

CA ADS™ for CA IDMS™

ADS Reference Guide

Release 18.5.00, 2nd Edition



This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the "Documentation") is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or 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 YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2014 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

CA Technologies Product References

This document references the following CA products:

- CA IDMS™/DB
- CA IDMS™/DC (DC)
- CA IDMS™/DC or CA IDMS™ UCF (DC/UCF)
- CA IDMS™ OLQ

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At <http://ca.com/support>, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at <http://ca.com/docs>.

Documentation Changes

The following documentation updates were made for the 18.5.00, 2nd Edition release of this documentation:

- [DISPLAY](#) (see page 334)—Updated the syntax diagram.

The following documentation updates were made for the 18.5.00 release of this documentation:

- [Overview of ACCEPT](#) (see page 373)—Added information to the ACCEPT sections to document the new *page-info* parameter.
- [Extended Run Units](#) (see page 143), [Overview of Navigational Database Access](#) (see page 365), [READY](#) (see page 423)—Added information concerning the use of the new FORCE option.
- [APPCCODE and APPCERC](#) (see page 553)—Added an overview table summarizing the APPCCODEs.
- [Runtime Error-Status Codes](#) (see page 737)—This appendix was newly added, previously it was available in the ADS Quick Reference Guide.
- [Online Debugger Syntax](#) (see page 751)—This appendix was newly added, previously it was available in the ADS Quick Reference Guide.

Contents

Chapter 1: Introduction to CA ADS 19

Introduction.....	19
Syntax Diagram Conventions	19
What is CA ADS?	21
What CA ADS Does.....	22
Creating a CA ADS Application.....	23
Tools Used To Develop an Application.....	24
The CA ADS Application Compiler (ADSA)	25
Mapping Facilities (MAPC and the Batch Compiler/Utility)	27
CA ADS Dialog Compilers (ADSC and ADSOBCOM)	28
IDD Menu Facility and Online IDD.....	30
The CA ADS Runtime System.....	31
CA ADS Screens.....	31
Action Bar	33
Action Bar Actions.....	35
Checkout and Release Procedures.....	44
How to check out or release an entity	45
Listing Checkouts (ADSL).....	47
Modifying Checkouts (ADSM)	48
CA ADS Help Facility.....	48

Chapter 2: CA ADS Application Compiler (ADSA) 51

Overview.....	51
Application Compiler Session.....	51
Invoking the Application Compiler.....	52
Sequencing Through Application Compiler Screens	53
Suspending a Session.....	56
Terminating a Session.....	57
Application Compiler Screens	57
Main Menu	57
General Options Screen—Page 1	60
General Options Screen—Page 2	63
Response/Function List Screen	66
Response Definition Screen.....	70
Function Definition (Dialog) Screen	74
Function Definition (Program) Screen	77

Function Definition (Menu) Screen	80
Global Records Screen.....	85
Task Codes Screen.....	87

Chapter 3: CA ADS Dialog Compiler (ADSC) 91

Overview.....	91
Dialog Compiler Session.....	92
Invoking the Dialog Compiler.....	92
Sequencing Through Dialog Compiler Screens	94
Suspending a Session.....	97
Terminating a Session.....	98
Dialog Compiler Screens	98
Main Menu.....	98
Options and Directives Screen	101
Map Specifications Screen.....	105
Database Specifications Screen.....	109
Records and Tables Screen.....	111
Process Modules Screen	114

Chapter 4: CA ADS Runtime System 119

Initiating the CA ADS Runtime System	119
How to Define Runtime Tasks.....	119
How to Start a CA ADS Application	120
Runtime Menu and Help Screens	124
Menu Screens	124
Site-Defined Menu Maps.....	126
System-Defined Menu Maps.....	126
Application Help Screen	133
Runtime Flow Of Control	135
Effects of Automatic Editing on Flow of Control	139
Message Prefixes.....	140
CA ADS Tasks, Run Units, and Transactions	141
Run units and database access.....	142
Extended Run Units	143
Dialog Abort Information Screen.....	145
Debugging a Dialog.....	148
Linking From CA ADS To CA OLQ	149
Linking to CA OLQ.....	149
Passing Syntax to CA OLQ	149
Linking Built-In Functions With The Runtime System.....	150
ADSOVCON Module Creation	150

Managing Storage.....	151
Adjusting Record Compression.....	151
Calculating RBB Storage	152
Writing Resources to Scratch Records	152
Using XA Storage	153

Chapter 5: Introduction to Process Language **155**

Overview.....	155
Process Modules	156
Creating Process Modules	156
Adding Process Modules to Dialogs.....	157
Executing Process Modules.....	157
Process Commands.....	159
Constructing Commands.....	159
Coding Considerations.....	160
Data Types	161
Conversion Between Data Types	167

Chapter 6: Arithmetic Expressions **171**

Overview.....	171
Syntax.....	171
Syntax: Arithmetic-Expression.....	172
Evaluation Of Arithmetic Expressions	173
Evaluation of Arithmetic Expressions	174
Coding Considerations.....	174

Chapter 7: Built-in Functions **175**

Overview.....	177
Invocation Names	178
Built-In Function Values.....	178
Coding Parameters.....	179
User-Defined Built-In Functions	179
System-Supplied Functions	179
Arithmetic Functions	179
Date Functions.....	180
Date-Time Stamp Functions.....	181
String Functions.....	182
Trailing-Sign Functions	183
Trigonometric Functions.....	184
ABSOLUTE-VALUE.....	185

ARC COSINE	186
ARC SINE	187
ARC TANGENT	188
CONCATENATE	189
COSINE	190
DATECHG	191
DATEDIF	194
DATEEXT	196
DATEINT	196
DATEOFF	197
DATETIMX	199
DISPDT	199
DTINT	200
EXTRACT	201
FIX	202
GOODDATE	203
GOODTRAILING	204
INITCAP	205
INSERT	206
INVERT-SIGN	207
LEFT-JUSTIFY	208
LIKE	209
LOGARITHM	210
MODULO	211
NEXT-INT-EQHI	212
NEXT-INT-EQLO	213
NUMERIC	214
RANDOM-NUMBER	216
REPLACE	218
RIGHT-JUSTIFY	219
SIGN-VALUE	220
SINE	220
SOCKET	221
SQUARE-ROOT	223
STRING-INDEX	223
STRING-LENGTH	224
STRING-REPEAT	225
SUBSTRING	226
TANGENT	228
TIMEEXT	229
TIMEINT	230
TODAY	231

TOLOWER	232
TOMORROW	233
TOUPPER	234
TRAILING-TO-ZONED	235
TRANSLATE	236
VERIFY	237
WEEKDAY	238
WORDCAP	240
YESTERDAY	241
ZONED-TO-TRAILING	242

Chapter 8: Conditional Expressions **245**

Overview	245
General Considerations	246
Syntax for Conditional Expressions	247
Batch-Control Event Condition	248
Command Status Condition	249
Comparison Condition	251
Cursor Position Condition	253
Dialog Execution Status Condition	254
Environment Status Condition	256
Level-88 Condition	257
Map Field Status Condition	257
Map Paging Status Conditions	261
Set Status Condition	265
Arithmetic and Assignment Command Status Condition	266

Chapter 9: Constants **269**

Overview	269
Figurative Constants	269
Graphic Literals	271
Multibit Binary Constants	272
Nonnumeric Literals	273
Numeric literals	273

Chapter 10: Error Handling **277**

Overview	277
The Autostatus Facility	278
Status Codes Returned by the Autostatus Facility	279
Error Expressions	279

The ALLOWING Clause	280
Status Definition Records	281
ADSO-STAT-DEF-REC.....	284

Chapter 11: Variable Data Fields **285**

Overview	285
User-Defined Data Field Names	285
System-Supplied Data Field Names	287
Entity Names	293

Chapter 12: Introduction to Process Commands **295**

Overview	295
Summary Of Process Commands	296
INCLUDE	300
Dialog Compiler Directive.....	301

Chapter 13: Arithmetic and Assignment Commands **303**

Overview	303
General Considerations.....	304
Numeric Fields	304
EBCDIC and DBCS Fields	304
Arithmetic and Assignment Command Status Condition.....	305
Arithmetic Commands.....	306
ADD	306
COMPUTE	307
DIVIDE.....	309
MULTIPLY	311
SUBTRACT	313
Assignment Command	314
MOVE.....	315

Chapter 14: Conditional Commands **317**

Overview	317
EXIT	318
IF.....	319
NEXT	321
WHILE	322

Chapter 15: Control Commands 325

Overview.....	325
General Considerations.....	326
Application Thread.....	327
Operative and Nonoperative Dialogs	327
Application Levels	328
Mainline Dialog.....	328
The Menu Stack.....	328
Database Currencies.....	329
CONTINUE.....	332
DISPLAY	334
EXECUTE NEXT FUNCTION	339
INVOKE	341
LEAVE.....	343
LINK.....	345
READ TRANSACTION	353
RETURN	353
TRANSFER	356
WRITE TRANSACTION	358

Chapter 16: Database Access Commands 363

Overview.....	363
Navigational DML.....	365
Overview of Navigational Database Access.....	365
Use of Native VSAM Data Sets	368
Record Locking.....	369
Suppression of Record Retrieval Locks.....	371
Overview of ACCEPT	373
ACCEPT DB-KEY FROM CURRENCY	374
ACCEPT DB-KEY RELATIVE TO CURRENCY	376
ACCEPT PAGE-INFO	378
ACCEPT STATISTICS	380
BIND PROCEDURE	383
COMMIT.....	384
CONNECT	386
DISCONNECT	389
ERASE.....	391
Overview of FIND/OBTAIN.....	394
FIND/OBTAIN CALC	394
FIND/OBTAIN CURRENT	397
FIND/OBTAIN DB-KEY	399

FIND/OBTAIN OWNER	402
FIND/OBTAIN WITHIN SET/AREA	404
FIND/OBTAIN WITHIN SET USING SORT KEY	408
GET	411
KEEP	412
KEEP LONGTERM	413
MODIFY	420
READY	423
RETURN DB-KEY	426
ROLLBACK	428
STORE	429
Logical Record Facility Commands	433
Overview of LRF Database Access	433
WHERE Clause.....	434
Conditional Expression.....	434
Comparison Expression.....	436
ERASE.....	438
MODIFY	439
OBTAIN	440
ON Command.....	442
STORE	445

Chapter 17: Map Commands 449

Overview.....	449
Map Modification Commands	450
Attributes Command	450
CLOSE.....	454
MODIFY MAP.....	455
Pageable Maps.....	464
Areas of a Pageable Map	465
Map Paging Session	466
Map Paging Dialog Options	471
GET DETAIL	472
PUT DETAIL.....	474
Creating or Modifying a Detail Occurrence of a Pageable Map	477
Specifying a Numeric Value Associated with an Occurrence	477
Specifying a Message to Appear in the Message Field of an Occurrence.....	477

Chapter 18: Queue and Scratch Management Commands 483

Overview.....	483
Queue Records	485

DELETE QUEUE.....	486
GET QUEUE.....	488
PUT QUEUE.....	491
Scratch Records.....	494
CA ADS Usage.....	494
CA ADS Batch Considerations.....	495
DELETE SCRATCH.....	496
GET SCRATCH.....	498
PUT SCRATCH.....	502

Chapter 19: Subroutine Control Commands **505**

Overview.....	505
CALL.....	505
DEFINE.....	506
GOBACK.....	507

Chapter 20: Utility Commands **509**

Overview.....	509
ABORT.....	510
ACCEPT.....	513
INITIALIZE RECORDS.....	515
SNAP.....	516
TRACE.....	518
WRITE PRINTER.....	519
WRITE TO LOG/OPERATOR.....	523

Chapter 21: Cooperative Processing Commands **527**

Using SEND/RECEIVE Commands.....	527
How Cooperative Processing Works.....	528
Sample Cooperative Application.....	528
Program A: Client Listing (PC).....	530
Dialog B: Server listing (Mainframe).....	532
SEND/RECEIVE Commands.....	534
ALLOCATE.....	535
CONFIRM.....	538
CONFIRMED.....	539
CONTROL SESSION.....	540
DEALLOCATE.....	541
PREPARE-TO-RECEIVE.....	543
RECEIVE-AND-WAIT.....	543

REQUEST-TO-SEND.....	545
SEND-DATA.....	545
SEND-ERROR	546
Design Guidelines.....	547
Understanding Conversation States	548
Conversation States.....	549
Conversation States in a Successful Data Transfer	550
Testing APPC Status Codes and System Fields.....	552
Status Codes.....	552
System Fields.....	552
When APPC Status Codes and System Field Values are Returned	552
APPCCODE and APPCERC	553
System Fields.....	557

Chapter 22: OSCaR Commands **559**

OSCaR Command Syntax.....	559
OPEN.....	560
SEND	561
CLOSE.....	562
RECEIVE	563
Sample OSCaR Application	563
OSCaR to APPC Mapping.....	565

Appendix A: System Records **567**

Overview.....	567
ADSO-APPLICATION-GLOBAL-RECORD	568
ADSO-APPLICATION-GLOBAL-RECORD	576
ADSO-APPLICATION-MENU-RECORD.....	579
ADSO-APPLICATION-MENU-RECORD.....	582

Appendix B: CA ADS Dialog and Application Reporter **583**

Overview.....	583
AREPORTs Documenting CA ADS Dialogs.....	583
Dialog Reports.....	584
Application Reports	595
Control Statements	596
APPLICATIONS.....	596
DIALOGS	599
LIST.....	603
SEARCH.....	603

SYSIDMS Parameter File.....	604
JCL and Commands To Run Reports	605

Appendix C: Dialog Statistics **611**

Overview.....	611
Collecting Selected Statistics.....	611
Enabling Dialog Statistics.....	615
Selecting Dialogs.....	616
Setting a Checkpoint Interval.....	617
Collecting and Writing Statistics.....	617
Statistics Reporting.....	618

Appendix D: Application and Dialog Utilities **621**

Overview.....	621
ADSOBCOM	621
Standard Control Statements	622
Special Control Statements	623
SIGNON	623
COMPILE	624
DECOMPILE.....	626
Dialog-expression.....	628
JCL and Commands	649
ADSOBSYS	654
Control Statements	655
SYSTEM Statement.....	655
JCL and Commands	656
ADSOBTAT	662
Control Statements	664
JCL and Commands	666
ADSOTATU	671
TAT Update Utility Screen.....	672

Appendix E: Activity Logging for an CA ADS Dialog **675**

Overview.....	675
Data Dictionary Organization.....	676
Activity Logging Record Formats	676

Appendix F: Built-in Function Support **681**

Overview.....	681
---------------	-----

Internal Structure Of Built-In Functions	681
Master Function Table	682
Model XDE Module	683
XDEs and VXDEs	685
Processing Program Modules	693
Runtime Processing of Built-In Functions	699
Assembler Macros	701
#EFUNMST	701
RHDCEVBF	702
#EFUNMOD	705
Changing Invocation Names	713
Creating User-Defined Built-In Functions	714
Steps for Generating a User-Defined Built-In Function	714
LRF Considerations for User-Defined Built-In Functions	715
Calling a User-Defined Built-In Function	715

Appendix G: Security Features **717**

Overview	717
CA ADS Compiler Security	718
CA ADS Application Security	719
Response Security	719
Signon Security	720

Appendix H: Debugging an CA ADS Dialog **723**

Overview	723
Creating a Symbol Table	723
Trace Facility	725
Online Debugger	727

Appendix I: Compiler Overview and Default Control Keys **733**

Summary of Application Compiler Process	733
Default Control Keys	734
Summary of Dialog Compiler Process	735
Default Control Keys	736

Appendix J: Runtime Error-Status Codes **737**

Status Codes Returned by the Autostatus Facility	737
Major DB Status Codes	738
Minor DB Status Codes	738

Major DC Status Codes.....	743
Minor DC Status Codes.....	744
ERROR-STATUS Condition Names	748
Autostatus Return Codes	748
Default Level-88 Values.....	749

Appendix K: Online Debugger Syntax 751

General Registers Symbols	751
DC/UCF System Symbols.....	752
Address Symbols and Markers.....	752
User Symbols.....	753
Program Symbols	753
Syntax: Data Field Names	753
Syntax: Line Numbers.....	753
Syntax: Qualifying Program Symbols.....	753
Expression Operators	753
Delimiters	754
Debugger Commands	754
Syntax: AT	754
Syntax: DEBUG	755
Syntax: EXIT	755
Syntax: IOUSER	755
Syntax: LIST.....	755
Syntax: MENU	755
Syntax: PROMPT	755
Syntax: QUALIFY	756
Syntax: QUIT.....	756
Syntax: RESUME	756
Syntax: SET	756
Syntax: SNAP	756
Syntax: WHERE	757

Index 759

Chapter 1: Introduction to CA ADS

This section contains the following topics:

[Introduction](#) (see page 19)

[Syntax Diagram Conventions](#) (see page 19)

[What is CA ADS?](#) (see page 21)

[What CA ADS Does](#) (see page 22)

[Creating a CA ADS Application](#) (see page 23)

[Tools Used To Develop an Application](#) (see page 24)

[CA ADS Screens](#) (see page 31)

[Checkout and Release Procedures](#) (see page 44)

[CA ADS Help Facility](#) (see page 48)

Introduction

This guide is a reference for CA ADS for CA IDMS development tools and facilities. It provides reference information for application developers defining online and batch applications.

Syntax Diagram Conventions

The syntax diagrams presented in this guide use the following notation conventions:

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.

italicized lowercase

Represents a value that you supply.

lowercase bold

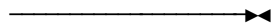
Represents a portion of the syntax shown in greater detail at the end of the syntax or elsewhere in the document.

←

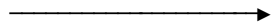
Points to the default in a list of choices.

▶—————▶

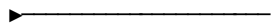
Indicates the beginning of a complete piece of syntax.



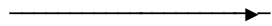
Indicates the end of a complete piece of syntax.



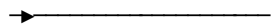
Indicates that the syntax continues on the next line.



Indicates that the syntax continues on this line.



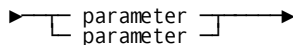
Indicates that the parameter continues on the next line.



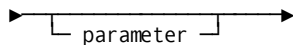
Indicates that a parameter continues on this line.



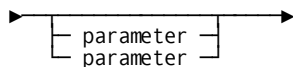
Indicates a required parameter.



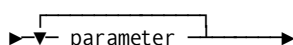
Indicates a choice of required parameters. You must select one.



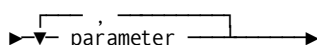
Indicates an optional parameter.



Indicates a choice of optional parameters. Select one or none.



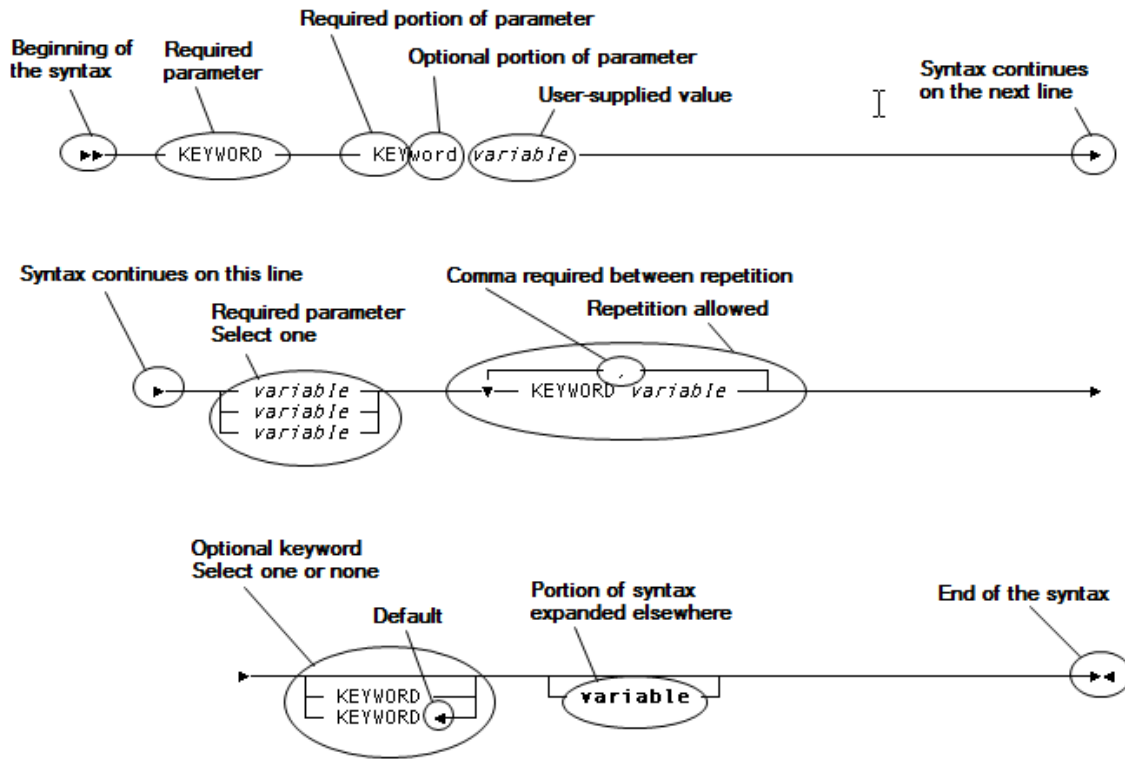
Indicates that you can repeat the parameter or specify more than one parameter.



