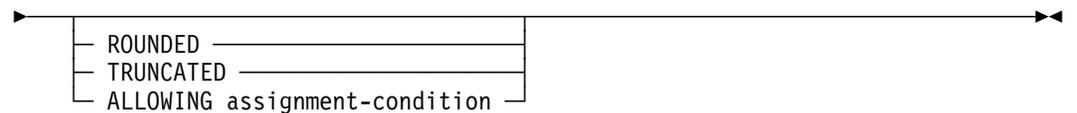
13.3.4 MULTIPLY

Purpose: Calculates the product of two variables.

Syntax:

```
►►—— MULTIPLY arithmetic-expression by variable — options — . —————►►
```

Expansion of options



Parameters

arithmetic-expression

Specifies the arithmetic expression being added to the value contained in *variable*.

►► For information on *arithmetic-expression*, see Chapter 6, “Arithmetic Expressions.”

by *variable*

Specifies the data field that contains the value by which *arithmetic-expression* is multiplied. Following execution of the command, *variable* contains the result of the MULTIPLY operation.

ROUNDED

Rounds the result of the multiplication to the number of decimal positions found in *variable*.

TRUNCATED

Truncates the result of the multiplication to the number of decimal positions found in *variable*.

The default specification is **ROUNDED** if **COBOL moves are enabled** is not selected and **TRUNCATED** if the option is selected,

►► For more information, see 13.4, “Assignment command” later in this section.

ALLOWING

Specifies which error conditions that would normally abend should cause control to be returned to the dialog for error handling. The list of allowable assignment-condition names can be found in the section entitled “Arithmetic and Assignment Command Status Condition.”

Usage:

Definition: The **MULTIPLY** command is used to perform multiplication. A variable data field value, a numeric literal, or the result of an arithmetic expression is multiplied by a variable data field value. The result is placed in the variable data field that contains the right operand.

Example: The example below illustrates the use of the **MULTIPLY** command to multiply the value in the **FICA-PCT** field by the value in the second occurrence of the **DEDUCT** field:

```
MULTIPLY FICA-PCT BY DEDUCT(2).
```

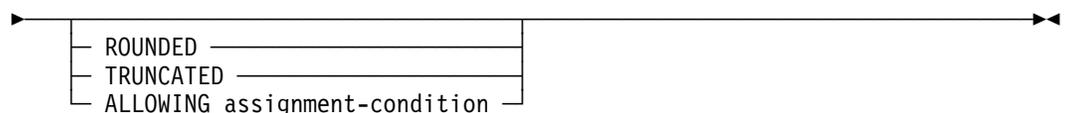
13.3.5 SUBTRACT

Purpose: Calculates the difference between two variables.

Syntax:

►► — **SUBTRACT** **arithmetic-expression** from variable — **options** — . —►►

Expansion of options



Parameters

arithmetic-expression

Specifies the arithmetic expression being subtracted from the value contained in *variable*.

►► For information on arithmetic-expression, see Chapter 6, “Arithmetic Expressions.”

from variable

Specifies the data field that contains the value from which arithmetic-expression is subtracted. Following execution of the command, *variable* contains the result of the **SUBTRACT** operation.

ROUNDED

Rounds the result of the multiplication to the number of decimal positions found in *variable*.

TRUNCATED

Truncates the result of the multiplication to the number of decimal positions found in *variable*.

The default specification is ROUNDED if **COBOL moves are enabled** is not selected and TRUNCATED if the option is selected.

►► For more information, see 13.4, “Assignment command” later in this section.

ALLOWING

Specifies which error conditions that would normally abend should cause control to be returned to the dialog for error handling. The list of allowable assignment-condition names can be found in the section entitled "Arithmetic and Assignment Command Status Condition."

Usage:

Definition: The SUBTRACT command is used to perform subtraction. A variable data field value, a numeric literal, or the result of an arithmetic expression is subtracted from a variable data field value. The result is placed in the variable data field that contains the right operand.

Example: The example below illustrates the use of the SUBTRACT command to subtract the value in the QTY-SHIPED field from the value in the BAL-ON-HAND field:

```
SUBTRACT QTY-SHIPED FROM BAL-ON-HAND.
```

13.4 Assignment command

MOVE command: The MOVE command is used to move a variable data field value, a numeric, nonnumeric, multi-bit binary, or figurative constant, or the result of an arithmetic expression into a variable data field.

Comparison of CA-ADS and COBOL rules for move operations: COBOL and CA-ADS differ slightly when moving the results of an arithmetic or assignment command into the target field of the command. The table below compares the COBOL and CA-ADS rules.

The application developer determines the set of rules to be used on a dialog-by-dialog basis. The default set of rules is specified in the ADSO statement issued during system generation. The default specification can be overridden for a dialog on the Options and Directives screen of the CA-ADS dialog compiler.

►► For more information, see as described in Chapter 3, “CA-ADS Dialog Compiler (ADSC).”

Operation	CA-ADS rules	COBOL rules
Move a numeric result to an EBCDIC target field	<ol style="list-style-type: none"> 1. Drop the decimal portion 2. Place a negative sign (if any) to the left of the result 3. Right justify the result in the target field 	<ol style="list-style-type: none"> 1. Retain the decimal portion without the decimal point 2. Drop any negative sign 3. Left justify the result in the target field
Round or truncate the value ¹	Round the value (By default the value is truncated for a DIVIDE command with the REMAINDER parameter)	Truncate the value

¹ Arithmetic and assignment commands allow the application developer to override the default rounding or truncating rule by means of the `ROUNDED/TRUNCATED` specification.

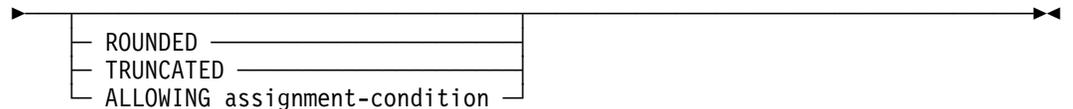
13.4.1 MOVE

Purpose: Moves a value to a target field.

Syntax:

►— MOVE value to variable — **options** — . —————►

Expansion of options



Parameters

value

Specifies the value being moved to *variable*. *Value* can contain an arithmetic expression, a numeric literal, a user-defined variable, or a literal enclosed in single quotation marks.

to variable

Specifies a variable data field that contains the result of the MOVE operation.

ROUNDED

Rounds the result of the move to the number of decimal positions found in *variable*.

TRUNCATED

Truncates the result of the move to the number of decimal positions found in *variable*.

The default specification is ROUNDED if the **COBOL moves are enabled** option on the Options and Directives screen has not been chosen and TRUNCATED if the option has been chosen.

ALLOWING

Specifies which error conditions that would normally abend should cause control to be returned to the dialog for error handling. The list of allowable assignment-condition names can be found in the section entitled "Arithmetic and Assignment Command Status Condition."

Usage

Consideration: ROUNDED/TRUNCATED is ignored if *value* is nonnumeric.

Example: The example below illustrates the use of the MOVE command to move the value from the ACCT-BAL field to the TOT-BAL field:

```
MOVE ACCT-BAL TO TOT-BAL.
```

Chapter 14. Conditional Commands

- 14.1 Overview 14-3
- 14.2 EXIT 14-4
- 14.3 IF 14-5
- 14.4 NEXT 14-8
- 14.5 WHILE 14-10

14.1 Overview

CA-ADS conditional commands are used to specify processing based on the outcome of a conditional test. The conditions to be tested are specified by coding conditional expressions.

►► For information, see Chapter 8, “Conditional Expressions.”

Summary of conditional commands: Conditional commands are listed below. Each command is presented in alphabetical order after the table.

Command	Purpose
EXIT	Terminates WHILE and ON ¹ command processing and passes control to the next command in the process
IF	Performs a conditional test and specifies actions to be taken based on the outcome of the test
NEXT	Terminates IF or ON ¹ command processing and passes control to the next command in the process
WHILE	Performs a conditional test and specifies actions to be taken as long as the outcome of the test is true

¹ See Chapter 16, “Database Access Commands.”

14.2 EXIT

Purpose: Exits a processing loop created by a WHILE or ON command regardless of the outcome of the command condition.

The WHILE command is described later in this chapter.

►► For information about the ON command, see Chapter 16, “Database Access Commands.”

Terminates WHILE and ON command processing.

Syntax

►► EXIT . 

Usage

Considerations

- EXIT can be used in conjunction with the REPEAT parameter in an ON command.
- Control passes to the next command outside the WHILE or ON structure following an EXIT command.
- EXIT is typically used following an IF statement that tests for a secondary condition.

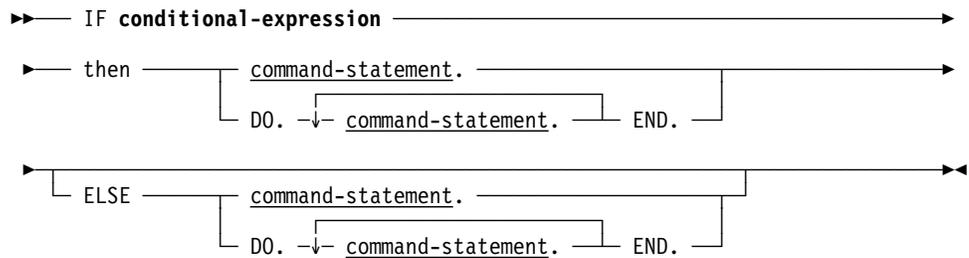
Example: The statements below illustrate the use of an EXIT command. The DISPLAY command is executed when A is greater than B or when Z becomes greater than 100, whichever occurs first:

```
WHILE A LE B
  REPEAT.
    ADD 1 TO Z.
    IF Z GT 100
      THEN EXIT.
    ADD 1 TO A.
  END.
DISPLAY.
```

14.3 IF

Purpose: Evaluates one or more conditional expressions and specifies actions based on the outcome of the evaluation.

Syntax:



Parameters

conditional-expression

Specifies the conditional expression to be evaluated. The outcome of the evaluation determines the processing that occurs.

Conditional-expression contains one or more conditions to be evaluated and is specified according to the rules presented in Chapter 8, “Conditional Expressions.”

then command-statement

Specifies the commands to be executed if the condition is true.

Multiple command statements must be preceded by DO and followed by END.

Command-statement can be any valid CA-ADS process command, including another conditional command.

ELSE command-statement

Specifies the commands to be executed if the condition is false.

Multiple command statements must be preceded by DO and followed by END.

Usage

Considerations

- An entire conditional expression is evaluated before a result is returned.
 - If the outcome is **true**, CA-ADS executes the commands following the conditional expression.
 - If the outcome is **false**, CA-ADS bypasses commands following the conditional expression and executes commands that specify alternative processing.
- If no alternative processing commands exist for a false outcome, CA-ADS proceeds to the next executable command outside the IF statement.

- IF commands can be nested to any level.
Indentation should be used wherever possible to make statements more readable and to ensure that the required clauses are properly matched.
- A given IF statement can include only one ELSE clause, and that ELSE clause must match the most recent IF command not associated with an ELSE clause.

Examples: Example 1: Using a simple IF command

In this example, a simple IF command tests the status of a map field and executes a DISPLAY command if the condition is true:

```
IF FIELD PROD-NUM IS NOT CHANGED
THEN
    DISPLAY MSG TEXT IS 'ENTER PRODUCT NUMBER'.
```

Example 2: Using an IF command with an ELSE clause

This example includes an ELSE clause to display an alternative message if the field ERROR-FIELD contains 0:

```
IF ERROR-FIELD NE '0'
THEN
    DISPLAY MSG CODE IS 171075 PARM=(MSG-NUM).
ELSE
    DISPLAY MSG TEXT IS 'ENTER NEXT PRODUCT NUMBER'.
```

Example 3: Using a nested IF command

This example illustrates a nested IF command that tests for CA-INDX if DB-END-OF-SET is reached:

```
IF DB-END-OF-SET
THEN
  IF CA-INDX EQ 1
  THEN
    DO.
      MOVE 'NO CUSTOMERS QUALIFY' TO MSG-FIELD.
      MOVE '1' TO ERROR-FIELD.
      RETURN.
    END.
  ELSE
    DISPLAY MSG TEXT IS 'CUSTOMER NUMBER LIST COMPLETE'.
ELSE
  DISPLAY MSG TEXT IS 'ADDITIONAL CUSTOMERS MAY QUALIFY'.
```

14.4 NEXT

Purpose Exits IF and ON command processing.

Syntax:

▶▶ — NEXT command — . —————▶▶

Usage

Definition: The NEXT command is used to exit IF and ON command processing. When a NEXT command is used, control passes to the command following the IF or ON statement.

▶▶ For information about the ON command, see Chapter 16, “Database Access Commands.”

For information about the IF command, see 14.3, “IF” earlier in this chapter.