Indicates that you must enter a comma between repetitions of the parameter.

Sample Syntax Diagram

The following sample explains how the notation conventions are used:



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.

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 CA 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

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.

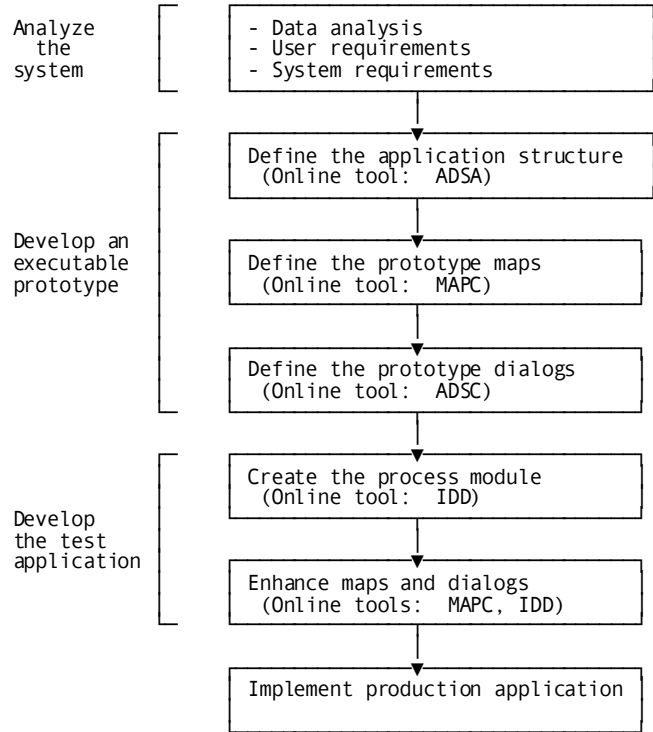
Procedure

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 following diagram 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



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.

Note: For more information about the transfer control facility, see the *CA IDMS Common Facilities Guide*.

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

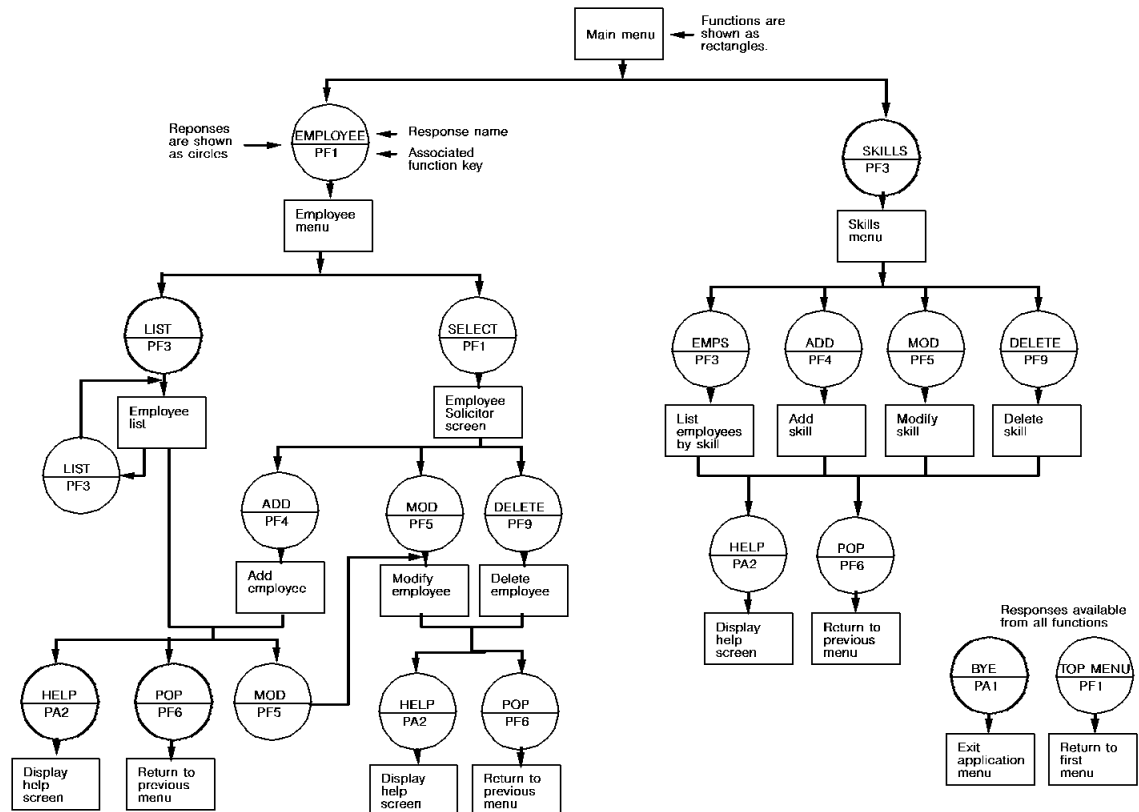
Function Type	What it Does
Dialog	Performs a variety of processing activities, such as data retrieval and update
Program	Performs processing specified in user-written COBOL, PL/I, or Assembler programs
Menu	Displays a system-defined menu screen Performs standard menu processing activities at runtime
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

Function Type	What it Does
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 of CA IDMS/DC or DC/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



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.

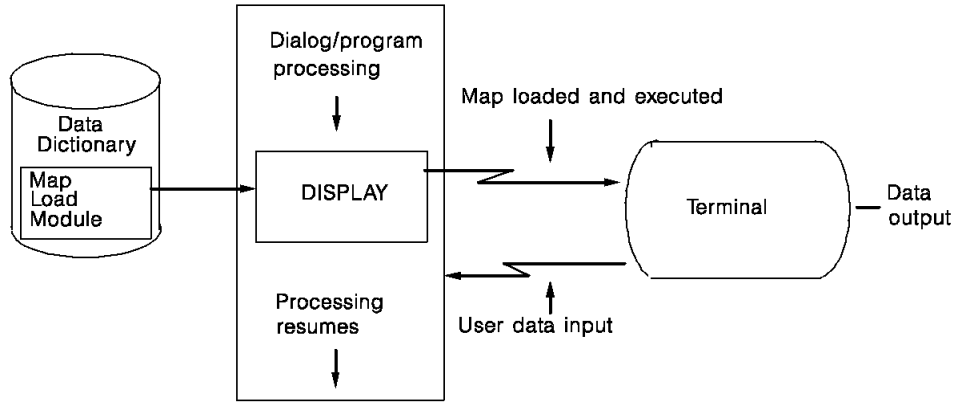
Note: For more information about the online mapping facility and the batch compiler and utility, see the *CA IDMS Mapping Facility Guide*.

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 following diagram 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



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.

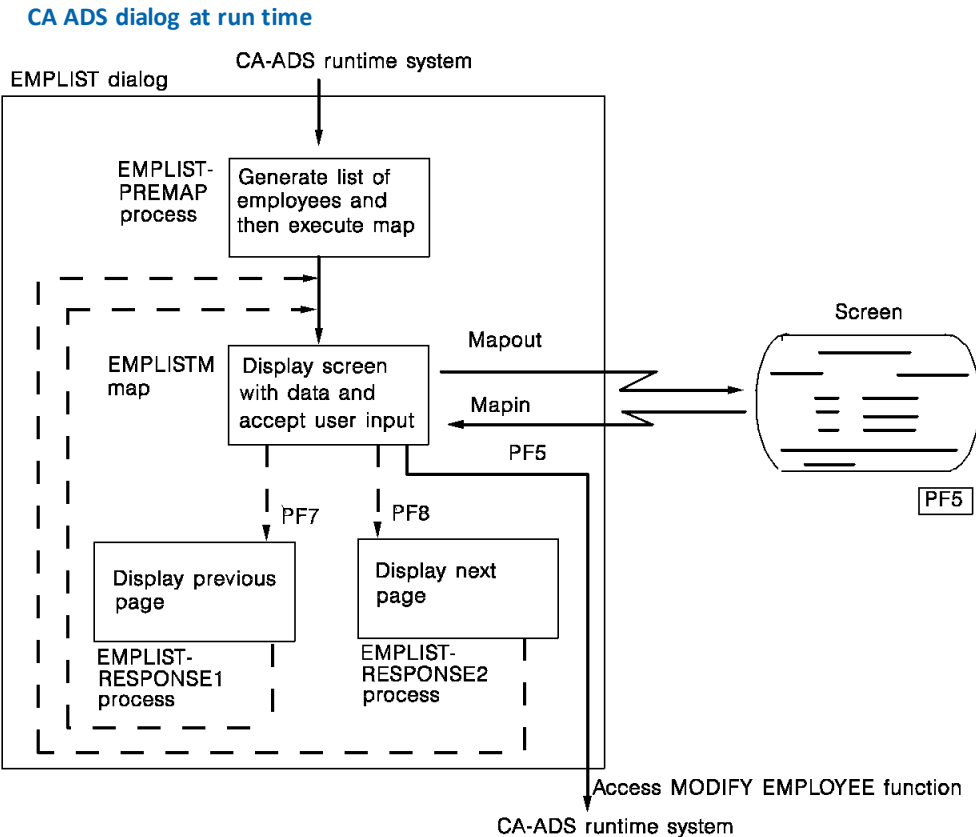
Components of a dialog

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 the <i>CA IDMS SQL Programming Guide</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 following diagram 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.



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 CAADS 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.

Note: For more information about how to use the IDD menu facility and online IDD screens, see the *CA IDMS Common Facilities Guide*. For more information about online IDD and DDDL statements, see the *CA IDMS IDD Quick Reference Guide*.

The CA ADS Runtime System

CA IDMS/DC and DC/UCF

The CA ADS runtime system is a CA IDMS/DC or DC/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.

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 following screen and described in the following table.

```

Add Modify Compile Delete Display Switch
-----
CA ADS OnLine Dialog Compiler
CA, 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) 2007 CA. ALL RIGHTS RESERVED

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).

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

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 pull-down 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 pull-down window being opened.

```

Add Modify Compile Delete Display Switch
-----
Copy from dialog | CA ADS Online Dialog Compiler
Name             |
Version 1        | CA, 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]

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.

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 following screen 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          CA, 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.

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

Add		Modify	Compile	Delete	Display	Switch
1		1. Checkout	Online Dialog Compiler			
		2. Release	A, INC.			
		3. List Checkouts	-----			
		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 following screen shows the **Compile** action on the dialog compiler Main Menu.

```

Add  Modify  Compile  Delete  Display  Switch
-----
          1 1. Compile          log Compiler
          2. Display messages
          -----
          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 following screen 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                                1 Commands
Version 0001 Type 2                            1 Errors
Key _____ Value                          _ 1. Display
                                           _ 2. Print
Name _____                                Commands
Version _____ Type _                       Errors
Key _____ Value                          _ 1. Display
                                           _ 2. Print
Name _____                                Commands
Version _____ Type _                       Errors
Key _____ Value                          _ 1. Display
                                           _ 2. Print
Name _____                                Commands
Version _____ Type _                       Errors
Key _____ Value                          _ 1. Display
                                           _ 2. Print

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.

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.

Note: For more information about the error messages used by CA ADS, see the *CA IDMS Messages and Codes Guide*.

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, see the *CA IDMS Common Facilities Guide*. For more information about correcting errors in process code, see 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 following screen 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.

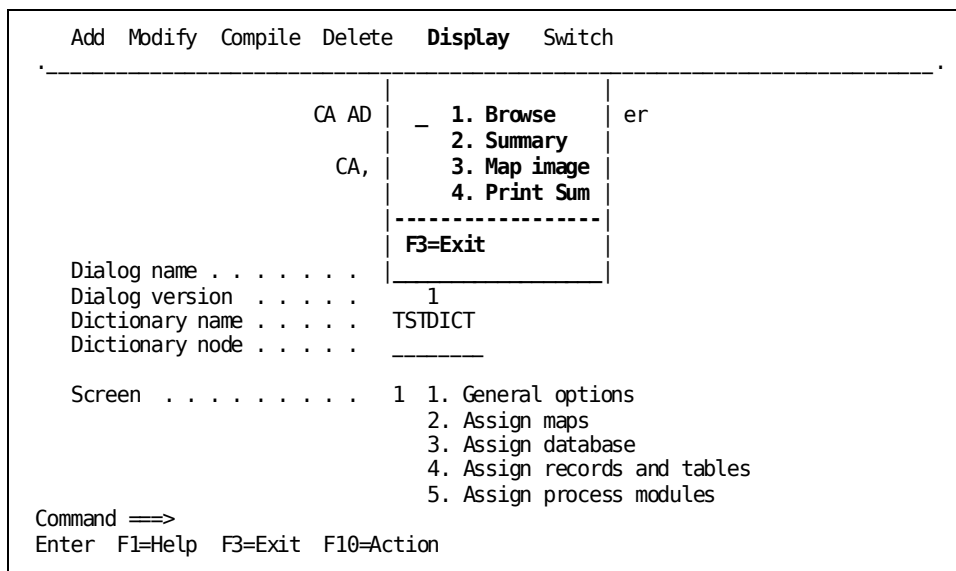
Command ==>
Enter F1=Help F3=Exit F10=Action

```

Display

Specifies that summary information for the named entity be displayed.

The following screen 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 following screen 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: JMASQL 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 DC/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 following screen 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	

More information:

- [Checkout and Release Procedures](#) (see page 44)
- [Introduction to Process Commands](#) (see page 295)

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:

DC498107 to user MET on dictionary TESTDCT.

If ADSOBCOM:

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

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 pull down menu.
- A CHECKOUT action from the MODIFY pull down 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 pull down 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
```

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 nn.n                                volser
                                         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.

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 nn.n                                volser
                                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) 2007 CA. ALL RIGHTS RESERVED
                                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.

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.

Chapter 2: CA ADS Application Compiler (ADSA)

This section contains the following topics:

[Overview](#) (see page 51)

[Application Compiler Session](#) (see page 51)

[Application Compiler Screens](#) (see page 57)

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.

Note: For more information about application compiling features, see the *CA ADS Application Design Guide*. For examples of using the application compiler, see the *CA ADS User Guide*.

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.

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 DC/UCF

By specifying the appropriate CA IDMS/DC or DC/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.

Note: 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 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 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 DC/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.

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. For more information about the transfer control facility, see the *CA IDMS Common Facilities Guide*.

TCF Selection Screen

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

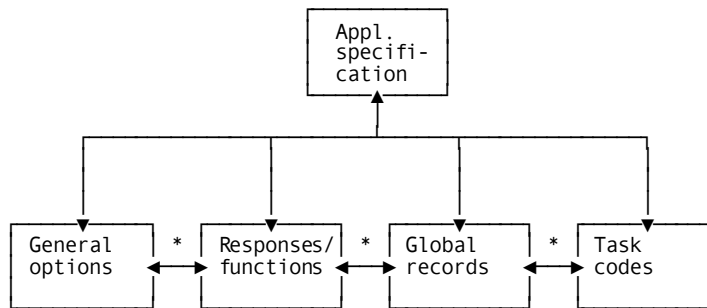
TRANSFER CONTROL FACILITY		CA, INC.		*** SELECTION SCREEN ***	
-	SUSPEND TCF SESSION (PF9)	DBNAME..:		DBNODE..:	
-	TERMINATE TCF SESSION (PF3)	DICTNAME:	TSTDICT	DICTNODE:	
TCF TASKCODES			*SUSPENDED SESSIONS*		
SELECT ONE TO START A NEW SESSION			SELECT ONE TO RESUME AN OLD SESSION		
-	TCF		TASKCODE		DESCRIPTOR
-	SYSGENT	SYSGEN COMPILER	_ADSCT		MPKDIA1 0001
-	MAPCT	MAP DEFINITION	_ADSAT		MPKAPP1 001
-	ADSCT	DIALOG GENERATOR	_ADSAT		MPKAPP1 002
X	ADSAT	APPLICATION GENERATOR	_ADSCT		MPKDIA2 0001
-	ASF				
-	ASFT				
-	IDDT	IDD COMMAND MODE			
-	SSCT	SUBSCHEMA COMPILER			
-	SCHEMAT	SCHEMA COMPILER			
-	IDDMT	IDD MENU MODE			
-	OLQ	OLQ COMMAND MODE			
-	OLQT	OLQ COMMAND MODE			

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)

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)

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

Step in process	Screen	Purpose
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

Default Control Keys

Activity	Control key	Description
HELP	[PF1]	<p>Displays a map or field help screen, depending on cursor position</p> <p>If the cursor is on a map field associated with help text, a half screen of map field help text is displayed.</p> <p>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.</p>

Activity	Control key	Description
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

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 [Invoking the Application Compiler](#) (see page 52).

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.

Note: For an example of an application compiler session, see the *CA ADS User Guide*.

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.

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, to name an application and a dictionary, and to 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].

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

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].

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 DC/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.

More information:

[Sequencing Through Application Compiler Screens](#) (see page 53)

[Introduction to CA ADS](#) (see page 19)

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 3. yy/mm/dd	2. dd/mm/yy 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.

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, 07/25/04). MM/DD/YY is the default.
- **DD/MM/YY** (for example, 25/07/04)
- **YY/MM/DD** (for example, 04/07/25)
- **DDD/YY** (for example, 207/04)

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.

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.

More information:

[System Records](#) (see page 567)

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 security has been enabled.

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.

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 ADAPG0P2. Only application developers having execution authority for ADAPG0P2 would have access to the second page of the General Options screen.

Note: For more information about CA IDMS central security, see the *CA IDMS Security Administration Guide*.

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.

More information:

[Security Features](#) (see page 717)

[System-Defined Menu Maps](#) (see page 126)

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.

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 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.

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 nn.n                               Page 1 of 2
                                Response/Function List
                                volser
Application name: METAPPL1  Version: 1
Select (/)  Response name  Assigned key  Select (/)  Function name/type(1,2,3)*  Program/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

                                * 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 nn.n                               Response/Function List           volser
                                           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
    
```

More information:

- [Response Definition Screen](#) (see page 70)
- [Function Definition \(Dialog\) Screen](#) (see page 74)
- [Function Definition \(Program\) Screen](#) (see page 77)
- [Function Definition \(Menu\) Screen](#) (see page 80)
- [CA ADS Dialog Compiler \(ADSC\)](#) (see page 91)

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. CAADS 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.

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.

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 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.
- 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.