Considerations

- When used with an ON command, NEXT can be used in conjunction with the THEN/ELSE parameters.
- When used in a nested IF structure, NEXT exits only the current IF statement.

NEXT is typically used in a nested IF structure as a means of associating an ELSE clause with the correct IF command.

Example: The statements below illustrate the use of the NEXT command to match the second ELSE clause with the second IF command:

```
IF A = B
THEN
DO.
  IF X = Y
  THEN
    IF Y = Z
    THEN
      MOVE A TO B.
    ELSE
      NEXT.
  ELSE MOVE B TO A.
  SNAP ALL.
  DISPLAY.
END.
```

If A equals B and X equals Y, but Y does not equal Z, control exits from the innermost IF command and passes to the SNAP ALL command. If the ELSE NEXT statement is not included, the ELSE MOVE B TO A statement is matched incorrectly with the third rather than with the second IF.

14.5 WHILE

Purpose: Creates a processing loop based on conditions in a specified expression.

Syntax:

```
▶▶— WHILE conditional-expression —————▶  
  
▶— repeat. ———┐ command-statement. ———▶  
                  └──────────────────────────┘
```

Parameters

conditional-expression

Evaluates the specified expression and returns a true or false value to the dialog.

Conditional-expression contains one or more conditions to be evaluated and is specified according to the rules presented in Chapter 8, “Conditional Expressions.”

repeat. command-statement

Specifies the commands to be executed as long as the WHILE condition is true. REPEAT begins the WHILE command loop. END terminates the loop. Each command is executed sequentially before the conditional expression is evaluated again.

Command-statement can be any valid CA-ADS process command, including another conditional command.

Usage

Definition: The WHILE command is used to create a processing loop. One or more process commands are executed repeatedly as long as the conditions in a specified expression are true.

Considerations

- The conditional expression is evaluated prior to execution of the first process command.
- Processing continues to loop until the conditional expression is false or as soon as an EXIT or control command is encountered.
- WHILE commands can be nested to any level.

Indentation should be used in coding wherever possible to make the statement more readable and to ensure that the required clauses are properly matched.

Example: The example below illustrates the use of the WHILE command with a nested IF command:

```
MOVE 1 TO CA-INDX.  
OBTAIN NEXT SALES WITHIN PRODUCT-SALES.  
WHILE NOT DB-END-OF-SET AND CA-INDX LE 45  
  REPEAT.  
    IF SALES-AMT GE FULL-AMT  
      DO.  
        MOVE SLS-CUST-NUMBER TO CA-CUST(CA-INDX).  
        ADD 1 TO CA-INDX.  
      END.  
    OBTAIN NEXT SALES WITHIN PRODUCT-SALES.  
  END.  
IF DB-END-OF-SET  
THEN  
  INVOKE 'EOS'.  
ELSE  
  RETURN.
```


Index

Special Characters

\$BACKWARD condition 8-23
\$BATCH condition 8-16
\$DETAIL condition 8-23
\$DETAIL-NOT-FOUND condition 8-23
\$END-OF-DATA condition 8-23
\$END-OF-FILE condition 8-6
\$ERROR-COUNT field 11-9
\$FORWARD condition 8-22
\$HEADER condition 8-23
\$INPUT-COUNT field 11-9
\$IOERROR condition 8-6
\$MAXIMUM-DETAILS-PUT condition 8-24, 17-32
\$MESSAGE field 8-18, 17-33
\$ONLINE condition 8-16
\$OUTPUT-COUNT field 11-9
\$PAGE field 8-18, 17-24
\$PAGE-READY condition 8-22
\$RESPONSE field 8-18
#EFMBIFS macro F-28
#EFUNMOD macro F-31
#EFUNMST macro F-27

A

ABORT command 20-4
ABSOLUTE-VALUE 7-11
ACCEPT command 16-12, 20-8
access module 3-21
activity logging E-3—E-8
 activity logging records E-3—E-6
 function commands E-6
 function numbers E-6
ADB
 See application definition block (ADB)
ADD command 13-6
ADSA
 See application compiler (ADSA)
ADSC
 See dialog compiler (ADSC)
ADSL 1-30
ADSM 1-31
ADSO-APPLICATION-GLOBAL-RECORD A-4—A-14
 AGR-CURRENT-RESPONSE 4-19, 15-17
 definition of A-4
 usage A-4
ADSO-APPLICATION-MENU-RECORD
 at runtime 4-8
 definition of A-15
 usage A-15
ADSO-STAT-DEF-REC record 10-9
ADSOBCOM D-4—D-36
 control statements D-5—D-29
 JCL and command statements D-30
ADSOBSYS D-37—D-48
 ADSOOPTI load module D-37
 JCL and command statements D-39
ADSOBTAT D-48—D-56
 JCL and command statements D-51
 task application table (TAT) D-48
ADSOMSON menu map 4-9
ADSOMUR1 menu map 4-9
ADSOMUR2 menu map 4-9
ADSOOPTI load module
 See ADSOBSYS
ADSORPTS
 application reports B-15
 control statements B-16—B-23
 dialog debugging B-13
 dialog reports B-4—B-5
ADSOTATU D-57—D-59
AFACT-057 record E-4—E-6
AGR-CURRENT-RESPONSE
 See ADSO-APPLICATION-GLOBAL-RECORD
ALLOCATE command 21-10
ALLOWING clause 10-7
AMR-RESPONSE-FIELD
 See ADSO-APPLICATION-MENU-RECORD
APPC (Advanced Program to Program
 Communication) 21-3
APPC status codes 21-30
APPCCODE status code 21-30, 21-31
APPCERC status code 21-30, 21-31
application compiler (ADSA) 2-19, 2-27
 control key assignments 2-10
 Function Definition (Menu) screen 2-32—2-37
 Function Definition (Program) 2-30
 Function Definition (Program) screen 2-32
 Function Definition screen 2-27
 General Options 2-14—2-19
 General Options screen—Page 2 2-16—2-19
 General Options—Page 1 2-14—2-16
 Global Records screen 2-37
 Response Definition screen 2-23

application compiler (ADSA) (*continued*)
 Response/Function List screen 2-19—2-23
 secondary screens 2-32
 Task Codes screen 2-39
 application compiler sequence
 screen sequence 2-7
 application compiler session
 invoking 2-4
 screen sequence 2-10
 suspending 2-10
 application definition block (ADB) 20-12
 application reporter
 See ADSORPTS
 application response
 See response
 application security G-5—G-7
 application structure
 levels of 15-6
 mainline dialogs 15-7
 application thread
 definition of 15-5
 menu stack 15-7
 nonoperative dialogs 15-6
 operative dialogs 15-6
 applications
 compiling of 2-10
 defining global records 2-37
 defining responses and functions 2-19
 defining task codes 2-39
 specifying control blocks 2-30
 specifying menus 2-32
 specifying record buffers 2-30
 APPLICATIONS statement B-16—B-18
 arc cosine values 7-12
 arc sine values 7-13
 arc tangent values 7-14
 AREPORTs B-3
 arithmetic built-in functions
 ABSOLUTE-VALUE 7-11
 INVERT-SIGN 7-30
 LOG-BASE-10 7-34
 LOG-BASE-E 7-34
 MODULO 7-35
 next integer equal or higher 7-36
 next integer equal or lower 7-37
 NUMERIC 7-38
 RANDOM-NUMBER 7-40
 sign inversion 7-30
 SIGN-VALUE 7-45
 SQUARE-ROOT 7-47
 arithmetic commands
 See also arithmetic expression
 ADD 13-6
 COMPUTE 13-7
 DIVIDE 13-8
 MULTIPLY 13-10
 SUBTRACT 13-11
 summary of 13-3
 arithmetic expressions
 binary operations 6-3
 coding rules 6-6
 operands 6-3
 order of evaluation 6-5
 unary operations 6-3
 variable data fields 6-5
 WHERE clause 16-66
 Assembler
 See user program
 assigned key 2-21
 assignment command
 MOVE 13-12
 automatic editing 4-22, 8-20
 autostatus facility 10-4—10-5
 status codes 10-4

B
 BACKWARD function 1-9
 See also system functions
 batch control event conditions
 \$END-OF-FILE (\$EOF) 8-6
 \$IIOERROR (\$IIOERR) 8-6
 batch dialog compiler
 See ADSOBCOM
 batch processing
 \$ERROR-COUNT 11-9
 \$INPUT-COUNT 11-9
 \$OUTPUT-COUNT 11-9
 binary data 5-11
 BIND PROCEDURE command 16-19
 BS2000/OSD JCL
 ADSOBCOM D-35
 ADSOBSYS D-46
 ADSOBTAT D-55
 ADSORPTS B-29
 built-in functions 4-34—4-39
 See also functions
 calling user-defined functions F-42
 changing invocation names F-3
 coding user-defined functions F-42
 creating user-defined functions F-41

built-in functions (*continued*)

- data type conversion 7-4
- date formats 7-6
- error processing 7-4
- internal structure F-4
- invocation name 7-3, F-40
- omitted optional parameter 7-4
- parameters 7-4
- runtime processing F-24
- user-defined 7-5, F-41
- with LRF F-42

C

CA-ADS

- conversion rules 13-12

CA-ADS comment character 5-9

CA-ADS dialog and application reporter

- See* ADSORPTS

CA-ADS statistics block

- See* dialog statistics

CA-IDMS statistics block 16-18

CA-OLQ 4-33

CALL command 19-4

CHANGED condition 8-19

characteristic 9-10

checkouts

- explicit 1-28
- implicit 1-29
- listing 1-30
- modifying with ADSM 1-31
- releasing with ADSM 1-31

checkpoint 4-26, 16-59

- database 16-59
- queue 16-59
- scratch 16-59

CLOSE command 17-10

COBOL

- See also* user program
- conversion rules 13-12

COBOL moves 3-16, 5-17, 13-6, 13-7, 13-8, 13-9, 13-10, 13-12, 13-13, 13-14

coding

- arithmetic expressions 6-6
- CA-ADS comment character 5-9
- general rules 5-8
- SQL comment character 5-9

command status condition

- ERROR-STATUS 8-7

command-statements

- DO 14-5

command-statements (*continued*)