More information:

[CA ADS Runtime System](#) (see page 119)

[Response Security](#) (see page 719)

[Control Commands](#) (see page 325)

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	Drop function (/) _	
Function name:	F1				
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
```

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.

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.

More information:

[System Records](#) (see page 567)

[CA ADS Runtime System](#) (see page 119)

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.

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.

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.

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.

More information:

[CA ADS Runtime System](#) (see page 119)

[Control Commands](#) (see page 325)

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.

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
Function name:	MENU1	Drop function (/) _
Description . . .	UNDEFINED	
Associated dialog	_____	User exit dialog _____
Default response	_____	
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.

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.

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.

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:

- specifies that the menu is user-defined, meaning that the user can specify the number of responses per page, from 0 to 50.
- (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:

- (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.
- 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.

More information:

[System Records](#) (see page 567)

[System-Defined Menu Maps](#) (see page 126)

[CA ADS Runtime System](#) (see page 119)

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.

Note: For more information about associating records with maps, see the *CA IDMS Mapping Facility Guide*.

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.

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.

More information:

[System Records](#) (see page 567)

[CA ADS Dialog Compiler \(ADSC\)](#) (see page 91)

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.

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 about the TASK statement, see the *CA IDMS System Generation Guide*.

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.

More information:

[CA ADS Runtime System](#) (see page 119)

Chapter 3: CA ADS Dialog Compiler (ADSC)

This section contains the following topics:

[Overview](#) (see page 91)

[Dialog Compiler Session](#) (see page 92)

[Dialog Compiler Screens](#) (see page 98)

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.

Note: For more information about modifying dialogs, see the *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.

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 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.

More information:

[CA ADS Application Compiler \(ADSA\)](#) (see page 51)

[Application and Dialog Utilities](#) (see page 621)

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.

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 DC/UCF

By specifying the appropriate CA IDMS/DC or DC/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, ADSCF.

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.

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.

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. For more information about the transfer control facility, see the *CA IDMS Common Facilities Guide*.

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

```

                                CA, 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         CEXME220001
    
```

More information:

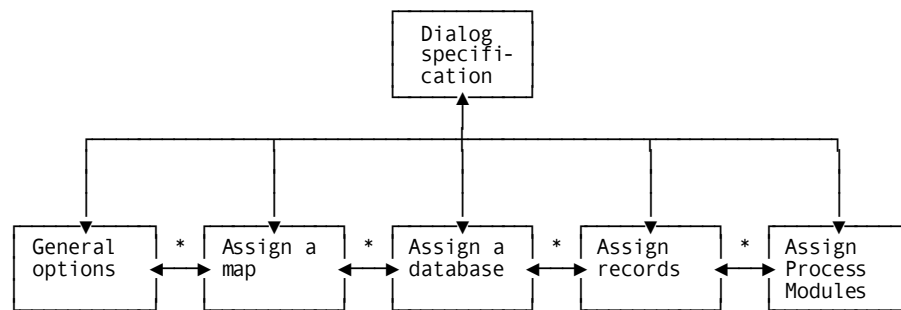
[Dialog Compiler Screens](#) (see page 98)

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)

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)

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 and 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

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.

Activity	Control key	Description
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

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.

More information:

[Invoking the Dialog Compiler](#) (see page 92)

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].

Dialog Compiler Screens

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].

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

Message Area

Displays informational and error messages returned from the dialog compiler.

Note: 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.

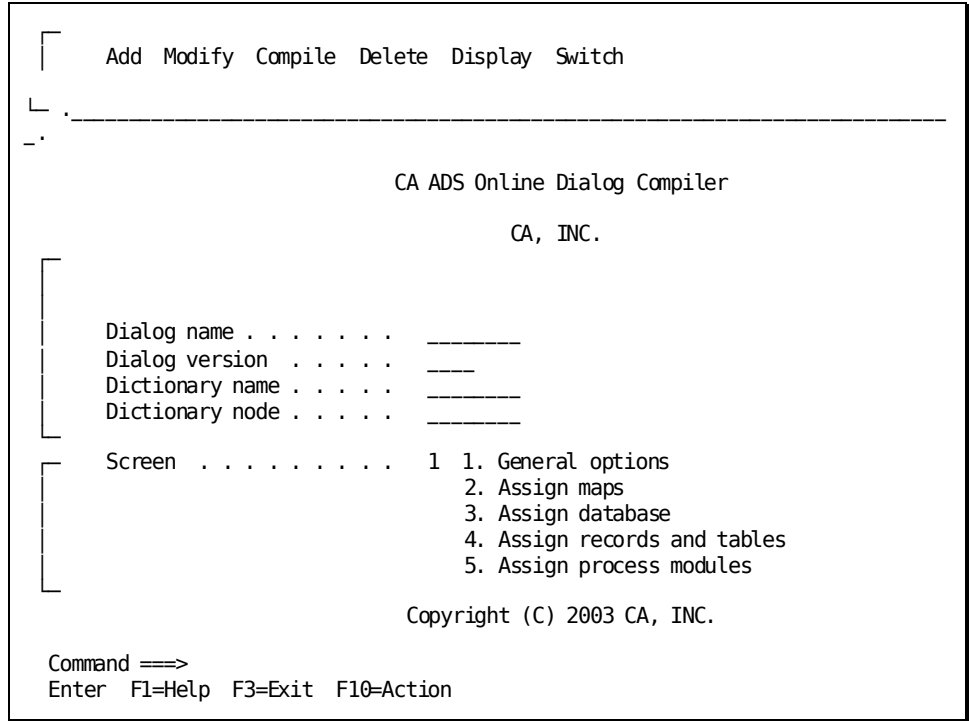
If an error is detected after the application developer selects an activity, the dialog compiler redisplay the current screen. The activity selection is retained and executed when the error is corrected. The application developer can override the initial selection by specifying another activity on the 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



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.

More information:

[Introduction to CA ADS](#) (see page 19)

[Sequencing Through Dialog Compiler Screens](#) (see page 94)

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 . . . . . ADS0-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.

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.

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.

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.

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.

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.

Note: For more information about the COBOL MOVE subclause of the system generation ADSO statement, see the *CA IDMS System Generation Guide*.

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.

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

Note: For more information about the system generation ADSO statement, see the *CA IDMS System Generation Guide*.

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.

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.

More information:

[CA ADS Runtime System](#) (see page 119)

[Control Commands](#) (see page 325)

[Error Handling](#) (see page 277)

[Debugging an CA ADS Dialog](#) (see page 723)

[Assignment Command](#) (see page 314)

[Activity Logging for an CA ADS Dialog](#) (see page 675)

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 JPKTD01  Version  1

Map name . . . . . _____  Input map . . . . . _____
Version . . . . . _____  Version . . . . . _____
                               Label . . . . . _____

Paging options  - 1. Wait
                  2. No Wait
                  3. Return

Paging mode . . . _ Update
                  _ Backpage
                  _ Auto display

                               Suspense file label _____

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.

No Wait is the default for pageable maps.

Note: The map paging dialog options **No Wait** 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. **Update** is the default for pageable maps.

Note: The map paging dialog options **No Wait** and **Update** cannot be specified together. 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. **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.

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 z/OS ddname (z/VSE filename and z/VM 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 z/OS ddname (z/VSE filename, z/VM 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 z/OS ddname (z/VSE filename, z/VM 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.

More information:

[Pageable Maps](#) (see page 464)

[Conditional Expressions](#) (see page 245)

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.

Note: For more information on creating and executing access modules, see:

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

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.

Note: For more information on SQL standards, see the *CA IDMS SQL Reference Guide*.

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

Note: For more information on date/time representations, see the *CA IDMS SQL Reference Guide*.

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.

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.

Note: For more information on automatic editing, see the *CA IDMS Mapping Facility Guide*.

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.

Note: For more information about the usage for WHENEVER and DECLARE CURSOR, see the *CA IDMS SQL Programming Guide*. For more information on the declaration module, see the *CA IDMS SQL Programming Guide*.

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.

Sample Screen

Process Modules		Page	1 of	1
Name	Dialog NAME1	Version	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.

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.

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.

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.

More information:

[CA ADS Screens](#) (see page 31)

[Introduction to CA ADS](#) (see page 19)

[CA ADS Runtime System](#) (see page 119)

[Suspending a Session](#) (see page 97)

[Conditional Expressions](#) (see page 245)

[Conditional Commands](#) (see page 317)

Chapter 4: CA ADS Runtime System

This section contains the following topics:

[Initiating the CA ADS Runtime System](#) (see page 119)

[Runtime Menu and Help Screens](#) (see page 124)

[Runtime Flow Of Control](#) (see page 135)

[Message Prefixes](#) (see page 140)

[CA ADS Tasks, Run Units, and Transactions](#) (see page 141)

[Dialog Abort Information Screen](#) (see page 145)

[Debugging a Dialog](#) (see page 148)

[Linking From CA ADS To CA OLQ](#) (see page 149)

[Linking Built-In Functions With The Runtime System](#) (see page 150)

[Managing Storage](#) (see page 151)

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.

Note: For more information about the CA ADS Batch runtime system, see the *CA ADS Batch User Guide*.

How to Define Runtime Tasks

Tasks that initiate the CA ADS runtime system are defined at CA IDMS/DC or DC/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. 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.

Note: For more information on the system generation TASK statement, see the *CA IDMS System Generation Guide*.

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.

More information:

[CA ADS Application Compiler \(ADSA\)](#) (see page 51)

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.

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

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.

Dialog Selection Screen

```

CA ADS RELEASE nn.n                                     PAGE:
  1
          DIALOG SELECTION FOR USER:

ENTER DIALOG NAME:          OR SELECT ONE BELOW

  _ ADSA          _ ADSOTATU          _ ASFADSGØ          _ ASFOOAKD
-  OLQ           -  RQERDQ

PA1 - PAGE FORWARD
PA2 - PAGE BACK
CLEAR - EXIT CA ADS
    
```

The user initiates a dialog from the Dialog Selection screen by entering a nonblank character in the response field corresponding to the applicable dialog, or by entering the name of the dialog in response to the ENTER DIALOG NAME prompt.

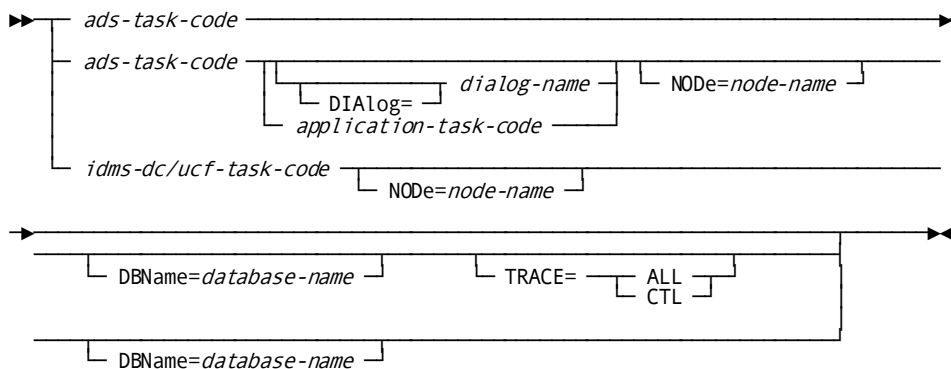
Initiating the CA ADS Runtime System

Syntax and syntax rules for the statement used to initiate the CA ADS runtime system are shown below.

Use these statements in response to the DC/UCF prompt:

ENTER NEXT TASK CODE

Syntax



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

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:

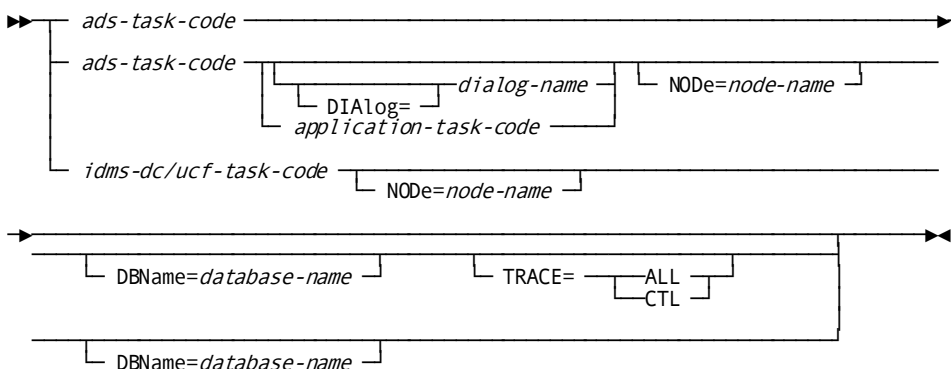
```
ADS TESTAPPL DBNAME=TESTDB
```

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.

More information:

- [CA ADS Application Compiler \(ADSA\)](#) (see page 51)
- [Application and Dialog Utilities](#) (see page 621)

Runtime System Initiating Statement



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.

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.

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.

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.

The CA ADS runtime system processes system-defined and site-defined menu maps in the same way, as described below.

More information:

[CA ADS Application Compiler \(ADSA\)](#) (see page 51)

[System Records](#) (see page 567)

[Runtime Flow Of Control](#) (see page 135)

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 [System Records](#) (see page 567).
- 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.

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.

Note: For more information on site-defined menu maps, see the *CA ADS Application Design Guide*.

More information:

[CA ADS Application Compiler \(ADSA\)](#) (see page 51)

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.

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.

Sample Short Description Menu Screen (ADSOMUR1)

DIALOG:		PAGE: 1 OF: 1	
DATE: mm/dd/yy		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

Sample Long Description Menu Screen (ADSOMUR2)

DIALOG:		PAGE: 1 OF: 1
DATE: mm/dd/yy		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.

Signon menu maps can be site-defined or system-defined.

Sample Signon Menu Screen (ADSOMSON)

```

DIALOG:                                     PAGE: 1 OF: 1
DATE: mm/dd/yy                             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

Specifies 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.

More information:

[CA ADS Application Compiler \(ADSA\)](#) (see page 51)

[System Records](#) (see page 567)

[Security Features](#) (see page 717)

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: mm/dd/yy	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.

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:

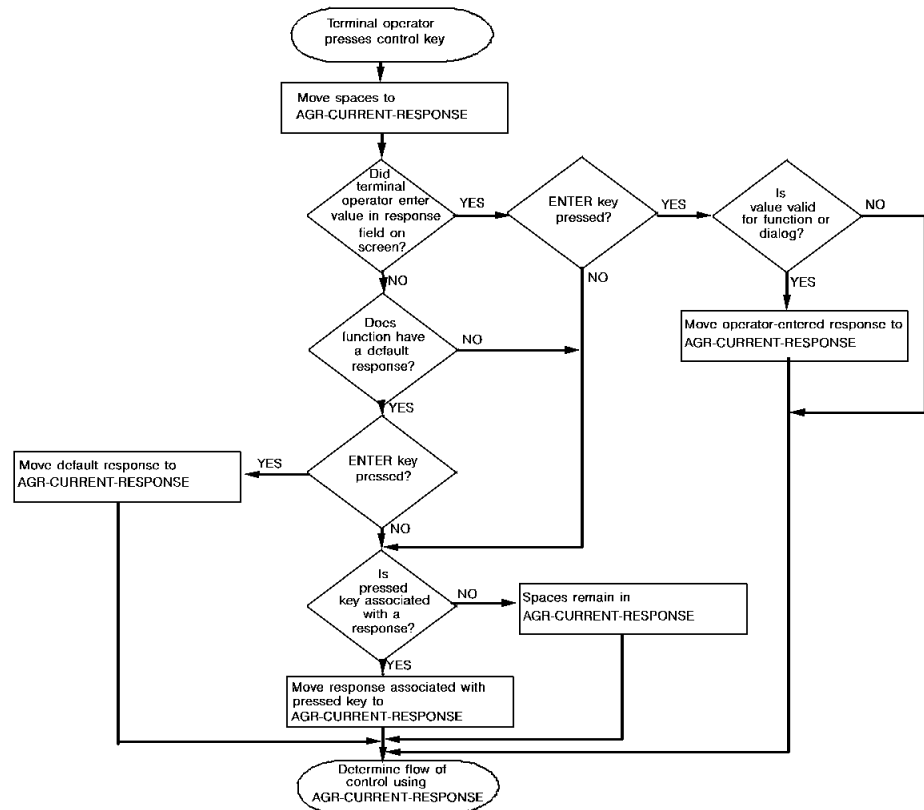
1. The runtime system moves spaces to AGR-CURRENT-RESPONSE.
2. The runtime system checks the AGR-MAP-RESPONSE field of ADSO-APPLICATION-GLOBAL-RECORD (AMR-RESPONSE-FIELD of ADSO-APPLICATION-MENU-RECORD for menu functions) for a response entered by the user. If the user entered a response and pressed 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.

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.

Establishing the Value of AGR-CURRENT-RESPONSE



Valid Response

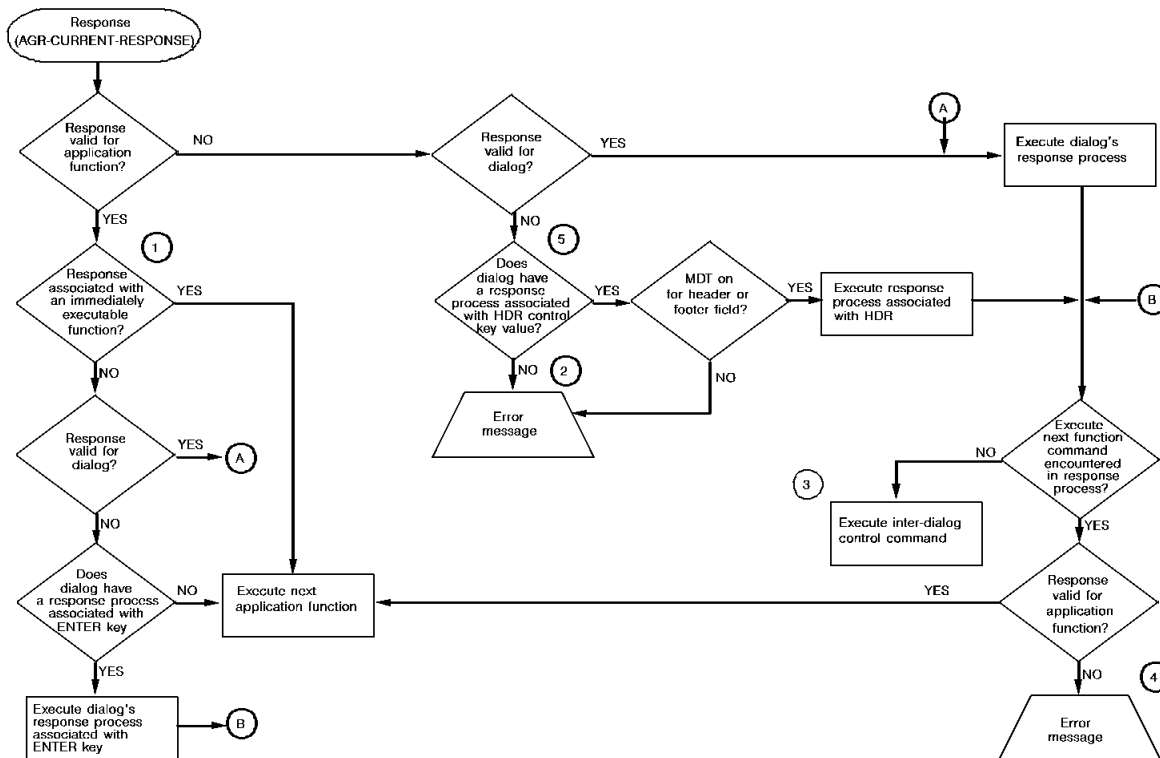
If the response established in AGR-CURRENT-RESPONSE is valid for the current function, the runtime system moves the name of the function associated with the response to the AGR-NEXT-FUNCTION field of ADSO-APPLICATION-GLOBAL-RECORD.

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 certain criteria. The following diagram shows how the flow of control is directed within an application at runtime.

Application Flow of Control



Notes:

1. Immediately executable functions are HELP, SIGNON, SIGNOFF.
2. Message displayed on user's screen:
UNACCEPTABLE RESPONSE. PLEASE TRY AGAIN.
3. Inter-dialog control commands are DISPLAY, INVOKE, LEAVE, LINK, RETURN, and TRANSFER.
4. Message displayed on user's screen:
INVALID RESPONSE SPECIFIED BY DIALOG PROCESS CODE
5. HDR can be specified only in a dialog associated with a pageable map.

Default Control Key Assignments

At system generation, **default control key assignments** can be specified for certain formats of the LEAVE and RETURN process commands. By default, PA1 is assigned to LEAVE APPLICATION; CLEAR is assigned to RETURN CLEAR. At runtime, the user can press these keys to perform their associated commands.

Note: For more information on defining default control key assignments, see the discussion of the KEYS statement in the *CA IDMS System Generation Guide*.

Default control key assignments are overridden by control key assignments specified for application responses and dialog response processes.

More information:

[System Records](#) (see page 567)

[Control Commands](#) (see page 325)

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.

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.

More information:

[CA ADS Dialog Compiler \(ADSC\)](#) (see page 91)

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.

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 [Initiating the CA ADS Runtime System](#) (see page 119). 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.

Note: For more information on transactions within an SQL session, see the *CA IDMS SQL Programming Guide*. For a list of SQL statements that start and end a database transaction, see the *CA IDMS SQL Programming Guide*.

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.

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.

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.

■ Adding an area to a dialog

It is possible that the extended run unit will no longer be extended, if:

- You add an area to one of the dialogs in an application thread (for example, when the record is migrated into a different area).

and

- Any of the dialogs are recompiled.

A run unit that is no longer extended can have a serious impact on handling a potential ROLLBACK or abend. If the run unit is no longer extended, then recovery can be incomplete and can disrupt the integrity of the database. If such a change to the subschema occurs, we recommend comparing the original application run unit structure to the modified application run unit structure. For this comparison, use, for example, ADS TRACE=CTL or Journal Reports.

■ Using default usage mode with the option FORCE

We recommend not to use the subschema FORCE option for ADS applications with extended run units. Change the dialog code instead.