- ELSE 14-5
- END 14-5, 14-10
- REPEAT 14-10
- THEN 14-5

commands 12-4, 14-3

See also Logical Record Facility database access

See also navigational database access

See also process commands

See also VM/ESA commands

ABORT 20-4

ACCEPT 16-12, 20-8

ADD 13-6

arithmetic 13-3

assignment 13-12

BIND PROCEDURE 16-19

CALL 19-4

CLOSE 17-10

COMMIT 16-20

COMPUTE 13-7

conditional 14-3

CONNECT 16-22

CONTINUE 15-10

control 15-3—15-38

DEFINE 19-5

DELETE QUEUE 18-7

DELETE SCRATCH 18-17

DISCONNECT 16-25

DISPLAY 15-12

DIVIDE 13-8

ERASE 16-27, 16-68

EXECUTE NEXT FUNCTION 15-17

EXIT 14-4

FIND/OBTAIN 16-30

GET 16-44

GET DETAIL 17-28

GET QUEUE 18-9

GET SCRATCH 18-19

GOBACK 19-6

IF 14-4

INCLUDE 12-8

INITIALIZE RECORDS

INVOKE 15-19

KEEP 16-46

KEEP LONGTERM 16-47

LEAVE 15-22

LINK 15-24

LRF 16-64—16-76

map modification 17-3

MODIFY 16-53, 16-69

MULTIPLY 13-10

commands (*continued*)
 navigational database access 16-5—16-63
 NEXT 14-8
 OBTAIN 16-70
 pageable map 17-3
 PUT DETAIL 17-30
 PUT QUEUE 18-12
 PUT SCRATCH 18-22
 queue management 18-3
 READ TRANSACTION 15-30
 RETRUN 15-31
 RETURN DB-KEY 16-57
 ROLLBACK 16-59
 scratch management 18-3
 SNAP 20-11
 STORE 16-60, 16-75
 subroutine control 19-3
 SUBTRACT 13-11
 summary of 12-4
 TRACE 20-13
 TRANSFER 15-34
 utility 20-3
 WHILE 14-10
 WRITE PRINTER 20-14
 WRITE TO LOG/OPERATOR 20-18
 WRITE TRANSACTION 15-36
 COMMIT command 16-20
 comparison conditions
 CONTAINS 8-10
 mask characters 8-11
 MATCHES 8-10
 operators 8-10
 compiler (ADSA)
 security G-4
 compiler (ADSC)
 security G-4
 COMPUTE command 13-7
 CONCATENATE built-in function 7-15
 conditional commands
 EXIT 14-4
 IF 14-4
 NEXT 14-8
 WHILE 14-10
 conditional expressions
 \$MESSAGE field 8-18
 \$PAGE field 8-18
 \$RESPONSE field 8-18
 batch control event condition 8-6
 command status condition 8-7
 comparison condition 8-10
 cursor position condition 8-12
 conditional expressions (*continued*)
 dialog execution status condition 8-14
 environment status condition 8-16
 for maps 17-4
 level-88 condition 8-17
 map field status condition 8-18
 map paging status conditions 8-22
 operators in 8-4
 order of precedence 8-4
 set status condition 8-25
 summary 8-5
 CONFIRM command 21-13
 CONFIRMED command 21-14
 CONNECT command 16-22
 constants 9-3
 figurative constants 9-4
 fixed-point numeric literals 9-9
 floating-point numeric literals 9-9
 graphic literals 9-6
 multibit binary 9-7
 nonnumeric literals 9-8
 numeric literals 9-9
 CONTAINS condition 8-10
 CONTINUE command 15-10
 control commands
 BIND PROCEDURE 16-19
 CLOSE 17-10
 CONTINUE 15-10
 DISPLAY 15-12
 EXECUTE NEXT FUNCTION 15-17
 INVOKE 15-19
 LEAVE 15-22
 LEAVE APPLICATION 15-22
 LINK 15-24
 READ TRANSACTION 15-30
 RETURN 15-31
 TRANSFER 15-34
 WRITE TO LOG/OPERATOR 20-18
 WRITE TRANSACTION 15-36
 control event 2-21, 8-6
 See also assigned key
 control keys
 See also application compiler
 See also dialog compiler (ADSA)
 default assignments 4-21
 CONTROL SESSION command 21-15
 control statements
 See also ADSORPTS
 ADSOBSYS D-37
 ADSOBTAT D-49
 COMPILE D-6

control statements (*continued*)

DECOMPILE D-8

conversation, ending 21-25

conversion

CA-ADS rules 13-12

COBOL rules 13-12

cosine values 7-16

currency

See also NOSAVE

database 15-7, 16-5

index 16-58

of area 16-5

of record type 16-5

of run unit 16-5, 16-33

of set type 16-5

queue 18-5

scratch requests 18-15

cursor position condition

CURSOR-COLUMN 8-12

CURSOR-ROW 8-12

cursor position data field

CURSOR-COLUMN 11-8

CURSOR-ROW 11-8

CURSOR-COLUMN condition 8-12

CURSOR-ROW condition 8-12

D

data dictionary

AFACT-057 E-4—E-6

LR-190 E-4

LRACT-193 E-4—E-6, E-8

organization E-4

RCDACT-059 E-4—E-6, E-8

SETACT-061 E-4—E-6

SSA-024 E-4

SSOR-034 E-4

SSR-032 E-4

Data Dictionary Reporter

See AREPORTs

data types

available to CA-ADS 5-10

binary 5-11

conversion of 5-16

conversion rules 13-12

definition of 5-10

doubleword binary 5-11

EBCDIC 5-11

examples of 5-14

floating point 5-12

fullword binary 5-11

data types (*continued*)

group 5-12

halfword binary 5-11

multibit binary 5-13

packed decimal 5-13

zoned decimal 5-13

database

access of 4-25

CA-IDMS statistics block 16-18

CALC key 16-31

checkpoint 16-20

currency 16-5

db-key 16-12, 16-14, 16-34, 16-58

location modes 16-62

modification of CALC elements 16-54

modification of sort-control elements 16-54

monitoring activity 16-47—16-51

set membership options 16-22

statistics 16-17

usage modes 16-55

database access 4-24

Logical Record Facility (LRF) 16-64

database activity

See activity logging

database commands

See also Logical Record Facility database access

See also navigational database access

implicit E-3

database currency

See currency

See NOSAVE

date built-in functions

DATECHG 7-17

DATEDIF 7-20

DATEOFF 7-21

GOODDATE 7-25

TODAY 7-54

TOMORROW 7-56

WEEKDAY 7-62

YESTERDAY 7-66

date formats

calendar 7-6

European 7-6

Gregorian 7-6

Julian 7-6, 11-8

date offset 7-21

DATECHG built-in function 7-17

DBCS data

as a graphic literal 9-6

as a nonnumeric literal 9-8

storage of 13-4

deadlock 4-26
 DEALLOCATE command 21-17
 debugging 4-32
 See also ADSORPTS
 See also online debugger
 See also trace facility
 DEFINE command 19-5
 DELETE QUEUE command 18-7
 DELETE SCRATCH command 18-17
 Design guidelines 21-25
 detail area 17-21
 diagnostic screen
 See Dialog Abort Information screen
 diagnostic table 3-15
 dialog
 See also ADSOBCOM
 FDB B-5—B-11
 Dialog Abort Information screen 4-28, B-14, H-7
 enabling of 20-6
 dialog compiler
 See ADSOBCOM
 dialog compiler (ADSC)
 control keys 3-7
 Dialog Summary Report screen 1-26
 Dialog Summary screen 1-26
 Map Image screen 1-26
 Options and Directives 3-13
 screens 3-10
 session 3-4
 dialog compiler session
 invoking 3-4
 suspending 3-8
 dialog execution status
 FIRST-TIME 8-14
 dialog expression (ADSOBCOM) D-10—D-29
 dialog function 1-9
 See also function
 dialog reporter
 See ADSORPTS
 Dialog Selection screen 4-4
 dialog statistics
 CA-ADS statistics block C-5
 checkpoint interval C-10
 enabling of C-8
 runtime collection and writing C-11
 selecting C-9
 statistics block identifiers C-11
 statistics reporting C-12
 transaction statistics block C-4
 Dialog Summary screen 1-26
 Map Image screen 1-26
 dialog, location of allocated 21-25
 dialogs
 mainline 4-3
 DIALOGS statement B-19—B-21
 DISCONNECT command 16-25
 DISPLAY command 15-12
 mapout rules 15-14
 status test outcome 8-14
 DIVIDE command 13-8
 DO command-statement 14-5
 double-byte character set
 See DBCS
 doubleword binary data type 5-11
 dumps
 snap dumps 20-6

E

EBCDIC data type 5-11, 13-4
 EDIT IS ERROR/CORRECT condition 8-18
 ELSE command-statement 14-5
 END command-statement 14-5, 14-10
 environment status condition
 \$BATCH 8-16
 \$ONLINE 8-16
 ERASE command 16-27, 16-68
 ERASED condition 8-20
 error expressions 10-6
 error handling
 ADSO-STAT-DEF-REC 10-9
 ALLOWING clause 10-7
 autostatus 10-4—10-5
 built-in functions F-25
 error expressions 10-6
 level-88 condition names 10-9
 site defined status definition record 10-10
 STATUS clause 10-9
 status definition record 10-9
 system-defined status definition record 10-9
 error messages
 suppression of 17-17
 ERROR-STATUS condition 8-7
 exclusive usage mode 16-55
 EXECUTE NEXT FUNCTION command 8-14
 mapless dialog 15-17
 EXECUTE ON EDIT ERRORS command 8-20
 execution modes 2-16, 3-3
 EXIT command 14-4
 explicit checkouts 1-28
 explicit releases 1-28

expression description element

See XDE module

extended run units 4-25—4-27

See also run unit

checkpoint 4-26

deadlock 4-26

EXTRACT built-in function 7-23

F

fast mode 2-16

FDB

See fixed dialog block

See fixed dialog block (FDB)

figurative constants 9-4

FIND/OBTAIN

FIND 16-31

FIND/OBTAIN command

FIRST-TIME condition 8-14, 15-34

FIX built-in function 7-24

fixed dialog block (FDB) 20-11

contents of B-5—B-11

fixed-point numeric literals 9-9

floating point data types

display 5-12

internal long 5-12

internal short 5-12

floating-point numeric literals 9-9

flow of control 4-19—4-22

automatic editing 4-22

default control key assignments 4-21

footer area 17-21

FORWARD function 1-9

See also system functions

fullword binary data type 5-11

function

See dialog function

See menu function

See menu/dialog function

See program function

Function Definition (Dialog) screen 2-27

Function Definition (Menu) screen 2-32

Function Definition (Program) screen 2-30

functions

See built-in functions

G

G-literals

See graphic literals

General Options screen—Page 2 2-16—2-19

General Options—Page 1 2-14—2-16

GET command 16-44

GET DETAIL command 8-23, 17-28

GET QUEUE command 18-9

GET SCRATCH command 18-19

Global Records screen (ADSA) 2-37

GOBACK command 19-6

GOODDATE built-in function 7-25

GOODTRAILING built-in function 7-26

graphic literals 9-6

group data type 5-12

H

halfword binary data type 5-11

header area 17-21

HELP function 1-9

See also system functions

help screen 4-16

I

ICTL statement

ADSOBCOM D-4

ADSOBSYS D-37

IDENTICAL condition 8-19

IF command 14-4

implicit checkouts 1-29

implicit releases 1-29

IN ERROR condition 8-18, 8-20

INCLUDE command 12-8

INDEX built-in function 7-48

INITCAP built-in function 7-27

INITIALIZE RECORDS command 20-10

INSERT built-in function 7-28

INSERT directive 12-8

intermediate result area (IRA) F-24

INVERT-SIGN built-in function 7-30

INVOKE command 15-19

status test outcome 8-14

ISEQ statement

ADSOBCOM D-4

ADSOBSYS D-37

J

JCL

See BS2000/OSD JCL

See OS/390 JCL

See VM/ESA commands

See VSE/ESA JCL

K

KEEP commands 16-46
KEEP LONGTERM command

L

LEAVE APPLICATION command 15-22
LEAVE command 15-22
 status test outcome 8-14
LEFT-JUSTIFY built-in function 7-31
length
 See STRING-LENGTH built-in function
level-88 condition 8-7, 8-17
LIKE built-in function 7-32
LINK command
 linking to OLQ 4-33
 nesting 15-26
 status test outcome 8-14
 with a user program 15-26
LIST statement B-21
literal
 See constants
local mode processing
 SYSIDMS parameter file B-24
location mode
 CALC 16-62
 DIRECT 16-62
 VIA 16-62
log
 See activity logging
LOG-BASE-10 built-in function 7-34
LOG-BASE-E built-in function 7-34
logarithms
 See arithmetic built-in functions
Logical Record Facility
 linking to dialog with LRF subschema 15-27
logical records
 See also Logical Record Facility database access
 in database access 16-64
 path 16-64
LR-190 record E-4
LRACT-193 record E-4—E-8
LRF
 path status 16-72
LRF commands
 ERASE 16-68
 MODIFY 16-69
 OBTAIN 16-70
 ON command 16-71
 STORE 16-75

LRF commands (*continued*)

 WHERE clause 16-65
LU 6.2 21-3

M

mainline dialogs 15-7
mantissa values 9-10
map field status conditions
 ALL BUT 8-19
 CHANGED 8-19
 ERASED 8-20
 EXCEPT 8-19
 IDENTICAL 8-19
 IN ERROR 8-18, 8-20
 pageable map considerations 8-20
 TRUNCATED 8-20
Map Image screen 1-26
map modification commands
 ATTRIBUTES 17-5—17-9
 MODIFY MAP 17-12—17-20
map paging session 17-23
map paging status condition 8-22
 See also pageable maps
maps
 See also Map Image screen
 See also map modification commands
 See also menu map
 See also pageable maps
 See also screens
 conditional expressions 17-4
 output data options 17-16
 permanent modifications 17-7, 17-13
 suppressing error message 17-17
 temporary modifications 17-7, 17-13
mask character 7-32, 8-11, 16-67
master function table F-4, F-5
MATCHES condition 8-10
matching string 7-32
menu definition 4-8
menu function 1-9
 See also function
menu maps
 ADSOMSON 4-9
 ADSOMUR1 4-9
 ADSOMUR2 4-9
 signon 4-13—4-16
 site-defined 4-9
 system-defined 4-10—4-13
menu stack 15-7

menu/dialog function 1-9
 See also function
message codes 15-15
messages 15-15
modified data tags (MDTs)
 resetting 17-14
 setting for map fields 17-19
MODIFY command 16-53, 16-69
MODIFY MAP
MODIFY MAP command 8-20, 17-12
modules
 See process modules
MODULO built-in function 7-35
MOVE command 13-12
multibit binary constants 9-7
multibit binary data type 5-13
multiple databases
 accessing 11-6
MULTIPLY command 13-10

N

native VSAM data sets 16-7—16-9
 currency requests 16-15
 set status condition 16-8
 with CONNECT 16-23
 with DISCONNECT 16-26
 with ERASE 16-27
 with FIND/OBTAIN OWNER 16-38
 with MODIFY 16-55
natural logarithm 7-34
navigational DML commands
 ACCEPT 16-12
 COMMIT 16-20
 CONNECT 16-22
 DISCONNECT 16-25
 ERASE 16-27
 FIND/OBTAIN 16-30
 GET 16-44
 KEEP 16-46
 MODIFY 16-53
 READY 16-55
 RETURN DB-KEY 16-57
 ROLLBACK 16-59
 STORE 16-60
nesting 15-26
NEXT command 14-8
NEXT-INTEGER-EQUAL-HIGHER built-in
 function 7-36
NEXT-INTEGER-EQUAL-OR-LOWER built-in
 function 7-37

nonnumeric literals 9-8
NUMERIC built-in function 7-38
numeric fields 13-4
numeric literals 9-9