Several problems can occur if you use the subschema FORCE option for ADS applications with extended run units. For example, if the dialog that needs a new area is the recipient of an extended run unit, the FORCE area is not READIED.

Note: For more information on the limitations of using the FORCE option with ADS dialogs, see the Area Statement section (in the Subschema Statements chapter) in the *CA IDMS Database Administration Guide*.

■ 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.

More information:

[Control Commands](#) (see page 325)

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.

Note: For more information on the ADSO statement, see the *CA IDMS System Generation Guide*.

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 nn.n          *** DIALOG ABORT INFORMATION ***  ABRT
DC173008 APPLICATION ABORTED. BAD IDMS STATUS RETURNED; STATUS=0306

DATE....: yy.ddm          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).

PROCESS

Specifies the name of the premap or response process containing the command that caused theabend.

VERSION

Specifies the version number of the process containing the command that caused theabend.

IDD SEQ NUMBER

Specifies the data dictionary sequence number of the source line containing the command that caused theabend. The IDD sequence number is not displayed if the dialog was compiled without diagnostic tables.

SEQUENCE NUMBER

Specifies the internal command numbers of the source line containing the command that caused theabend and of the source lines immediately preceding and following it.

Internal command numbers are not displayed if the dialog was compiled without diagnostic tables.

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 theabend and of the source lines immediately preceding and following the command. Source lines are not displayed if the dialog was compiled without diagnostic tables.

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.

More information:

[CA ADS Dialog Compiler \(ADSC\)](#) (see page 91)

[CA ADS Dialog and Application Reporter](#) (see page 583)

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.

More information:

[Debugging an CA ADS Dialog](#) (see page 723)

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.

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 .
```

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.

Linking Built-In Functions With The Runtime System

All CA-supplied BIF modules are linked with ADSOMAIN and all user-written BIF modules can optionally be linked together as a new ADSOVCON module.

Note: Creating an ADSOVCON module is not required. CA ADS dynamically loads user-written BIFs if they are not linked with ADSOVCON.

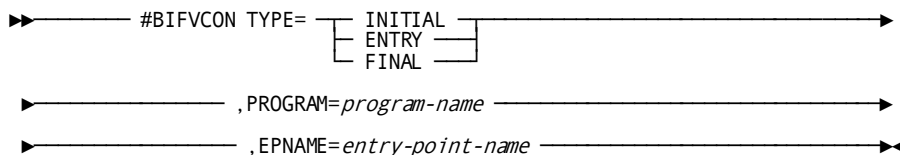
ADSOVCON Module Creation

Optionally create an ADSOVCON module using the #BIFVCON macro to identify your user-written BIF modules to be linked together. The following sample ADSOVCON module indicates that the UDATE and UCHECK BIF modules are to be linked with ADSOVCON.

```
#BIFVCON TYPE=INITIAL  
#BIFVCON TYPE=ENTRY , PROGRAM=UPDATE , EPNAME=UPDATE  
#BIFVCON TYPE=ENTRY , PROGRAM=UCHECK , EPNAME=UCHKEP1  
#BIFVCON TYPE=FINAL
```

To create an ADSOVCON module, create a source member as described in the following section and save it in your custom source library. Then assemble and link it into your custom load library.

The following diagram shows the syntax for the #BIFVCON macro:



#BIFVCON Macro Parameters

This section describes the parameters for for the #BIFVCON macro.

TYPE

Indicates the type of BIFVCON statement being generated.

INITIAL

Identifies the first BIFVCON statement in the program.

ENTRY

Identifies a BIFVCON statement defining a user-written BIF module.

FINAL

Identifies the last BIFVCON statement in the program.

PROGAM=*program-name*

Identifies the name of a user-written BIF module to be linked with ADSOVCON. *program-name* must be the same as the program name associated with a built-in function declared in your RHDCEVBF module.

This parameter is valid only if TYPE=ENTRY is coded.

EPNAME=*entry-point-name*

Identifies the name of the entry point of the user-written BIF module. *entry-point-name* must be the name of the entry point in the program identified by *program-name*.

This parameter is valid only if TYPE=ENTRY is coded.

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.

Note: For more information about storage management, see the *CA IDMS System Generation Guide*.

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.

Note: For more information, see the *CA IDMS System Generation Guide*.

At runtime, the current record compression setting can be changed by using the DCMT VARY ADSO command.

Note: For more information about this DCMT command, see the *CA IDMS System Tasks and Operator Commands Guide*.

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.

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.

Note: For more information about the ADSO and SYSTEM statements, see the *CA IDMS System Generation Guide*.

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:

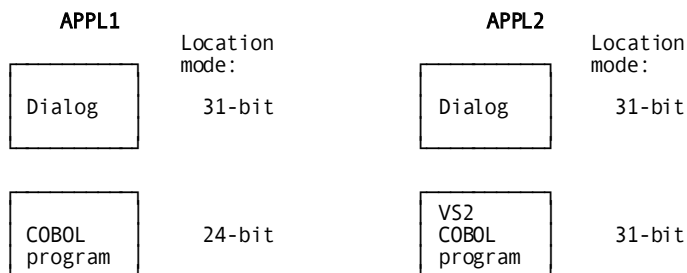
```
TASK ADS2 INVOKES ADSOMAIN LOCATION ANY.
```

- A task invoked after the return from a user program:

```
TASK ADS2R INVOKES ADSOMAIN LOCATION ANY.
```

Sample Task Definitions

The following diagram 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

This section contains the following topics:

[Overview](#) (see page 155)

[Process Modules](#) (see page 156)

[Process Commands](#) (see page 159)

[Data Types](#) (see page 161)

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.

Note: For more information about the usage for WHENEVER and DECLARE CURSOR, see the *CA IDMS SQL Reference Guide*. For further considerations regarding the declaration module, see the *CA IDMS SQL Programming Guide*.

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.

Note: For more information on defining data, see the *CA IDMS IDD Quick Reference Guide*. For more information on defining data in subschemas, see the *CA IDMS Database Administration Guide*.

More information:

[Data Types](#) (see page 161)

Process Modules

A process module is a discrete dialog unit that performs the processing operations required by a given dialog.

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.

Note: For more information on using the IDD menu facility in the CA ADS environment, see the *CA ADS User Guide*.

The online IDD PROCESS statement can also be used.

Note: For more information on IDD PROCESS statement syntax, see the *CA IDMS IDD Quick Reference Guide*.

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.

Note: For more information on using ADSC to associate process modules with dialogs, see the *CA ADS User Guide*.

More information:

[Application and Dialog Utilities](#) (see page 621)

Executing Process Modules

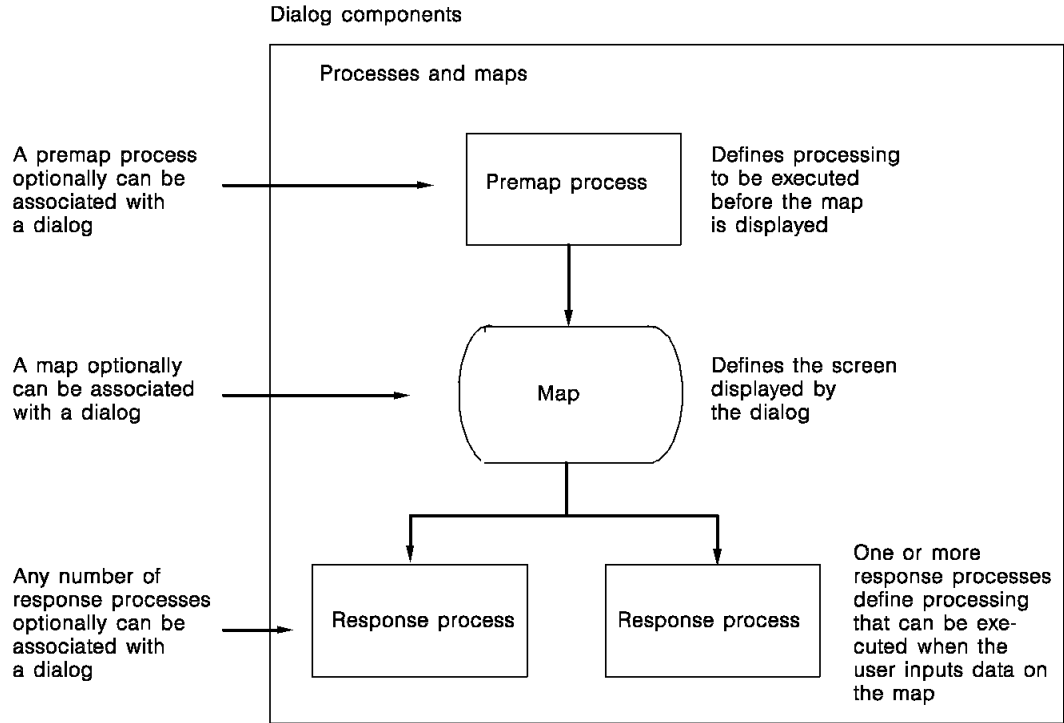
Process modules are executed before or after a dialog's map is displayed on the terminal screen (online applications) or are used to transfer input or output data (CA ADS Batch):

- 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

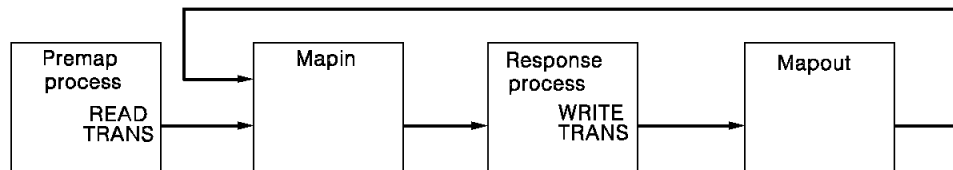
The following diagram shows the execution sequence of CA ADS premap and response process modules.

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 following diagram 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.



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.

Note: For more information about coding a declaration module, see the *CA IDMS SQL Programming Guide*.

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

Type of variable	Purpose
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.

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 placeholder 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.

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.

Note: CA ADS does not support edited data numeric fields. Therefore, PICTURE clauses on elements in records used in CA ADS dialogs cannot include edit characters such as \$, Z, period, comma, or +.

For more information about the correlation between CA ADS data types and SQL data types, see the *CA IDMS SQL Reference Guide* and the *CA IDMS SQL Programming Guide*.

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 short1	4 bytes ²	-64 to +63	USAGE IS COMPUTATIONAL-1 (No picture clause)
Internal long1	8 bytes ²	-64 to +63	USAGE IS COMPUTATIONAL-2 (No picture clause)
Display3	1 byte for each character	-64 to +63	PICTURE &plusmin.9V99E&plusmin.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


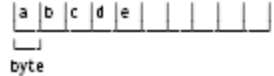
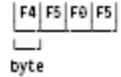
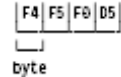

Zoned decimal numeric data fields occupy one byte of storage per digit. The sign of the number (hexadecimal C for positive, D for negative, F for unsigned positive) is stored in the four high-order bits of the rightmost digit. S is used only when the field is signed.


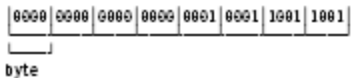
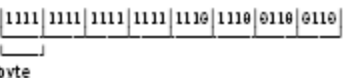
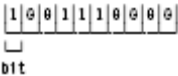
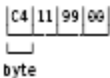
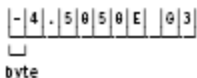
The following table describes the characteristics of a zoned decimal data field and how the field is defined in the data dictionary.

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 PICX(10). Internal representation: ²	MOVE 'abcde' TO EMPLOYEE. 
EBCDIC	FIELD-1 PICX(10). Internal representation:	MOVE 'abcde' TO EMPLOYEE. 
Zoned decimal	AMT-1 PIC9(4). Internal representation:	MOVE 4505 TO AMT-1. 
Zoned decimal	AMT-1 PICS9(4). Internal representation:	MOVE 4505 TO AMT-1. 
Packed decimal	AMT-1 PIC 9(4) USAGE COMP-3. Internal representation:	MOVE 4505 TO AMT-1. 

Type	Description ¹	Command
	AMT-1 PIC S9(4) USAGE COMP-3.	MOVE 4505 TO AMT-1.
	Internal representation:	
Binary	AMT-1 PIC S9(8) USAGE COMP.	MOVE 4505 TO AMT-1.
	Internal representation:	
	AMT-1 PIC S9(8) USAGE COMP.	MOVE -4505 TO AMT-1.
	Internal representation:	
Multitibit binary	FIELD-1 PICX(10) USAGE BIT.	MOVE B'10011100' TO FIELD-1.
	Internal representation:	
Floating point	AMT-1 USAGE COMP-1. (Internal short)	MOVE -45.05E02 TO AMT-1. (Value will be stored as -4505E04)
	Internal representation:	
	AMT-1 PIC S9.9999 E-99.	MOVE -45.05 E 02 TO AMT-1.
	Internal representation:	

Note:

¹ If no USAGE clause is provided, the default usage is DISPLAY.

² A blank space = a blank (X'40') in internal representations.

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 **	Multibit 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.

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.

More information:

[Options and Directives Screen](#) (see page 101)

Chapter 6: Arithmetic Expressions

This section contains the following topics:

[Overview](#) (see page 171)

[Syntax](#) (see page 171)

[Evaluation Of Arithmetic Expressions](#) (see page 173)

[Coding Considerations](#) (see page 174)

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.

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 [Built-in Functions](#) (see page 175).

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 [Variable Data Fields](#) (see page 285)

arithmetic-expression

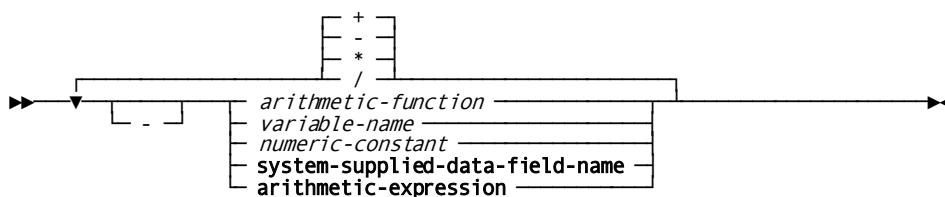
An arithmetic expression. Use parentheses to control the order in which operations are to be performed.

+ - * /

The arithmetic operators:

Operator	What it does
+	Addition
-	Subtraction
*	Multiplication
/	Division

Syntax: Arithmetic-Expression



Evaluation Of Arithmetic Expressions

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.

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

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

This section contains the following topics:

- [Overview](#) (see page 177)
- [User-Defined Built-In Functions](#) (see page 179)
- [System-Supplied Functions](#) (see page 179)
- [ABSOLUTE-VALUE](#) (see page 185)
- [ARC COSINE](#) (see page 186)
- [ARC SINE](#) (see page 187)
- [ARC TANGENT](#) (see page 188)
- [CONCATENATE](#) (see page 189)
- [COSINE](#) (see page 190)
- [DATECHG](#) (see page 191)
- [DATEDIF](#) (see page 194)
- [DATEEXT](#) (see page 196)
- [DATEINT](#) (see page 196)
- [DATEOFF](#) (see page 197)
- [DATETIMX](#) (see page 199)
- [DISPDT](#) (see page 199)
- [DTINT](#) (see page 200)
- [EXTRACT](#) (see page 201)
- [FIX](#) (see page 202)
- [GOODDATE](#) (see page 203)
- [GOODTRAILING](#) (see page 204)
- [INITCAP](#) (see page 205)
- [INSERT](#) (see page 206)
- [INVERT-SIGN](#) (see page 207)
- [LEFT-JUSTIFY](#) (see page 208)
- [LIKE](#) (see page 209)
- [LOGARITHM](#) (see page 210)
- [MODULO](#) (see page 211)
- [NEXT-INT-EQHI](#) (see page 212)
- [NEXT-INT-EQLO](#) (see page 213)
- [NUMERIC](#) (see page 214)
- [RANDOM-NUMBER](#) (see page 216)
- [REPLACE](#) (see page 218)
- [RIGHT-JUSTIFY](#) (see page 219)
- [SIGN-VALUE](#) (see page 220)
- [SINE](#) (see page 220)
- [SOCKET](#) (see page 221)
- [SQUARE-ROOT](#) (see page 223)
- [STRING-INDEX](#) (see page 223)
- [STRING-LENGTH](#) (see page 224)
- [STRING-REPEAT](#) (see page 225)
- [SUBSTRING](#) (see page 226)
- [TANGENT](#) (see page 228)
- [TIMEEXT](#) (see page 229)
- [TIMEINT](#) (see page 230)
- [TODAY](#) (see page 231)
- [TOLOWER](#) (see page 232)

[TOMORROW](#) (see page 233)
[TOUPPER](#) (see page 234)
[TRAILING-TO-ZONED](#) (see page 235)
[TRANSLATE](#) (see page 236)
[VERIFY](#) (see page 237)
[WEEKDAY](#) (see page 238)
[WORDCAP](#) (see page 240)
[YESTERDAY](#) (see page 241)
[ZONED-TO-TRAILING](#) (see page 242)

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.
- **User-defined functions** that perform site-specific functions defined by the installation.

User-defined and system-defined functions are described below, after a discussion of general considerations that apply to both types of built-in functions.

More information:

[Changing Invocation Names](#) (see page 713)
[Creating User-Defined Built-In Functions](#) (see page 714)

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).

More information:

[Changing Invocation Names](#) (see page 713)

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.

More information:

[Conversion Between Data Types](#) (see page 167)

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.

User-Defined Built-In Functions

User-defined built-in functions perform functions that are defined by individual sites.

More information:

[Built-in Function Support](#) (see page 681)

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.

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

Function	Keyword	What it does
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

Date Functions

Date built-in functions perform date processing in eight formats:

- **Gregorian**— The first format is *yyymmdd*, 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 *mmddyy*, 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

Date-Time Stamp Functions

Date-time stamp built-in functions convert external date-time stamps to internal date-time stamps. Conversely, internal date-time stamps can be converted to external date-time stamps. The date-time stamp built-in functions call the date-time functions of IDMSIN01.

Note: For more information about IDMSIN01, see the *CA IDMS Callable Services Guide*.

In the following table, 8-byte binary fields are defined as PIC9(16) COMPUTATIONAL fields, while display fields are defined with PICX definitions.

Note: For more information about the date-time stamp formats used by the date-time stamp functions, see the chapter "Representation of Date/Time Values" in the *CA IDMS SQL Reference Guide*.

Function	Keyword	What it does
External Date	DATEEXT	Returns a 10-byte external date stamp as an 8-byte internal binary date stamp
Internal Date	DATEINT	Returns an 8-byte internal binary date stamp as a displayable 10-byte date stamp

Function	Keyword	What it does
Display Date Time	DISPDT	Returns the current date-time stamp as a 26-byte displayable date-time stamp
External Date-Time	DATETIMX	Returns a 26-byte external date-time stamp as an 8-byte internal binary date-time stamp
Internal Date-Time	DTINT	Returns an 8-byte internal date-time stamp as a 26-byte displayable date-time stamp
External Time	TIMEEXT	Returns an 8-byte displayable time stamp as an 8-byte binary time stamp
Internal Time	TIMEINT	Returns an 8-byte internal time stamp as a displayable 8-byte time stamp

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
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

Function	Keyword	What it does
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

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

Function	Keyword	What it does
Zoned to trailing	ZONED-TO-TRAILING	Returns a COBOL trailing sign numeric from a zoned numeric

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

Function	Keyword	What it does
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
Tangent (radians)	TANGENT-RADIANS	Returns the tangent of a numeric value that represents an angle in radians

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 following example, 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'

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.
```

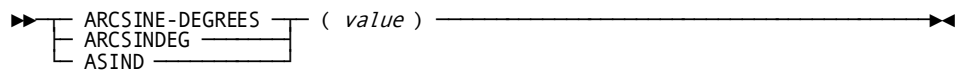
ARC SINE

Purpose

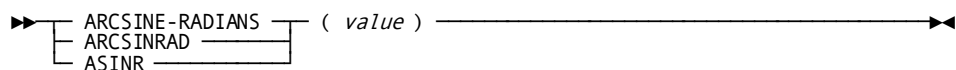
Returns the arc sine of a numeric value that represents an angle in either degrees or radians.

Syntax

Arc sine (degrees):



Arc sine (radians):



Parameters

ARCSINE-DEGREES

Returns an arc sine value in degrees.

ARCSINE-RADIANS

Returns an arc sine value in radians.

value

Specifies the numeric value representing the angle, in degrees or radians, whose arc sine 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 sine (degrees) of 0.8660 is calculated and moved to WK-RESULT (PIC S999V9999):

```
MOVE ASIND(0.8660) TO WK-RESULT.
Return value: 59.9971
```

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.
```

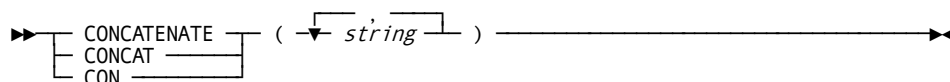
Return value: 60.0007

CONCATENATE

Purpose

Returns the concatenation of a specified list of string values.

Syntax



Parameters

string

Specifies one or more string values that are concatenated to form a single string value.

String can be:

- A string literal, enclosed in single quotation marks
- The name of a user-defined variable data field containing the string

Example 1: Using the concatenate function only

In the following example, the concatenate function is used to concatenate EMP-FNAME (PICX(15)) and EMP-LNAME (PIC X(15)) so that the first name precedes the last name:

Initial values:

```
EMP-FNAME: 'JOE          '
EMP-LNAME: 'SMITH        '
```

Statement:

```
MOVE CONCATENATE(EMP-FNAME,EMP-LNAME) TO WK-NAME.
```

Returned string:

```
'JOE          SMITH        '
```

Example 2: Using the concatenate function with the extract function

In this example, the concatenate function is used in conjunction with the extract function to concatenate EMP-FNAME (PIC X(15)), up to but not including the first blank, with a blank and then with EMP-LNAME (PIC X(15)):

```
Initial values:
  EMP-FNAME: 'JOE          '
  EMP-LNAME: 'SMITH          '
Statements:
  MOVE CON(EXTRACT(EMP-FNAME),' ',EMP-LNAME) TO WK-NAME.
Returned string:
  'JOE SMITH          '
```

Another example of the concatenate function is provided in [SUBSTRING](#) (see page 226).

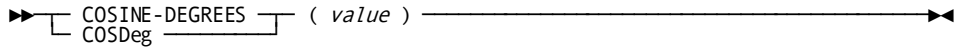
COSINE

Purpose

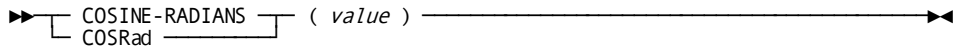
Returns the cosine of a numeric value that represents an angle in either degrees or radians.

Syntax

Cosine (degrees):



Cosine (radians):



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
```

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.

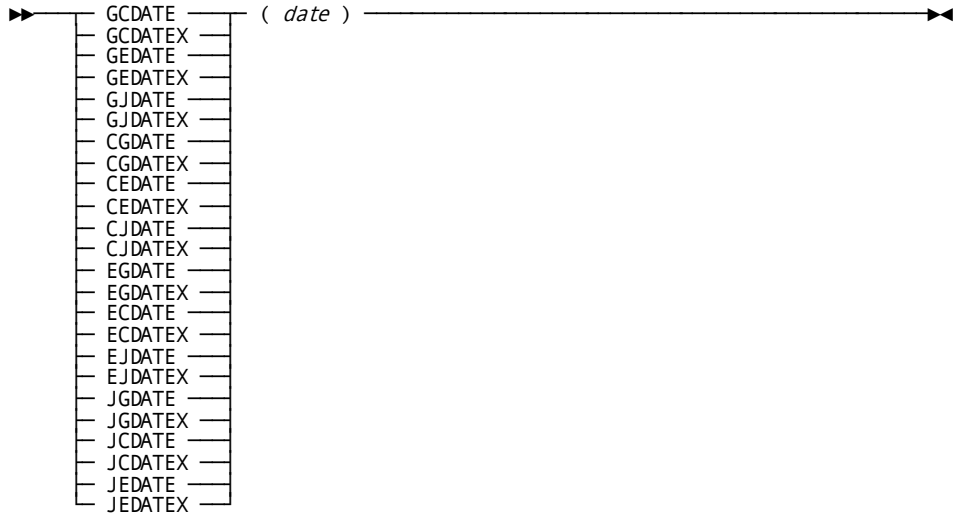
Syntax**Format 1:**

```

▶ ┌ DATECHG ───┐ ( date, input-date-format, output-date-format ) ───▶
  └ DATECHGX ─┘

```

Format 2:



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.
```

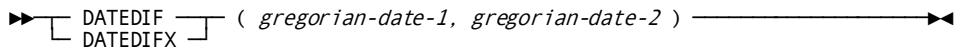
Returned value: 09121929

DATEDIF

Purpose

Returns the number of days between two specified dates.

Syntax



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

Example 1

In the following example, 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

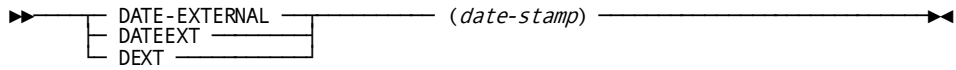
Again, if the dates were supplied in reverse order, the value -901 would have been returned.

DATEEXT

Purpose

Returns a 10-byte external date stamp as an 8-byte internal binary date stamp.

Syntax



Parameter

date-stamp

Specifies the 10-byte representation of the *date-stamp* in the format CCYY-MM-DD.

date-stamp can be one of the following:

- A string literal enclosed in single quotation marks
- A name of a user-defined variable data field containing the date string

Example

In the following example, the DATEEXT function is used to convert a 10-byte string date, with the format CCYY-MM-DD, to an 8-byte binary value:

Initial value:

```
DATE-FIELD: '2007-08-01'
```

Statement:

```
MOVE DATEEXT (DATE-FIELD) TO DATE-BINARY
```

Returned value:

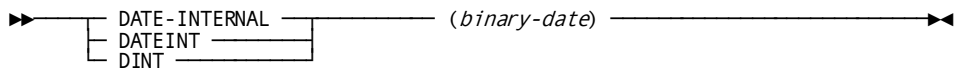
```
x0165DB0000000000
```

DATEINT

Purpose

Returns an 8-byte internal binary-date stamp as a displayable 10-byte date stamp. The returned value is in the format CCYY-MM-DD.

Syntax



Parameter***binary-date***

Specifies the user-defined variable that contains an 8-byte internal *binary-date* stamp.

binary-date must be the name of a user-defined variable that contains an 8-byte internal binary date stamp.

Example

In the following example, the DATEINT function converts an 8-byte internal date stamp to a 10-byte displayable value. The returned value is in the format CCYY-MM-DD.

Initial value:

DATE-STAMP-BINARY: x0165DB0000000000

Statement:

MOVE DATEINT(DATE-STAMP-BINARY) TO DATE-FIELD

Returned string:

'2007-08-01'

DATEOFF

Purpose

Returns the date resulting from adding a specified number of days to a specified date.

Syntax

► DATEOFF
DATEOFFX (*gregorian-date*, *offset*) ◄

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
```

Example 1

In the following example, 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.
```

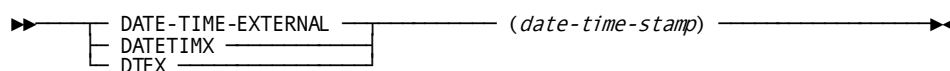
Returned value: 20000102

DATETIMX

Purpose

Returns a 26-byte external date-time stamp as an 8-byte date-time internal binary stamp.

Syntax



Parameter

date-time-stamp

Specifies the 26-byte date-time-stamp to convert to an 8-byte binary date-time-stamp.

date-time-stamp can be one of the following:

- A string literal enclosed in quotation marks in the format CCYY-MM-DD-HH.MM.SS.NNNNNN
- The name of a user-defined variable containing the date-time string

Example

In the following example, the DATETIMX function converts a 26-byte string date-time, with the format CCYY-MM-DD-HH.MM.SS.NNNNNN, to an 8-byte binary value:

Initial value:

```
DT-FIELD: '2007-08-01-15.37.11.876526'
```

Statement:

```
MOVE DATETIMX(DT-FIELD) TO DATE-TIME-BINARY
```

Returned value:

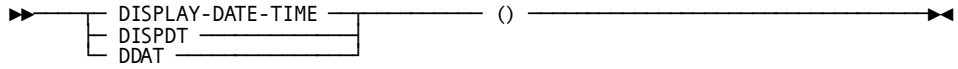
```
x0165DB0DBA7D5FEE
```

DISPDT

Purpose

Returns the current date-time stamp as a 26-byte displayable date-time stamp. The returned value is in the format CCYY-MM-DD-HH.MM.SS.NNNNNN.

Syntax



Example

In the following example, the DISPDT function is used to move the current date-time stamp to the field DATE-TIME-FIELD. The DISPDT function is executed on August 1, 2007 at approximately 3:37 p.m.

Statement:
MOVE DISPDT() TO DATE-TIME-FIELD

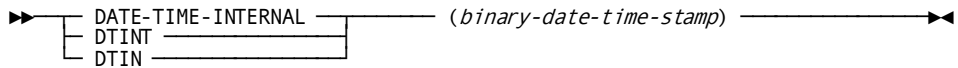
Returned string:
'2007-08-01-15.37.11.876526'

DTINT

Purpose

Returns an 8-byte internal binary date-time stamp as a 26-byte displayable date-time stamp.

Syntax



Parameter

binary-date-time-stamp

Specifies the user-defined variable that contains an 8-byte internal binary date-time stamp.

binary-date-time stamp must be the name of a user-defined variable that contains an 8-byte internal binary date-time stamp.

Example

In the following example, the DTINT function converts an 8-byte internal date-time stamp to a 26-byte displayable date-time stamp. The returned value is in the format CCYY-MM-DD-HH.MM.SS.NNNNNN.

Initial value:

```
DATE-TIME-STAMP-BINARY: x0165DB0DBA7D5FEE
```

Statement:

```
MOVE DTINT(DATE-TIME-STAMP-BINARY) TO DT-FIELD
```

Returned string:

```
'2007-08-01-15.37.11.876526'
```

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 following example, 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](#) (see page 189) and in [STRING-LENGTH](#) (see page 224).

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 following example, 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  '

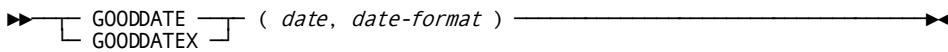
```

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:

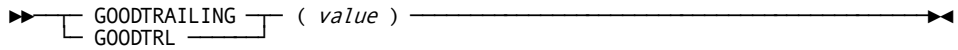
```
IF (GOODDATE(MYDATE, 'C')) THEN
    CALL DATECONV.
ELSE
    CALL DATERROR.
```

GOODTRAILING

Purpose

Returns TRUE or FALSE to indicate whether the value passed is a valid trailing sign field.

Syntax



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.
```

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 following example, 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
```

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 following example, the INSERT function is used with the SUBSTRING function to insert the first six letters of the string contained in EMP-LNAME (PICX(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*'
```

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

```

▶▶ [ INVERT-SIGN | INVERT ] ( value )

```

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 following example, the sign inversion function is used to form the negative of a value if the transaction code is 'DB':

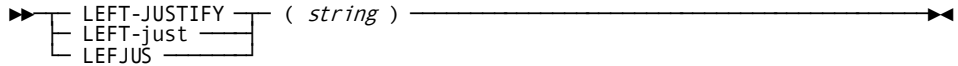
```
Initial values:
  TRANS-CODE: 'DB'
  WK-AMT:     453.29
Statements:
  IF TRANS-CODE EQ 'DB'
  THEN
    MOVE INVERT-SIGN(WK-AMT) TO WK-AMT.
Returned value: -453.29
```

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



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 following example, 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      '

```


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

Example 1: Testing for an embedded string

In the following example, 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 following example, 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.
```

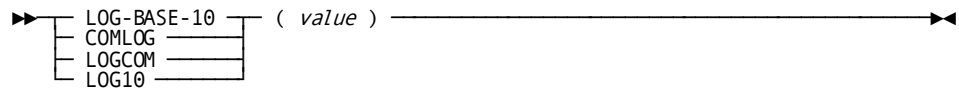
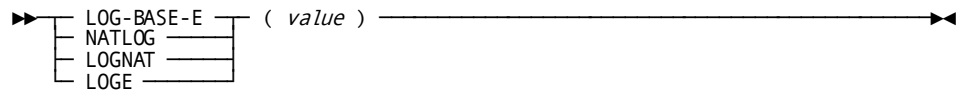
Example 3: Examples using an escape character

- 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*%', '*')

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 following example, the logarithm function is used to calculate the base-10 logarithm of a numeric value:

Initial value:

WK-VALUE: 100

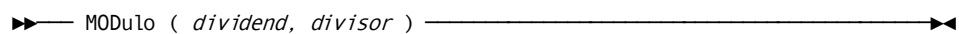
Statement:

MOVE LOG-BASE-10(WK-VALUE) TO WK-LOG-EQUIVALENT.

Returned value: 2

MODULO**Purpose**

Returns the modulus (remainder) of one numeric value divided by another.

Syntax

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 following example, 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.

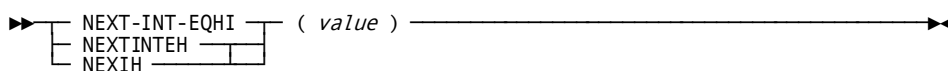
Returned value: 3

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 following example, 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

NEXT-INT-EQLO

Purpose

Returns the largest integer that is equal to or less than a numeric value.

Syntax

NEXT-INT-EQLO (*value*)

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 following example, 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
```

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.
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.

Valid examples	Invalid examples
.5	- .6
-9	

Example

In the following example, 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.

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 following example, 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.
```

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 following example, 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:

```
'AAXXCXXXBDD'
```

A further example of the replace function is provided in [SUBSTRING](#) (see page 226).

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 ) ▶▶
   RIGHTjus
```

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 following example, the right justify function is used to right justify EMP-LNAME (PICX(20)):

Initial value:

```
EMP-LNAME: '      SMITH      '
```

Statement:

```
MOVE RIGHT-JUSTIFY(EMP-LNAME) TO EMP-LNAME.
```

Returned string:

```
'      SMITH'
```

SIGN-VALUE

Purpose

Returns a +1, 0, or -1, depending on whether the specified numeric value is positive, zero, or negative, respectively.

Syntax

►► $\left[\begin{array}{l} \text{SIGN-VALUE} \\ \text{SIGV} \end{array} \right] (\text{value})$ —————►►

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 following example, 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

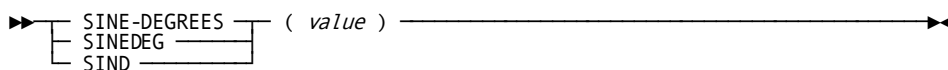
SINE

Purpose

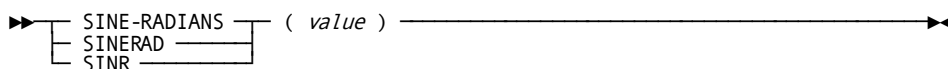
Returns the sine of a numeric value that represents an angle in either degrees or radians.

Syntax

Sine (degrees):



Sine (radians):



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):

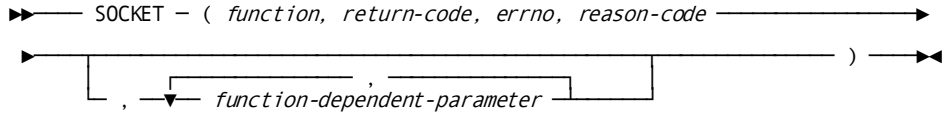
```
MOVE SIND(-60) TO WK-RESULT.
Return value: -0.8660
```

SOCKET

Purpose

Interface from CA ADS to TCP/IP.

Syntax



Parameters

The parameters are the standard parameters used for CA IDMS interface to TCP/IP as documented in *CA IDMS Callable Services Guide*.

The first four parameters passed are always the same:

1. function: a 4-byte binary field, S9(5) USAGE IS COMP, in which the desired socket function must be filled in by the programmer.
2. return-code: a 4-byte binary field which receives the outcome of the operation. Returned values are:
 - 0 - no errors occurred
 - 20 - a parameter list error was encountered
 - -1 - a socket error was encountered; the errno and reason-code fields contain more detailed information about the error.
3. errno: a 4-byte field which receives the ERRNO value when the return code is -1.
4. reason code: a 4-byte binary field which receives the reason code value when the return-code is -1.

Depending on the function, zero or more parameters can follow.

Note: If an optional parameter is not to be specified in the parameter list, it should be replaced by the @ character. A pre-defined record in the dictionary, *SOCKET-CALL-INT*, describes the socket function's return codes and the errno's, and should be included as a work record.

An ADS dialog associated with a server task (a task started by a listener):

- Must be mapless
- Should include *SOCKET-LISTENER-PARMS* as a work record.

The build-in function should be used within an IF statement. For example:

```
IF (SOCKET(function,
           return-code,
           errno,
           reason-code,
           function-dependent-parameter1,
           ...)) NE ZERO
```

SQUARE-ROOT

Purpose

Returns the square root of a numeric value.

Syntax

►► `SQUARE-ROOT`
`SQRT` (*value*)

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 following example, 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](#) (see page 213).

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

►► `STRING-INDEX`
`INDEX`
`INDX` (*string*, *search-string*)

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 following example, 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
```

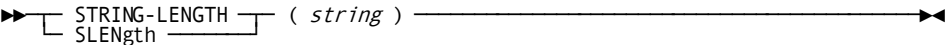
Since the string 'ABC' appears in the product code (starting at character position 8), the condition is false.

STRING-LENGTH

Purpose

Returns the length of a string value.

Syntax



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 following example, 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

Note: If EXTRACT is called with an argument consisting only of spaces (x'40's), then it returns a string of one space. If that is passed on to the STRING-LENGTH function, the result will be 1.

Initial value:

EMP-LNAME: ' '

Statement:

MOVE SLENGTH(EXTRACT(EMP-LNAME)) TO WK-NAME-LENGTH.

Returned string from extract function:

' '

Returned value from slength function:

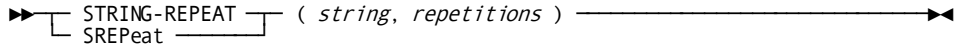
1

STRING-REPEAT

Purpose

Returns the string that results from repeating a string value a specified number of times.

Syntax



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 following example, the repeat function is used to repeat the constant 'NAME' two times:

Statement:

```
MOVE SREPEAT('NAME',2) TO WK-TARGET.
```

Returned string:

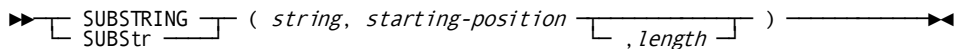
```
'NAMENAMENAME'
```

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.

Example 1: Extracting a substring

In the following example, the substring function is used to extract a substring of EMP-LNAME (PICX(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](#) (see page 206).

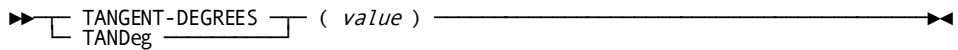
TANGENT

Purpose

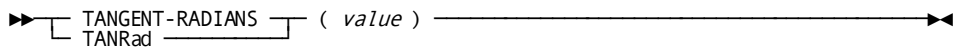
Returns the tangent of a numeric value that represents an angle in either degrees or radians.

Syntax

Tangent (degrees):



Tangent (radians):



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:

$$(n * 180) + 90$$

- **For the tangent (radians) function**, *value* cannot be a value equal to the following expression:

$$(n * \pi) + \pi/2$$

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.
```

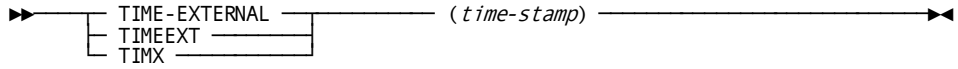
Returned value: 1.7321

TIMEEXT

Purpose

Returns an 8-byte displayable time as an 8-byte internal binary time stamp.

Syntax



Parameter

time-stamp

Specifies the 8-byte displayable time stamp to be converted into an 8-byte binary internal time stamp.

time-stamp can be one of the following:

- A string literal enclosed in quotation marks in the format HH.MM.SS
- The name of a user-defined variable that contains the 8-byte time stamp string

Example

In the following example, the TIMEEXT function is used to convert an 8-byte displayable time stamp to an 8-byte binary internal time stamp in the format HH.MM.SS:

Initial value:

TIME-FIELD: '17.08.09'

Statement:

MOVE TIMEEXT(TIME-FIELD) TO TIME-BINARY

Returned value:

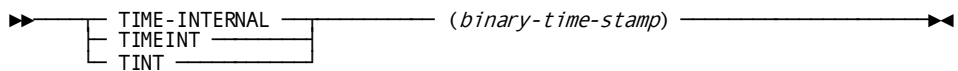
x00000000F0F90000

TIMEINT

Purpose

Returns an 8-byte internal binary time stamp as a displayable 8-byte time stamp.

Syntax



Parameters

binary-time-stamp

Specifies the 8-byte internal binary time stamp.

binary-time-stamp must be the name of a user-defined variable that contains an 8-byte internal binary time stamp.