O

OBTAIN command 16-70
OCB
 See online control block (OCB)
OCTL statement
 ADSOBCOM D-4
 ADSOBSYS D-37
OLQ
 See CA-OLQ
ON command 16-71
online control block (OCB) 20-11
online debugger H-7—H-10
online help
 in CA-ADS applications 4-8
 in CA-ADS compilers 1-32
online terminal block (OTB) 20-11
online terminal block extension (OTBX) 20-11
online work area (OWA) 20-11
Options and Directives 3-13
OS/390 JCL
 ADSOBCOM D-30
 ADSOBSYS D-39
 ADSOBTAT D-51
 ADSORPTS B-25
OTB
 See online terminal block (OTB)
OTBX
 See online terminal block extension (OTBX)
OWA
 See online work area (OWA)

P

packed decimal data type 5-13
pageable map commands
 GET DETAIL 17-28
 PUT DETAIL 17-30
pageable maps
 \$PAGE field 17-24
 areas of 17-21
 Auto display specification 3-19
 Backpage specification 3-19
 flow of control 17-25
 map paging dialog options 3-18, 17-27—17-28
 map paging session 17-22

pageable maps (*continued*)
 severity codes 17-33
 UPDATE specification 3-19
parameters for process commands
 keywords 5-7
 variable terms 5-7
path status (LRF) 16-72
PF keys
 See application compiler
PL/I
 See user program
POP function 1-9
 See also system functions
POPTOP function 1-9
 See also system functions
PREPARE-TO-RECEIVE command 21-19
printer output 17-15
process commands
 See also commands
 coding of 5-8
 comment character 5-9
 quoted strings 5-9
process modules
 premap 5-5
 response 5-5
processing
 cooperative 21-3
program
 See user program
program function 1-9
 See also function
 See also user program
protected usage mode 16-55
PUT DETAIL command 8-20, 17-30
PUT QUEUE command 18-12
PUT SCRATCH command 18-22

Q

queue management commands
 DELETE QUEUE 18-7
 GET QUEUE 18-9
 PUT QUEUE 18-12
queue records 18-5
QUIT function 1-9
 See also system functions
quoted strings
 See process commands

R

RANDOM-NUMBER built-in function 7-40
RBB
 See record buffer block (RBB)
RCDACT-059 record E-4—E-8
READ TRANSACTION command 15-30
READY command 16-55
RECEIVE-AND-WAIT command 21-20
record buffer block (RBB) 20-12
record locking
 queue 18-5
record locks
 deadlock conditions 16-10
 exclusive 16-9
 explicit 16-9
 implicit 16-9
 long-term explicit 16-9
 release of 16-20
 retrieval locks 16-10—16-12
 shared 16-9
records
 queue 18-5
 scratch 18-15
recovery 16-59
releases
 explicit 1-28
 implicit 1-29
releasing entities 1-29
remainder values 7-35
REPEAT command-statement 14-10
repeat string 7-50
REPLACE built-in function 7-42
replace string 7-42
reports
 See ADSORPTS
 See AREPORTs
REQUEST-TO-SEND command 21-21
REQUEST-TO-SEND-RECEIVED system field 21-30,
 21-34
reset keyboard 17-14
response 1-9
 runtime selection 4-9
Response Definition screen (ADSA) 2-23
response process
 security 3-29
Response/Function List screen 2-19
Response/Function List screen (ADSA) 2-19
Response/Function search 2-22
responses
 runtime selection 4-16

responses (*continued*)
 security 4-20
 RETURN command 15-31
 status test outcome 8-14
 RETURN DB-KEY command 16-57
 RETURN function 1-9
 See also system functions
 RHDCEVBF source module F-28
 RIGHT-JUSTIFY built-in function 7-44
 ROLLBACK command 16-59
 run units 4-25
 extended 4-25—4-27
 usage modes 4-26
 runtime system
 ADSOMAIN 4-3
 ADSORUN1 4-3
 initiation of 4-3—4-7

S

scratch area 18-15
 scratch management commands
 DELETE SCRATCH 18-17
 GET SCRATCH 18-19
 PUT SCRATCH 18-22
 scratch records 18-15
 screens
 See also application compiler
 See also dialog compiler
 See also maps
 application compiler 2-11
 Dialog Abort Information screen 4-28—4-31
 Dialog Selection screen 4-4
 Dialog Summary Report screen 1-26
 Dialog Summary screen 1-26
 Function Definition (Dialog) screen 2-27
 Function Definition (Menu) screen 2-32
 Function Definition (Program) 2-30
 General Options—Page 1 (ADSA) 2-14—2-16
 General Options—Page 2 (ADSA) 2-16—2-19
 Global Records 2-37
 help 4-16—4-18
 Map Image screen 1-26
 Menu definition 4-8
 Options and Directives (ADSC) 3-13
 Response Definition 2-23
 Response/Function List 2-19
 Response/Function List screen 2-19
 screens 3-10
 task codes 2-39
 TAT Update Utility D-58

SEARCH statement B-22
 search, Response/Function 2-22
 security
 See also application security
 ADAPGOP2 2-17
 ADSOBCOM D-4
 General Options screen—Page 2 2-16—2-19
 response G-5
 response process 3-29
 security-tailored menus G-6
 signon G-6—G-7
 SEND-DATA command 21-22
 SEND-ERROR command 21-24
 SEND/RECEIVE commands 21-3
 summary 21-9
 SET condition 8-25
 set membership options 16-22
 SETACT-061 record E-4—E-6
 sign inversion 7-30
 SIGN-VALUE built-in function 7-45
 SIGNOFF function 1-9, G-7
 See also system functions
 SIGNON function 1-9, G-6
 See also system functions
 SIGNON statement (ADSOBCOM) D-5
 sine values 7-46
 SNA (Systems Network Architecture) 21-3
 SNAP command 20-11
 sort key 16-42
 sorted set 16-42
 source code
 See process modules
 source module
 See process modules