Example

In the following example, the TIMEINT function converts an 8-byte internal binary time stamp to an 8-byte displayable time stamp in the format HH.MM.SS:

Initial value:

TIME-BINARY: x0000000F0F900000

Statement:

MOVE TIMEINT (TIME-BINARY) TO TIME-FIELD

Returned string:

'17.08.09'

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

Example 1

In the following example, 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

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 following example, 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      '
```

TOMORROW

Purpose

Returns tomorrow's date in the format requested.

Syntax

```
►► ┌ TOMORROW ─── ( date-format ) ───────────────────────────────────►►
   │ TOMORROWX │
```

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

Example 1

In the following example, 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

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 following example, 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      '
```

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 following example, 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')
```

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 hexadecimal 00 through FF.

Example

In the following example, 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 '
```

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 following example, 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](#) (see page 226).

WEEKDAY

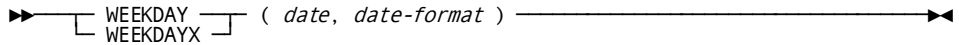
Purpose

Returns the weekday (Monday, Tuesday, etc.) of a specified date.

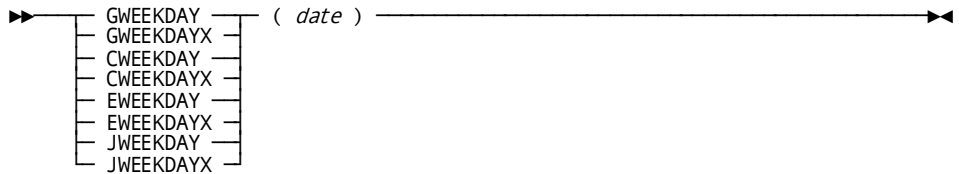
Weekday functions can be coded in two ways, as shown in the syntax diagrams below.

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

Example 1 (Format 1)

In the following example, 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.
```

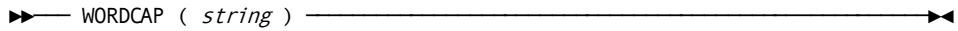
Returned value: 'TUESDAY'

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



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 following example, 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
```

YESTERDAY

Purpose

Returns yesterday's date in the format requested.

Syntax

```
► ┌ YESTERDAY ───┐ ( date-format ) ───────────────────────────────────►
  └ 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

Example 1

In the following example, 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

ZONED-TO-TRAILING

Purpose

Returns a COBOL trailing sign numeric from a zoned numeric.

Syntax

►► `ZONED-TO-TRAILING` (*value*) ◄◄
`ZNTRAIL`

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 following example, 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

This section contains the following topics:

[Overview](#) (see page 245)

[General Considerations](#) (see page 246)

[Batch-Control Event Condition](#) (see page 248)

[Command Status Condition](#) (see page 249)

[Comparison Condition](#) (see page 251)

[Cursor Position Condition](#) (see page 253)

[Dialog Execution Status Condition](#) (see page 254)

[Environment Status Condition](#) (see page 256)

[Level-88 Condition](#) (see page 257)

[Map Field Status Condition](#) (see page 257)

[Map Paging Status Conditions](#) (see page 261)

[Set Status Condition](#) (see page 265)

[Arithmetic and Assignment Command Status Condition](#) (see page 266)

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

Condition	Purpose
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

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

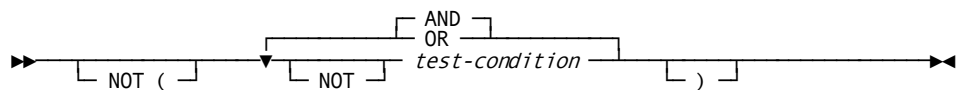
The default order of precedence can be overridden by using parentheses. The expression in the innermost parentheses is evaluated first.

Significant tests in conditional expressions should be coded to the left for greater runtime efficiency.

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.

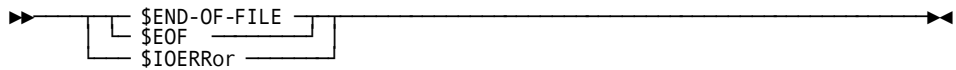
Batch-Control Event Condition

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 following example, execution of a group of commands continues until an end-of-file condition occurs:

```
WHILE NOT $EOF
  REPEAT.
  .
  .
  .
  WRITE TRANSACTION.
  END.
LEAVE APPLICATION.
```

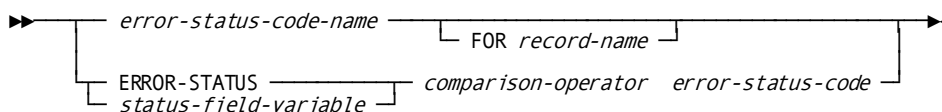

Command Status Condition

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.

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

Operator	Synonym	Meaning
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

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'...
```

More information:

[Error Handling](#) (see page 277)

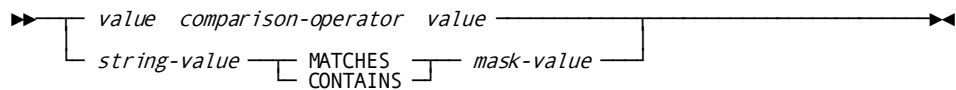
Comparison Condition

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 [Introduction to Process Language](#) (see page 155).

comparison-operator:

The comparison operators are:

Operator	Synonym	Meaning
EQ	=	Equal
NE		Not equal to
GT	>	Greater than
LT	<	Less than

Operator	Synonym	Meaning
GE		Greater than or equal to
LE		Less than or equal to

string-value:

Either the name of an elementary EBCDIC, 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*.

'String-value' in either MATCHES or CONTAINS must be defined as either EBCDIC (PIC X) or as unsigned zoned decimal (PIC 9).

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:

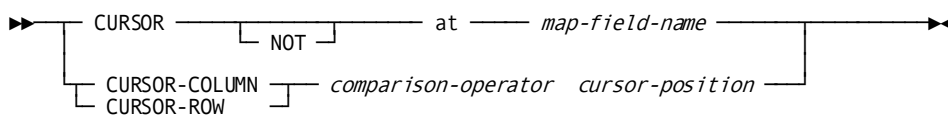
```
IF PART-ID MATCHES '**@398' ...
```

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 *cursor-position* is made using the value in the CURSOR-COLUMN.

CURSOR-ROW

Specifies that the comparison with *cursor-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.

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.

Example

The following example shows the use of the dialog execution status condition:

```
IF FIRST-TIME
THEN
    MOVE 1 TO COUNTER.
ELSE
    ADD 1 TO COUNTER.
```

More information:

[Control Commands](#) (see page 325)

Environment Status Condition

Purpose

Determines the application's environment.

Status conditions can be tested in both online and batch environments.

Syntax

```
┌──────────┴──────────┐
├── $BATCH ───────────┤
├── $ONLINE ───────────┤
└──────────┬──────────┘
```

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.
```


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.

Map Field Status Condition

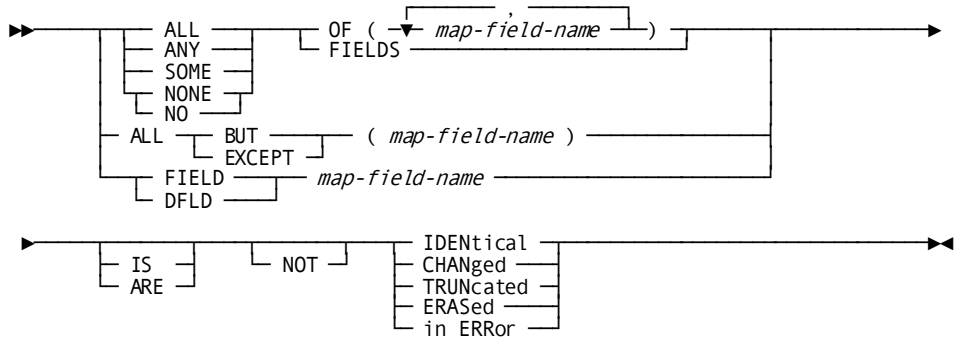
Purpose

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.

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.

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 ...
```

More information:

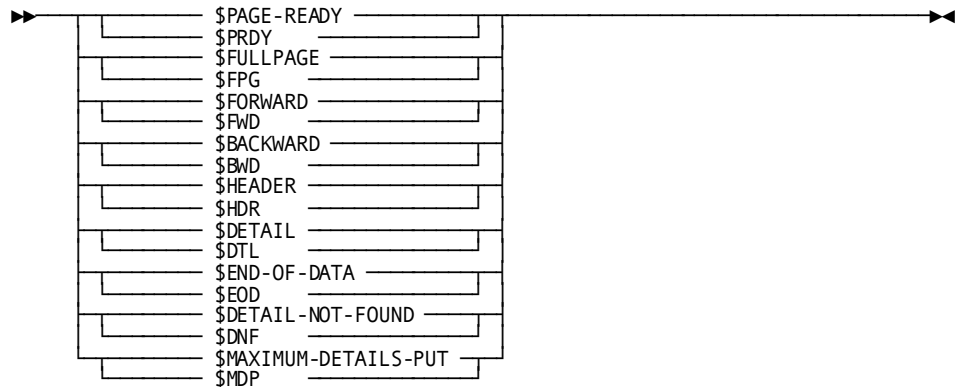
[CA ADS Dialog Compiler \(ADSC\)](#) (see page 91)
[Map Commands](#) (see page 449)

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 not be chosen for the dialog. This setting is made using the Map Specification screen.

`$PRDY` can be used in place of `$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.

Example

The following example 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).

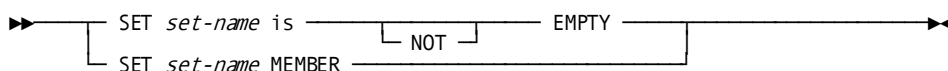
Set Status Condition

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.

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.

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  
.  
.  
.
```

More information:

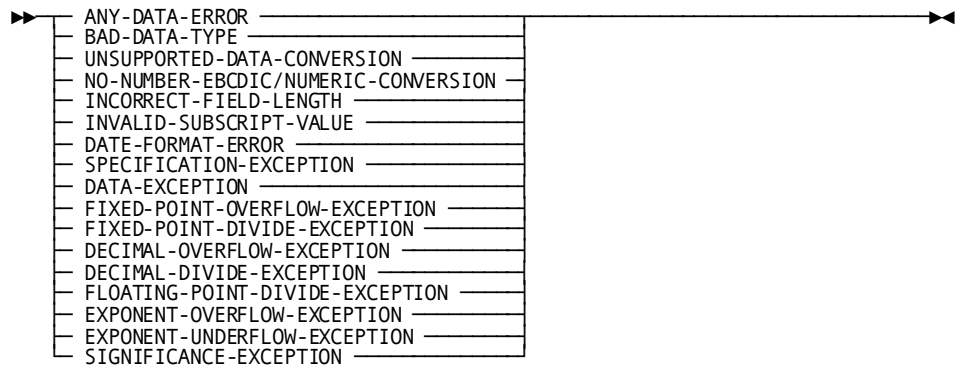
[Control Commands](#) (see page 325)

Arithmetic and Assignment Command Status Condition

Purpose

Tests the results of the previous assignment command.

Syntax



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 MSG MESSAGE TEXT 'SOURCE DATA TOO LARGE'.  
IF ANY-DATA-ERROR  
    DISPLAY MSG MESSAGE TEXT 'INVALID DATA VALUE'.
```


Chapter 9: Constants

This section contains the following topics:

- [Overview](#) (see page 269)
- [Figurative Constants](#) (see page 269)
- [Graphic Literals](#) (see page 271)
- [Multibit Binary Constants](#) (see page 272)
- [Nonnumeric Literals](#) (see page 273)
- [Numeric literals](#) (see page 273)

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

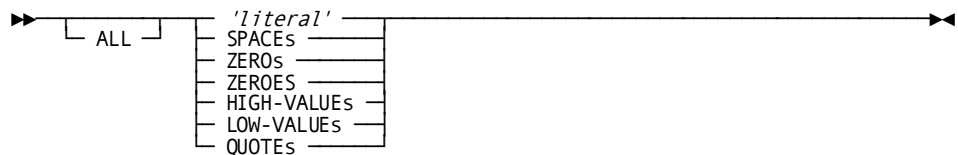
Figurative Constants

Purpose

A figurative constant is a reserved CA ADS word that represents a numeric value, a character, or a string of characters.

A figurative constant can be used as the source field in a MOVE operation or as an operand in a comparison expression.

Syntax



Parameters

ALL

Specifies that the figurative constant is repeated to fill the target field in a MOVE statement.

ALL can precede *'literal'*. Other figurative constants do not need to be preceded by ALL, since they always repeat to fill the target field.

'literal'

A nonnumeric literal of up to 255 characters, enclosed in single quotation marks.

If ALL is specified, the literal value is repeated as many times as required to fill the field. If ALL is not specified, any remaining positions in the target field are filled with blanks.

SPACEs

Represents a field that contains all blanks.

ZEROs/ZEROES

Represents a field that contains all zeros.

Note: PICX fields are treated as unsigned zoned decimal fields. ZEROS or ZEROES is the only figurative constant that can be specified in arithmetic expressions.

HIGH-VALUEs

Represents a field filled with the character that has the highest value in the computer collating sequence (that is, X'FF').

LOW-VALUEs

Represents a field filled with binary zeros (that is, X'00').

QUOTEs

Represents a field filled with single quotation marks.

Usage

The VALUE IS clause for fields in records used by an 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.

Example 1: Moving zero to a numeric field

```
MOVE ZERO TO COUNTER.
```

Example 2: Filling a field with binary zeros

```
MOVE ALL 'X0' TO HUGS-AND-KISSES.
```

Example 3: Comparing the contents of a field

```
IF EMP-NAME EQ SPACES  
  THEN  
    DISPLAY TEXT 'ENTER EMPLOYEE NAME'.
```

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.

Note: For more information about defining data to handle DBCS characters in the data dictionary, see the *CA IDMS IDD Quick Reference Guide*. For more information about defining maps that handle DBCS characters, see the *CA IDMS Mapping Facility Guide*.

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.
```

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 referenced 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.

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.
```


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 ('').

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.
```

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.

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

This section contains the following topics:

[Overview](#) (see page 277)

[The Autostatus Facility](#) (see page 278)

[Error Expressions](#) (see page 279)

[The ALLOWING Clause](#) (see page 280)

[Status Definition Records](#) (see page 281)

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.

Note: For more information about conditional statements and the WHENEVER SQLERROR statement processed during an SQL session, see the *CA IDMS SQL Programming Guide*.

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.

Note: For more information about using automatic editing and error-handling facilities to evaluate input data, see the *CA IDMS Mapping Facility Guide*.

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.

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.

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

Note: For more information about the AUTOSTATUS clause of the system generation ADSO statement, see the *CA IDMS System Generation Guide*.

More information:

[CA ADS Dialog Compiler \(ADSC\)](#) (see page 91)
[Database Access Commands](#) (see page 363)

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.

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*.

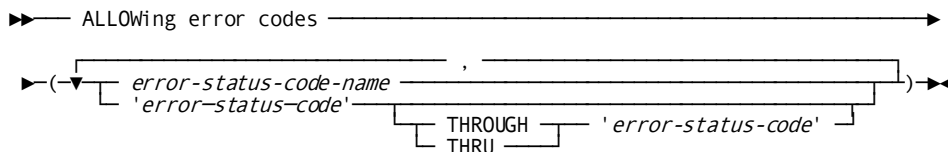
An error expression is allowed only if a dialog is compiled with the **Autostatus** option selected. If a dialog is compiled without the **Autostatus** option, the expression is flagged as an error during process compilation.

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.

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.

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

Note: For more information about the system generation ADSO statement, see the *CA IDMS System Generation Guide*.

A status definition record is associated with a dialog during dialog compilation. However, a buffer for this record is not allocated at runtime.

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.

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.

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.

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.
```

Note: For more information about using the RECORD statement to add, modify, or delete status definition records, see the *CA IDMS IDD Quick Reference Guide*.

More information:

[CA ADS Dialog Compiler \(ADSC\)](#) (see page 91)

ADSO-STAT-DEF-REC

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'.
```

Chapter 11: Variable Data Fields

This section contains the following topics:

[Overview](#) (see page 285)

[User-Defined Data Field Names](#) (see page 285)

[System-Supplied Data Field Names](#) (see page 287)

[Entity Names](#) (see page 293)

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.

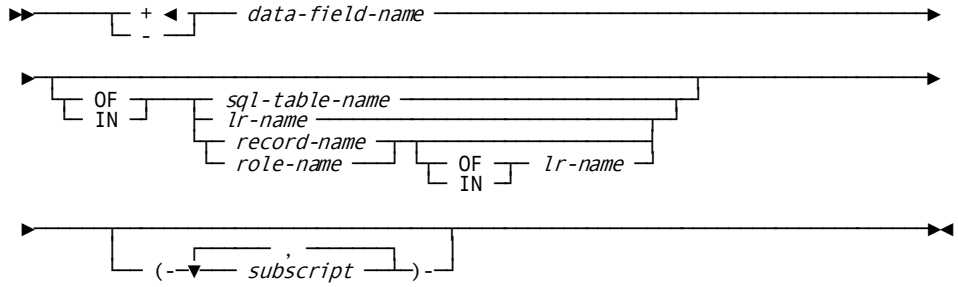
User-Defined Data Field Names

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.

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.
```

More information:

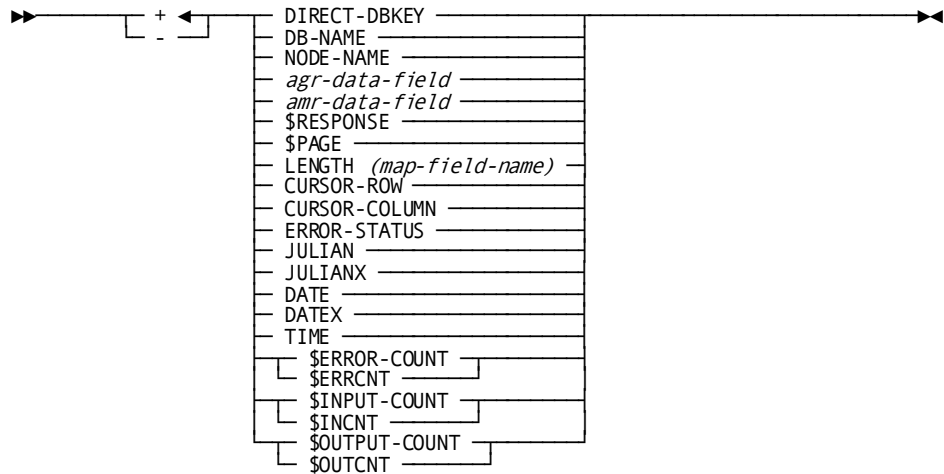
[Records and Tables Screen](#) (see page 111)

System-Supplied Data Field Names

Purpose

System-supplied data field names specify variable data fields supplied by the CA ADS runtime system.

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

DIRECT-DBKEY

References a binary full word field that contains the database key of the record being stored.

DIRECT-DBKEY is used in conjunction with a STORE operation when the location mode of the record being stored is DIRECT.

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,

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).

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.

\$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.

Once a response is selected, the \$RESPONSE field is cleared.

Note: For more information about the \$RESPONSE map field, see the *CA IDMS Mapping Facility Guide*.

\$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.

- 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.

Note: For more information about the \$PAGE map field, see the *CA IDMS Mapping Facility Guide*.

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.

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.

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.

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.
```

More information:

[System Records](#) (see page 567)

[CA ADS Runtime System](#) (see page 119)

[Control Commands](#) (see page 325)

[Initiating the CA ADS Runtime System](#) (see page 119)

[Map Commands](#) (see page 449)

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

This section contains the following topics:

[Overview](#) (see page 295)

[Summary Of Process Commands](#) (see page 296)

[INCLUDE](#) (see page 300)

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.

More Information:

[Syntax Diagram Conventions](#) (see page 19)

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 an CA ADS application

Category	Keywords	Purpose
	LINK	Passes control to a lower level dialog or to a user program with inline return expected
	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

Category	Keywords	Purpose
	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
	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

Category	Keywords	Purpose
	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
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 an CA IDMS/DC or DC/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.

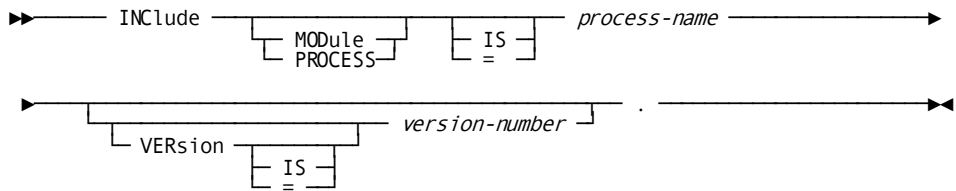
Note: For more information about statements (WHENEVER) and database commands, see the *CA IDMS SQL Reference Guide*. For more information about using SQL with CA ADS, see the *CA IDMS SQL Programming Guide*.

INCLUDE

Purpose

Inserts stored process source code into another process at compile time.

Syntax



Parameters

MODule 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 following example 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'.

```

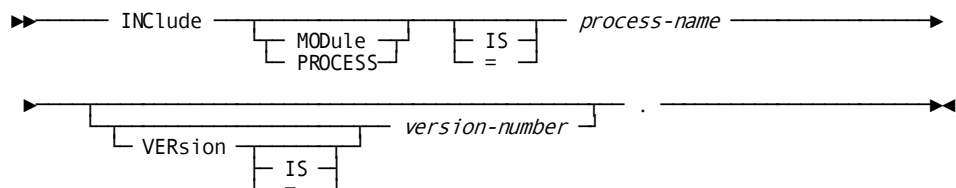
If the application developer specifies CUST-NUM-CHECK as a premap or response process using the CA ADS dialog compiler (ADSC), on the Process Modules screen, the following process source code will logically be compiled:

```

MOVE CUST-NUM TO A.
IF A = 1
THEN
    DISPLAY.
ELSE
    LINK TO 'LINKDIAL'.
RETURN.

```

Dialog Compiler Directive



Chapter 13: Arithmetic and Assignment Commands

This section contains the following topics:

- [Overview](#) (see page 303)
- [General Considerations](#) (see page 304)
- [Arithmetic Commands](#) (see page 306)
- [Assignment Command](#) (see page 314)

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.

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

More information:

- [Built-in Functions](#) (see page 175)

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.

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.

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.

More information:

[Assignment Command](#) (see page 314)

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. After an exception condition is encountered, the data 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'.
```

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.

ADD

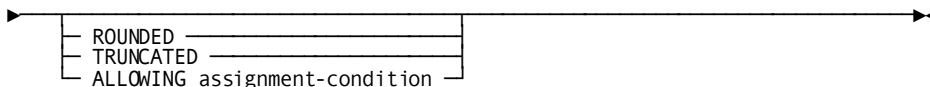
Purpose

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*.

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.

More information:

For more information, see 13.4, "Assignment Command" later in this chapter.

ALLOWING

Specifies which error conditions would normally abend and 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 following example uses the ADD command to add the value 1 to the contents of the variable data field COUNTER.

```
ADD 1 TO COUNTER.
```

More information:

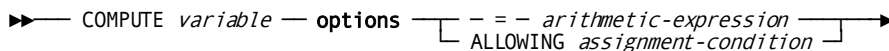
[Arithmetic Expressions](#) (see page 171)

COMPUTE

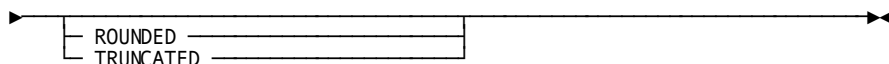
Purpose

Evaluates an arithmetic expression. The result of the evaluation is placed in a variable data field.

Syntax



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.

ALLOWING

Specifies which error conditions would normally abend and 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*.

More information:

For information on *arithmetic-expression*, see Chapter 6, Arithmetic Expressions.

Example

The following example 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).

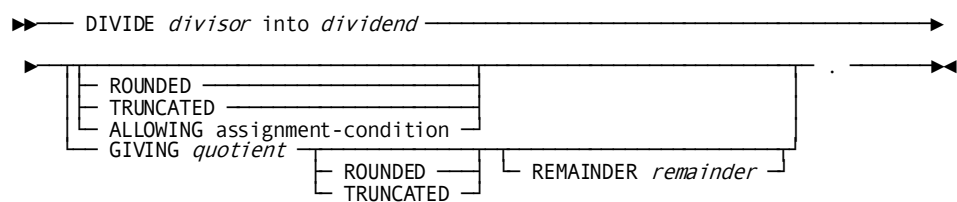
```

More information:

[Assignment Command](#) (see page 314)

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.

ALLOWING

Specifies which error conditions would normally abend and 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.

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.
```

More information:

[Assignment Command](#) (see page 314)

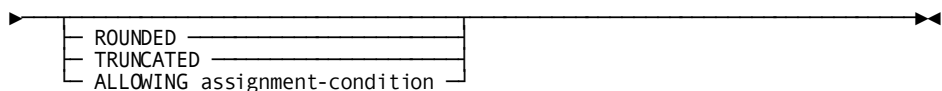
MULTIPLY

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*.

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,

ALLOWING

Specifies which error conditions would normally abend and 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 following example 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).
```

More information:

[Assignment Command](#) (see page 314)

[Arithmetic Expressions](#) (see page 171)

SUBTRACT

Purpose

Calculates the difference between two variables.

Syntax

```
►► SUBTRACT arithmetic-expression from variable — options — . ◀◀
```

Expansion of options

```
► [
  |— ROUNDED —
  |— TRUNCATED —
  |— ALLOWING assignment-condition —
  ] ◀◀
```

Parameters

arithmetic-expression

Specifies the arithmetic expression being subtracted from the value contained in *variable*.

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.

ALLOWING

Specifies which error conditions would normally abend and 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 following example 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.
```

More information:

[Assignment Command](#) (see page 314)

[Arithmetic Expressions](#) (see page 171)

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.

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.

More information:

[CA ADS Dialog Compiler \(ADSC\)](#) (see page 91)

MOVE

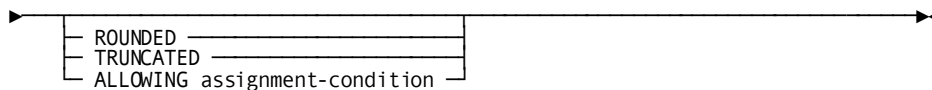
Purpose

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 would normally abend and 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 following example 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

This section contains the following topics:

[Overview](#) (see page 317)

[EXIT](#) (see page 318)

[IF](#) (see page 319)

[NEXT](#) (see page 321)

[WHILE](#) (see page 322)

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.

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 ON1 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 [Database Access Commands](#) (see page 363).

More information:

[Conditional Expressions](#) (see page 245)

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.

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.
```

More information:

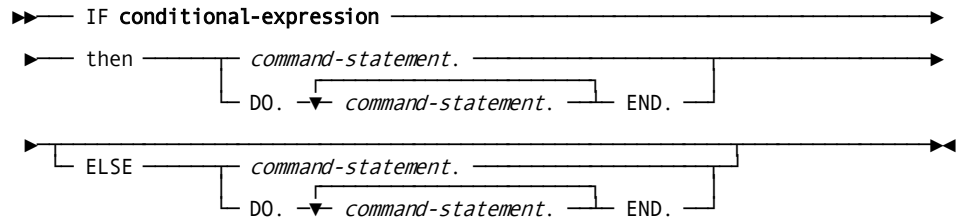
[Database Access Commands](#) (see page 363)

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 [Conditional Expressions](#) (see page 245).

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.

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'.

```

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.

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.

More information:

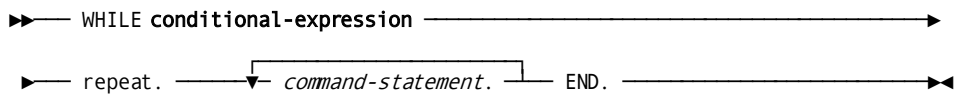
[Database Access Commands](#) (see page 363)
[IF](#) (see page 319)

WHILE

Purpose

Creates a processing loop based on conditions in a specified expression.

Syntax



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 [Conditional Expressions](#) (see page 245).

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 following example 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.
```

Chapter 15: Control Commands

This section contains the following topics:

- [Overview](#) (see page 325)
- [General Considerations](#) (see page 326)
- [CONTINUE](#) (see page 332)
- [DISPLAY](#) (see page 334)
- [EXECUTE NEXT FUNCTION](#) (see page 339)
- [INVOKE](#) (see page 341)
- [LEAVE](#) (see page 343)
- [LINK](#) (see page 345)
- [READ TRANSACTION](#) (see page 353)
- [RETURN](#) (see page 353)
- [TRANSFER](#) (see page 356)
- [WRITE TRANSACTION](#) (see page 358)

Overview

CA ADS control commands are used to pass control during the execution of an application. The execution of a control command terminates the execution of the process that issues the command. A control command can pass control to:

- Another dialog
- A copy of the same dialog
- Another component within the same dialog
- A user-written program
- Another application function when using the EXECUTE NEXT FUNCTION command

Summary of Control Commands

Control commands are listed in the table below. Each command is presented in alphabetical order after [General Considerations](#) (see page 326).

Command	Purpose
CONTINUE	Terminates the current process, executes the dialog's premap process, and writes a message
DISPLAY	Displays a dialog's map, reexecutes a dialog's premap process, or specifies a message that appears in a map's message field

Command	Purpose
EXECUTE NEXT FUNCTION	Passes control to the application function associated with a response by means of the control command specified for the response during application compilation
INVOKE	Initiates execution of a lower level dialog in the application thread
LEAVE	Terminates the current application or terminates the current CA ADS session
LINK	Initiates execution of a lower level dialog, creating a nested application structure, or initiates execution of a user program
READ TRANSACTION	Terminates the current process, performs a mapin operation, and selects the next application function or response to be executed (batch only)
RETURN	Initiates execution of a higher level dialog
TRANSFER	Initiates execution of a dialog at the same level as the dialog passing control
WRITE TRANSACTION	Terminates a current process, performs a mapout operation, and passes control within an application (batch only)

General Considerations

At run time, control commands connect the application functions or dialogs that make up the application by directing the flow of control. The following diagram shows how control commands pass control from one function or dialog to another. The way that control is transferred determines the data that is available to the function or dialog when it receives process control.

The application developer associates control commands with application responses by using the Response Definition screen during application compilation.

Alternatively, the application developer can code control commands wherever appropriate in a premap or response process.

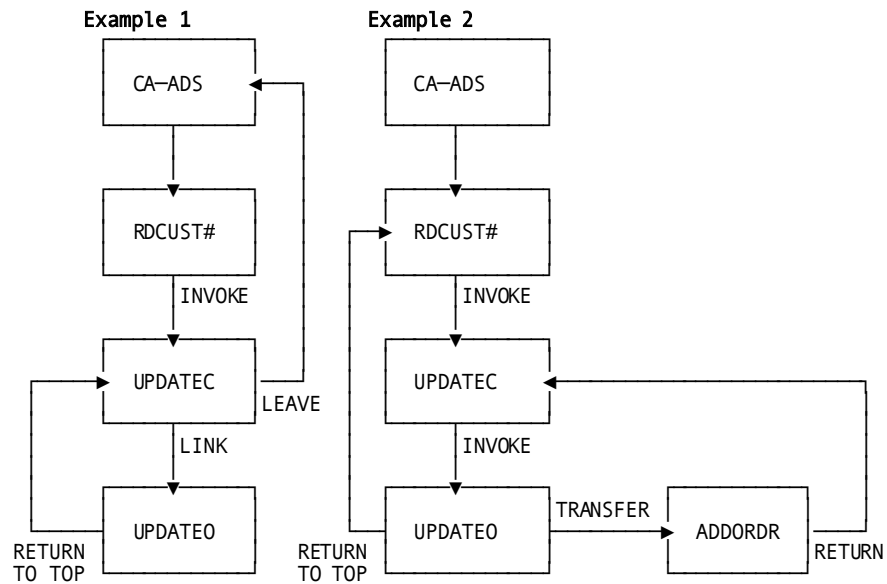
More information:

[CA ADS Application Compiler \(ADSA\)](#) (see page 51)

Application Thread

The current sequence of operative functions or dialogs in an application is called the application thread. A single dialog can occur more than once in an application structure and can execute more than once within an application thread, whether or not the function or dialog remains operative.

Control Command Processing



Operative and Nonoperative Dialogs

At run time, a function or dialog can be either operative or nonoperative within an application thread.

Operative Dialog

A dialog becomes **operative** when it receives processing control. A function or dialog remains operative when it passes control to a lower level function or dialog or to another part of itself.

Only one dialog can be operative at any time on any given application level. As long as a dialog or dialog function remains operative, all record buffers associated with the dialog are maintained.

Nonoperative Dialog

A function or dialog becomes **nonoperative** when it passes control to a higher level-function or dialog or to a function or dialog (including a copy of itself) on the same level. All functions and dialogs become nonoperative when control passes out of the application.

When a dialog or dialog function becomes nonoperative, the record buffers established by that dialog are released.

Application Levels

The first function or dialog executed in an application establishes the top level of the application structure. The INVOKE and LINK commands establish lower levels in the structure.

Maximum Number of Levels

By default, an application structure can contain a maximum of ten levels. This maximum number of levels can be reduced at system generation time. If the execution of an INVOKE or LINK command causes the maximum allowable number of levels to be exceeded, CA ADS abnormally terminates the application. The application developer should limit the total number of nested INVOKE and LINK commands accordingly.

Mainline Dialog

The dialog at the top of an application structure must be a mainline dialog. The application developer defines a dialog as mainline by using the Options and Directives screen of the dialog compiler.

If a dialog function is initiated by an application task code, the dialog associated with the function must be a mainline dialog. The application developer associates a function with a task code by using the Task Codes screen during application definition.

More information:

[CA ADS Application Compiler \(ADSA\)](#) (see page 51)

[CA ADS Dialog Compiler \(ADSC\)](#) (see page 91)

The Menu Stack

System-supplied menu-handling routines use a menu stack to keep track of menu execution at run time. The menu stack is maintained automatically at run time.

Considerations

The following considerations apply:

- When a menu (or menu/dialog) function is executed in an online application, the function name is added to the internal menu stack. The menu or menu/dialog function name is removed from the menu stack when a POP or RETURN function returns control in either of the following ways:
 - To the menu
 - To a menu that is higher in the menu stack
- If a menu or menu dialog function is already in the menu stack when a LINK, INVOKE, or TRANSFER command passes control again to the function, the first occurrence of the name is deleted from the stack. The name is then added to the end of the stack, as usual.
- Each menu name can appear only once in the menu stack.

Database Currencies

Database currencies are established by the last database command in an operative dialog. Currencies are saved and made available to lower level dialogs and to the dialog that established the currencies if control returns to that dialog from a lower application level.

Considerations

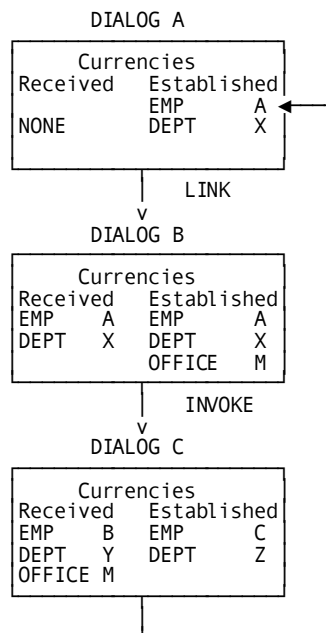
The following considerations apply to currencies:

- Database currencies are cumulative.

Currencies established by each dialog or dialog function are passed to lower level dialogs along with any currencies received from a higher level dialog. A lower level dialog can establish new currencies, which are passed to the next lower level dialog along with the currencies already established.
- All database currencies established for a dialog are released when a dialog or a dialog function becomes nonoperative.

Unless the dialog or dialog function receiving control specifies the NOSAVE keyword on a LINK command, it establishes its own currencies. These currencies are established either by restoring the currencies saved when it originally passed control or by using currencies previously established by a higher level dialog. The following diagram shows currencies in an CAADS application.

Currency Action



DIALOG A executes and establishes current records of two sets.

DIALOG A links to DIALOG B and establishes a current record of a third set.

DIALOG B invokes DIALOG C.

DIALOG C returns control to DIALOG A:

- Only currencies established by DIALOG A are available.
- Record buffers still contain data established by DIALOG C

Effect of Control Commands on Issuing and Receiving Dialogs

The following table outlines the effect of control commands on issuing and receiving dialogs. The EXECUTE NEXT FUNCTION command is not included in this table. The characteristics established by EXECUTE NEXT FUNCTION depend on which command is actually executed.

Command	New level established	Status of issuing dialog	Data avail. to receiving dialog/program	Currency action for issuing dialog	Currency action for receiving dialog/program
DISPLAY	No	Operative	All data	Saved	N/A
INVOKE	Yes	Operative	All data	Saved	Restored
LEAVE	No	Non-operative	N/A	Released	N/A
LINK:					

Command	New level established	Status of issuing dialog	Data avail. to receiving dialog/program	Currency action for issuing dialog	Currency action for receiving dialog/program
DIALOG	Yes	Operative	All data	Saved, unless NOSAVE is specified	Restored
PROGRAM	No	Operative	All, some, or none (depending on command specification)	Saved, unless NOSAVE is specified	Program receives currencies as part of extended run unit
RETURN	No	Non-operative (any operative dialogs between the issuing dialog and the receiving dialog also become non-operative)	Data previously available to the receiving dialog	Released (currencies for any dialogs between the issuing dialog and the receiving dialog are also released)	Restored
TRANSFER	No	Non-operative	All data except that acquired by the issuing dialog	Released	Can use currencies previously established by higher level dialogs

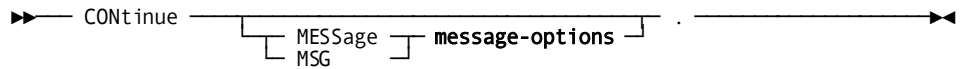
CONTINUE

Purpose

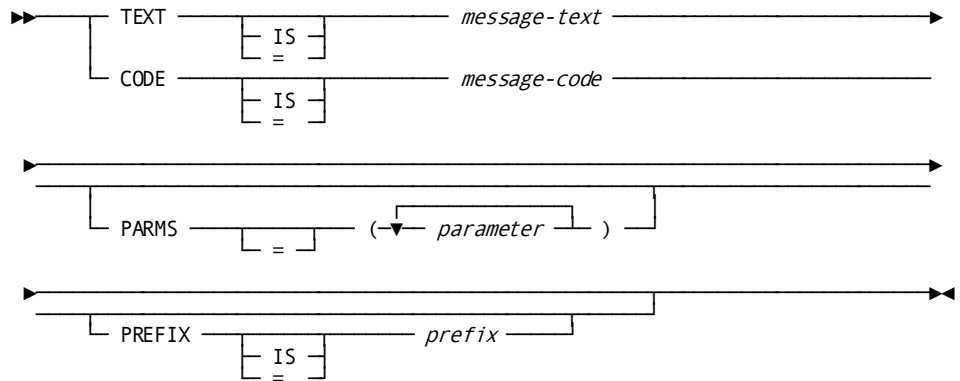
Terminates a current process, executes a dialog's premap process, and specifies a message.

In the online environment, the message appears at the terminal when the dialog executes a DISPLAY command. In the batch environment, the message is sent to the log file and/or the operator's console.

Syntax



Expansion of Message-Options



Parameters

MESSage *message-options*

Identifies message to be displayed.

MSG can be used in place of MESSAGE.

Expanded syntax for *message-options* is shown above immediately following the CONTINUE syntax.

TEXT IS *message-text*

Specifies the text of a message to be displayed in an online map's message field or sent to a batch application and a system log file.

Message-text specifies either the name of a variable data field containing the message text or the text string itself, enclosed in single quotation marks.

The text string can contain up to 240 displayable characters.

IS or = are optional keywords and have no effect on processing.

CODE IS *message-code*

Specifies the message dictionary code of a message to be displayed in an online map's message field or sent to the log file in a batch application.

In a batch application, the message is also sent to the operator, if directed by the destination specified in the dictionary.

Message-code specifies either the name of a variable data field that contains the message code or the 6-digit code itself, expressed as a numeric literal.

IS or = are optional keywords and have no effect on processing.

PARMS = *parameter*

Specifies a replacement parameter for each variable field in the stored message identified by *message-code*.

Up to nine replacement parameters can be specified for a message. Multiple parameters must be specified in the order in which they are numbered and separated by blanks or commas.

Parameter specifies either the name of an EBCDIC or unsigned zoned decimal variable data field that contains the parameter value or the actual parameter value, enclosed in single quotation marks.

The parameter value must contain displayable characters. At run time, each variable data field in a stored message expands or contracts to accommodate the size of its replacement parameter. A replacement parameter can be a maximum of 240 bytes.

PREFIX IS *prefix*

Overrides the default prefix of a dialog and a map.