SQL
 access module 3-21
 compliance 3-22
 SQL comment character 5-9
 SQUARE-ROOT 7-47
 SREPORTs C-12
 SSA-024 area E-4
 SSOR-034 set E-4
 SSR-032 record E-4
 statistics
 See dialog statistics
 status codes 21-31
 status definition record
 ADSO-STAT-DEF-REC 10-9
 level-88 condition names 10-9
 site defined 10-10
 STATUS clause 10-9

status definition record (*continued*)

 system defined 10-9

step mode 2-16

storage

 management 4-40

 XA 4-40

STORE command 16-60, 16-75

string built-in functions

 CONCATENATE 7-15

 EXTRACT 7-23

 FIX 7-24

 INDEX 7-48

 INITCAP 7-27

 INSERT 7-28

 LEFT-JUSTIFY 7-31

 LIKE 7-32

 REPLACE 7-42

 RIGHT-JUSTIFY 7-44

 STRING-INDEX 7-48

 STRING-LENGTH 7-49

 STRING-REPEAT 7-50

 SUBSTRING 7-51

 TOLOWER 7-55

 TOUPPER 7-57

 TRANSLATE 7-59

 VERIFY 7-61

 WORDCAP 7-65

string verification 7-61

STRING-LENGTH built-in function 7-49

STRING-REPEAT built-in function 7-50

subroutine control commands

 CALL 19-4

 DEFINE 19-5

 GOBACK 19-6

Subschema Control Block 16-20

SUBSCHEMA-CONTROL 15-25

SUBSTRING built-in function 7-51

SUBTRACT command 13-11

suspense file 15-37

symbol table 3-15

SYSIDMS parameters

 for physical requirements B-24

system fields 21-30, 21-34

system functions 1-9

See also function

 BACKWARD 1-9, A-13

 FORWARD 1-9, A-13

 HELP 1-9, 4-16

 POP 1-9

 POPTOP 1-9

 QUIT 1-9

system functions (*continued*)

 RETURN 1-9

 SIGNOFF 1-9, G-7

 SIGNON 1-9, G-6

 TOP 1-9

system records

See ADSO-APPLICATION-GLOBAL-RECORD

See ADSO-APPLICATION-MENU-RECORD

See ADSO-STAT-DEF-REC

SYSTEM statement (ADSOBSYS) D-38

system-supplied data fields 11-6

 \$ERROR-COUNT 11-9

 \$INPUT-COUNT 11-9

 \$OUTPUT-COUNT 11-9

 CURSOR-COLUMN 11-8

 CURSOR-ROW 11-8

 DATE 11-8

 DB-NAME 11-6

 DIRECT-DBKEY 11-6

 ERROR-STATUS 11-8

 LENGTH 11-8

 NODE-NAME 11-6

 TIME 11-9

T

tables

 diagnostic 3-15

 symbol 3-15

task 4-24

task application table (TAT) 2-39, 4-4, 20-12

See also ADSOBTAT

See also ADSOTATU

task code, for TCF 2-4

Task Codes screen (ADSA) 2-39

TAT

See task application table (TAT)

TAT Update Utility D-58

TCF

See transfer control facility

test condition

See conditional expressions

test conditions

 WHERE clause 16-65

THEN command-statement 14-5

TODAY built-in function 7-54

TOLOWER built-in function 7-55

TOMORROW built-in function 7-56

TOP function 1-9

See also system functions

TOUPPER built-in function 7-57
TRACE 20-13, H-6
trace facility H-5—H-6
trailing sign built-in functions
 GOODTRAILING 7-26
 TRAILING-TO-ZONED 7-58
 ZONED-TO-TRAILING 7-67
TRAILING-TO-ZONED built-in function 7-58
transaction statistics
 See dialog statistics
TRANSFER command 15-34
 status test outcome 8-14
transfer control facility 2-4, 3-4—3-5
TRANSLATE built-in function 7-59
trigonometric built-in functions
 ARCCOSINE-DEGREES 7-12
 ARCCOSINE-RADIANS 7-12
 ARCSINE-DEGREES 7-13
 ARCSINE-RADIANS 7-13
 ARCTAN-DEGREES 7-14
 ARCTAN-RADIANS 7-14
 COSINE-DEGREES 7-16
 COSINE-RADIANS 7-16
 SINE-DEGREES 7-46
 SINE-RADIANS 7-46
TRUNCATED condition 8-20

U

usage modes 4-26
 exclusive 16-56
 protected 16-56
 retrieval 16-55
 shared 16-56
 update 16-55
user program
 DC RETURN statement 15-27
 linking 15-26
user program function
 See program function
utilities
 ADSOBCOM D-4—D-29, D-36
 ADSOBSYS D-37—D-48
 ADSOBTAT D-48—D-56
 ADSORPTS B-3—B-30
 ADSOTATU D-57—D-59
utility commands
 ABORT 20-4
 ACCEPT 20-8
 INITIALIZE RECORDS 20-10
 SNAP 20-11

utility commands (*continued*)
 TRACE 20-13
 WRITE PRINTER command 20-14

V

variable dialog block (VDB) 20-12
variable expression description element
 See VXDE module
variable terms
 types of 5-8
variables
 arithmetic expressions 6-3—6-6
 conditional expressions 8-3
 constants 9-3
 data fields 11-3
 entity names 11-12
 error expressions 10-6
 system-supplied 11-6
 target fields 11-10
 user-defined 11-4
 variable target fields 11-10

VDB

See variable dialog block (VDB)

VDE module

processing of F-17

vector call codes B-11

VERIFY built-in function 7-61

VM/ESA commands

ADSOBCOM D-33

ADSOBSYS D-44

ADSOBTAT D-54

ADSORPTS B-27

VM/ESA systems

See VM/ESA commands

VSAM data sets

See native VSAM data sets

VSE/ESA JCL

ADSOBCOM D-31

ADSOBSYS D-42

ADSOBTAT D-52

ADSORPTS B-26

VXDE module F-4, F-8—F-17

processing of F-17

W

WEEKDAY built-in function 7-62

WHAT-RECEIVED system field 21-30, 21-34

WHERE clause 16-65

 comparison expression 16-66

WHERE clause (*continued*)

conditional expression 16-65

test condition 16-66

WHILE command 14-10

with FIND/OBTAIN DB-KEY 16-36

WORDCAP built-in function 7-65

WRITE PRINTER command

WRITE TO LOG/OPERATOR

WRITE TO LOG/OPERATOR command 20-18

WRITE TRANSACTION command 15-36

X

XDE module F-4, F-6, F-8—F-17

Y

YESTERDAY built-in function 7-66

Z

zoned decimal data type 5-13

ZONED-TO-TRAILING built-in function 7-67