Prefix must either specify an EBCDIC or unsigned zoned decimal variable data field that contains a 2-character prefix or the 2-character prefix itself, enclosed in single quotation marks

IS or = are optional keywords and have no effect on processing.

Usage

Considerations

- The premap process is reexecuted if CONTINUE is issued in the premap process. This differs from the DISPLAY CONTINUE command, which causes a pseudo-converse in the online environment.
- Any message specified on the CONTINUE command is ignored in the online environment if the DISPLAY command that follows also specifies a message.

- Up to nine replacement parameters can be specified for a message.
- Multiple parameters must be separated by blanks or commas.
- Multiple parameters must be specified in the order in which they occur in the stored message.

DISPLAY

Purpose

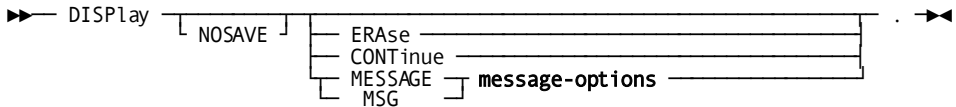
Displays a dialog's map, or reexecutes a dialog's premap process.

Additionally, DISPLAY can specify a message that appears in a map's message field. If a dialog has a map and a premap process, the premap process must include a DISPLAY command to display the map. If a DISPLAY command is not coded, nothing is written to the terminal at run time.

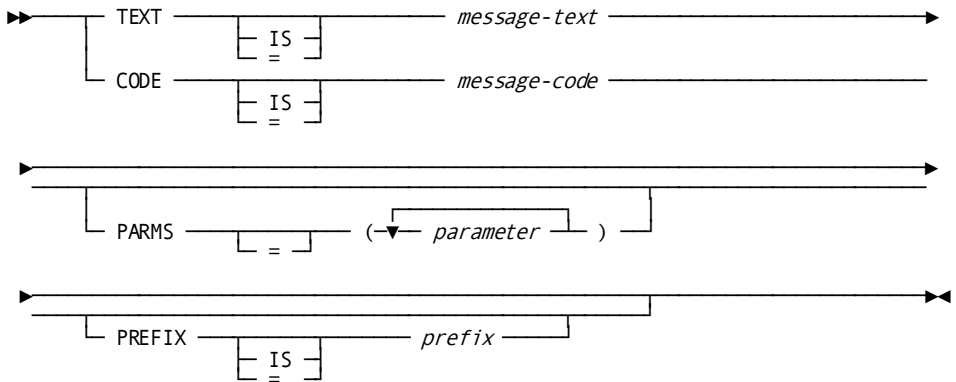
DISPLAY issued without the CONTINUE keyword, displays the map associated with the current dialog. DISPLAY can be used in a premap process or a response process.

In a pageable map, the detail occurrences that are displayed when the DISPLAY command is issued depend on the value of the system-defined data field \$PAGE and the number of detail occurrences that a single screen can hold. For example, given a screen that can hold ten detail lines, if \$PAGE equals 1, detail occurrences 1 through 10 are displayed; if \$PAGE equals 2, occurrences 11 through 20 are displayed; and so forth.

Syntax



Expansion of Message-Options



Parameters

NOSAVE

Specifies that currencies are not saved when control passes from the current process to the pseudo-converse or premap process. After the pseudo-converse, or when the premap process begins execution, the dialog's currencies are initialized to those of the next higher level dialog, if any.

ERAsE

Specifies that the following actions are performed at the terminal:

- Unprotected map data fields are cleared.
- The modified data tags (MDTs) for all unprotected map data fields are reset.
- The keyboard is unlocked.
- The cursor is placed at the first unprotected map data field.

If specified, ERASE is the only keyword that can follow DISPLAY in a DISPLAY command.

CONTInue

(Used in a response process) Requests reexecution of the premap process associated with the current dialog.

The keyword CONTINUE is ignored in a premap process.

MESSAge *message-options*

Identifies message to be displayed.

MSG can be used in place of MESSAGE.

Expanded syntax for *message-options* is shown above immediately following the CONTINUE syntax.

TEXT IS *message-text*

Specifies the text of a message to be displayed in an online map's message field or sent to a batch application and a system log file.

Message-text specifies either the name of a variable data field containing the message text or the text string itself, enclosed in single quotation marks.

The text string can contain up to 240 displayable characters.

IS or = are optional keywords and have no effect on processing.

CODE IS *message-code*

Specifies the message dictionary code of a message to be displayed in an online map's message field or sent to the log file in a batch application.

In a batch application, the message is also sent to the operator, if directed by the destination specified in the dictionary.

Message-code specifies either the name of a variable data field that contains the message code or the 6-digit code itself, expressed as a numeric literal.

IS or = are optional keywords and have no effect on processing.

PARMS = *parameter*

Specifies a replacement parameter for each variable field in the stored message identified by *message-code*.

Up to nine replacement parameters can be specified for a message. Multiple parameters must be specified in the order in which they are numbered and separated by blanks or commas.

Parameter specifies either the name of an EBCDIC or unsigned zoned decimal variable data field that contains the parameter value or the actual parameter value, enclosed in single quotation marks.

The parameter value must contain displayable characters. At run time, each variable data field in a stored message expands or contracts to accommodate the size of its replacement parameter. A replacement parameter can be a maximum of 240 bytes.

PREFIX IS *prefix*

Overrides the default prefix of a dialog and a map.

Prefix must either specify an EBCDIC or unsigned zoned decimal variable data field that contains a 2-character prefix or the 2-character prefix itself, enclosed in single quotation marks

IS or = are optional keywords and have no effect on processing.

Usage***Rules for Mapping Out Fields***

The DISPLAY command maps out literal fields and data fields according to these rules:

- If the map is different than the map previously displayed, both literal fields and data fields are mapped out.
- If the map is the same as the map previously displayed, literal fields are not mapped out. Data fields, except those set IN ERROR, are mapped out. Note that the MODIFY MAP command can be used to change the IN ERROR setting for a map field.

- If the ERASE keyword is specified, data fields are not mapped out. Instead, unprotected data fields on the screen are cleared.
- Data fields are further regulated by specifications made during map definition and by MODIFY MAP process commands. Both methods allow the specification that data is not displayed or is erased on a DISPLAY command.
- For a pageable map, if a PUT DETAIL command causes the first map page to be displayed, the following DISPLAY command does not map out literal or data fields. However, the DISPLAY command is still required to terminate the current process and create a pseudo-converse.

Specifying a Message

The DISPLAY command is also used to specify a message that is to appear in a map's message field.

Message fields are defined by the map field \$MESSAGE.

Note: For more information, see the *CA IDMS Mapping Facility Guide*.

One \$MESSAGE field can be defined anywhere on the map. If the \$MESSAGE field is defined in the detail area of a pageable map, the PUT DETAIL command is used to specify a message.

If a DISPLAY command specifies a message but the map has no message field, CA ADS creates a special message map.

Considerations for Specifying a Message Code

- Each message in the message dictionary is identified by a 6-digit code preceded by the letters DC. A request for message 987654 retrieves message DC987654.
User-defined messages added to the message dictionary should be identified by a code in the range 900001 through 999999 and preceded by letters other than DC.
- Each message in the message dictionary can be assigned a severity code. The severity code specifies the action that CA ADS takes when the message is retrieved. Severity codes are listed in the following table.

Message Dictionary Severity Codes

Severity code	Action
0	Processes the DISPLAY command
1	Snaps all CAADS resources and processes the DISPLAY command
2	Snaps all system areas and processes the DISPLAY command

Severity code	Action
3	Snaps all CAADS resources and terminates CA ADS with a task abend code of D002
4	Snaps all system areas and terminates CA ADS with a task abend code of D002
5	Terminates CA ADS with a task abend code of D002
8	Snaps all system areas and terminates the DC system with an operating system abend code of 3996
9	Terminates the DC system with an operating system abend code of 3996

A message in the message dictionary can contain one or more variable fields that are replaced with application-specific values at run time. In a DISPLAY command, the PARMs parameter can be used to code replacement parameters for each variable field in a specified message.

Within the message definition in the dictionary, symbolic parameters are identified by an ampersand (&) followed by a 2-digit numeric identifier. These identifiers can appear in any order. The position of the replacement values in the PARMs parameter must correspond directly to the 2-digit numeric identifiers in the message; the first value corresponds to &01, the second to &02, and so forth. For example, assume that the stored message text is as follows:

THIS IS TEXT &01 AND &03 OR &02

The PARMs parameter reads PARMs=('A','B','C'). The resulting text would read as follows:

THIS IS TEXT A AND C OR B

If the message is defined in the dictionary with more than one text line, only the first line appears in the map's message field.

If the message is defined in the dictionary with a destination of TERMINAL, the message will be redisplayed at the user's terminal when control exits from the CA ADS application.

Examples

The examples below are based on the sample applications shown in [Application Thread](#) (see page 327) where dialog UPDATEO updates or erases all ORDOR records associated with a CUSTOMER record that is retrieved by dialog UPDATEC.

Example 1: Retrieving records

The following sample premap process from UPDATEO retrieves the ORDOR records to be changed. The DISPLAY command is used to display the dialog's map with a message informing the user of the processing status:

```
READY.  
OBTAIN NEXT ORDOR WITHIN CUSTOMER-ORDER.  
IF DB-END-OF-SET  
THEN  
    DISPLAY MESSAGE TEXT IS  
    'CUSTOMER HAS NO ORDERS. HIT 'CLEAR' TO EXIT.'  
ELSE  
    DISPLAY MESSAGE CODE IS 900101  
    PARS = (ORD-NUMBER, 'ORDERS').
```

Example 2: Erasing records

The following sample response process from UPDATEO erases a retrieved ORDOR record. DISPLAY CONTINUE is used to return control to the dialog's premap process, which retrieves the next ORDOR record:

```
READY USAGE-MODE IS UPDATE.  
ERASE ORDOR ALL MEMBERS.  
DISPLAY CONTINUE.
```

More information:

[PUT DETAIL](#) (see page 474)

EXECUTE NEXT FUNCTION

Purpose

Passes control in a dialog that is associated with an application function.

Syntax

► — EXECute next function — . ————— ◄

Usage

EXECUTE NEXT FUNCTION is appropriate for use in applications defined by using the CA ADS application compiler (ADSA).

When the user selects a response that is valid for a dialog function at runtime, the function associated with the response is established as the next function to be executed. The EXECUTE NEXT FUNCTION command initiates execution of that function. Control is passed to the function by means of the control command associated with the application response during application compilation.

Considerations

- An EXECUTE NEXT FUNCTION command in a dialog that is not associated with an application function is processed by the CA ADS runtime system as a DISPLAY command. The runtime system displays the following message in the map's message field:

DC177018 PLEASE SELECT NEXT FUNCTION
- The EXECUTE NEXT FUNCTION command executes the function that is invoked by the application response specified in the AGR-CURRENT-RESPONSE field of the ADSO-APPLICATION- GLOBAL-RECORD. Note that the response is moved into AGR-CURRENT-RESPONSE when the user selects an application response.
- Premap and response process commands can modify the value of AGR-CURRENT-RESPONSE, thereby modifying the function executed by the EXECUTE NEXT FUNCTION command.
- The premap process of a mapless dialog must move a valid application response to AGR-CURRENT-RESPONSE before issuing an EXECUTE NEXT FUNCTION command.
- If AGR-CURRENT-RESPONSE is modified by a process command, the runtime system does not perform security checking.

Example

In this example, control passes to the next function in the CA ADS application after the end-of-set condition is reached:

```
WHILE NOT DB-END-OF-SET
  REPEAT.
    OBTAIN NEXT ORDOR WITHIN CUST-ORDOR.
    .
    .
    .
  END.
EXECUTE NEXT FUNCTION.
```

Because EXECUTE NEXT FUNCTION is used to pass control in this example, the CA ADS runtime system determines which function to execute next.

More information:

[Runtime Flow Of Control](#) (see page 135)

[ADSO-APPLICATION-GLOBAL-RECORD](#) (see page 568)

INVOKE

Purpose

Passes control to a specified dialog in the current application and implicitly establishes the next lower level in the application thread.

Syntax

```

▶—— INVoKe ———┬─── dialog-name ———▶
                  └─ NOSAVE ─┘

```

Parameters

NOSAVE

Specifies that database currencies are not saved for the dialog issuing the INVOKE command.

dialog-name

Specifies either the name of a variable data field containing the dialog name to which control passes or the dialog name itself, enclosed in single quotation marks.

Usage

Considerations

- The load module for the named dialog must be available at run time.
- The dialog that issues the INVOKE command remains operative.
- A lower level dialog can return control to the dialog by issuing a RETURN command.
- The issuing dialog's database currencies are saved and available to the dialog receiving control, unless the NOSAVE option is specified.

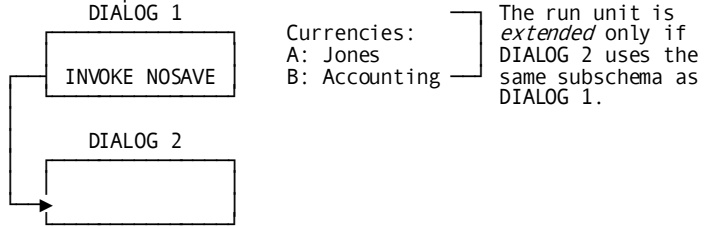
When a dialog that issued an INVOKE NOSAVE command regains control from a lower level dialog or program, database currencies are dependent upon whether or not the run unit was extended. The following diagram shows how currencies are affected when the NOSAVE option is used in extended and nonextended run units.

Currency Settings of Extended and Nonextended Run Units

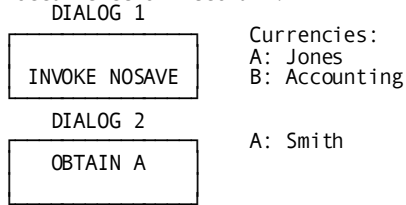
1. DIALOG 1, which uses subschema SS1, obtains values for records A and B:



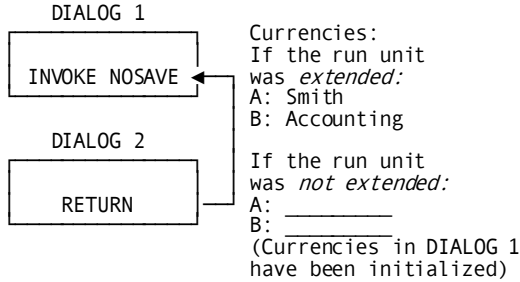
2. DIALOG 1 invokes DIALOG 2 using the NOSAVE option:



3. DIALOG 2, which can use DIALOG 1's record buffers and currencies, obtains a new occurrence of record A:



4. DIALOG 2 issues a RETURN to DIALOG 1:



Considerations for Using NOSAVE

- When the dialog that issues the command regains control from a lower level dialog and the run unit is extended by the INVOKE command, currencies are set to those of the most recent dialog returning control.
- When the dialog that issues the command regains control from a lower level dialog and the run unit is not extended by the INVOKE command, currencies are set to the original currencies available to the dialog when it became operative in the application thread.

Example

In the sample applications shown in [Application Thread](#) (see page 327), dialog RDCUST# prompts the user for the CALC key of a CUSTOMER record to be retrieved. RDCUST# passes control to dialog UPDATEC, which retrieves and displays the record, and then modifies or erases it as instructed by the user. RDCUST# uses the following response process to pass control to UPDATEC:

```
INVOKE 'UPDATEC'.
```

Because RDCUST# uses the INVOKE command to pass control, processing can return to the RDCUST# mapout operation following completion of UPDATEC processing. This allows the user to update multiple CUSTOMER records in one CA ADS runtime session.

More information:

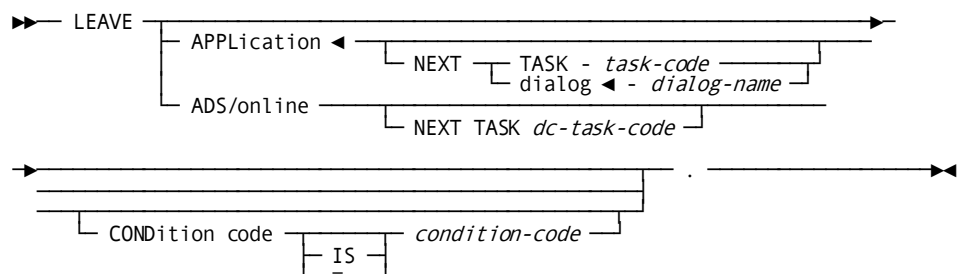
[CA ADS Runtime System](#) (see page 119)

LEAVE

Purpose

Terminates the current application thread or terminates the current CA ADS runtime session.

Syntax



Parameters

APPLiCation

Terminates the current application and passes control as specified by NEXT TASK or NEXT dialog.

LEAVE is the equivalent of LEAVE APPLICATION.

NEXT TASK *task-code*

Passes control to an application as defined on the Task Codes screen of the application compiler.

Task-code specifies an application task code, as defined on the **Task Codes** screen of the application compiler. *Task-code* is either the name of a variable field containing the task code or the task code itself, enclosed in single quotation marks.

NEXT dialog *dialog-name*

Specifies the name of a mainline dialog to which control passes. If the keyword TASK or dialog is not specified, dialog is the default.

Dialog-name is either the name of a variable data field that contains the dialog name or the dialog name itself, enclosed in single quotation marks.

The load module for the named dialog must be stored in the data dictionary.

ADS/online

Terminates the current application and the current CA ADS session. Control returns to CA IDMS/DC or DC/UCF (DC/UCF).

NEXT TASK *dc-task-code*

(Online only) Passes control to another DC/UCF task.

Dc-task-code is either the name of a variable field containing the DC/UCF task or the task name itself, enclosed in single quotation marks.

Dc-task-code must be defined with the NOINPUT parameter, which specifies that only a task code, and no additional data, is expected.

CONDition code IS *condition-code*

(Batch z/OS only) Clause introducing a completion code for the current job step.

The completion code can be tested using the COND parameter in the job control language (JCL).

Condition-code is either the name of a variable field containing the condition code or the number itself, expressed as a numeric literal.

IS or = are optional keywords and have no effect on processing.

Usage*Effects of Issuing LEAVE*

- All operative dialogs in the terminating application become nonoperative.
- All database currencies are released.
- All record buffers are freed.

Example

Dialog UPDATEC, shown in Example 1 in the earlier diagram, includes the following response process, which allows the terminal operator to terminate the application thread:

```
LEAVE APPLICATION.
```

When the above response process executes, control passes to the Dialog Selection screen. The user can then select the next mainline dialog to be executed.

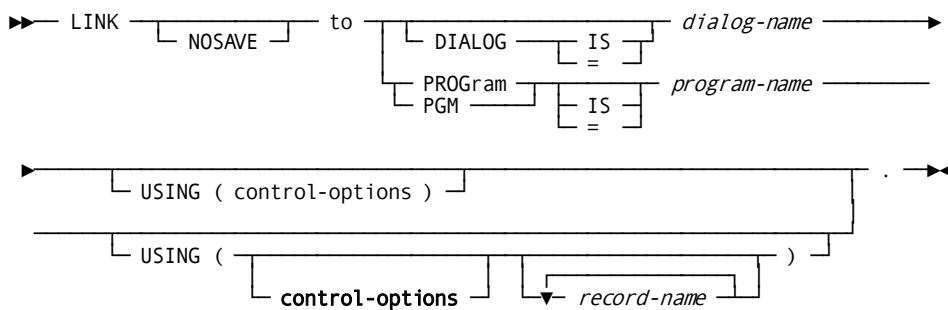
LINK

Purpose

Specifies the next dialog executed in a current application.

LINK is also used to request execution of a COBOL, PL/I, or Assembler program.

Syntax



Expansion of Control-Options



Parameters

NOSAVE

Specifies that the database currencies for the dialog that issues the LINK command are not saved.

When a dialog that issues a LINK NOSAVE command regains control from a lower level dialog or program, its database currencies are set as follows:

- If the LINK command extends a run unit, the dialog's currencies are passed back up to the dialog or program to which the linking dialog passed control.
- If the LINK command does not extend a run unit, the dialog's database currencies are reinitialized to whatever they were when the dialog gained control.

DIALOG IS *dialog-name*

Specifies the name of the dialog to which control passes.

Dialog-name is either the name of a variable data field that contains the dialog name or the dialog name itself, enclosed in single quotation marks.

The load module for the named dialog must be stored in the data dictionary of load library.

IS or = are optional keywords and have no effect on processing.

PROGram IS *program-name*

Specifies the name of the COBOL, PL/I, or Assembler program to which control is passed.

Program-name is either the name of a variable data field that contains the program name or the program name itself, enclosed in single quotation marks.

The load module for the named program must be defined under DC/UCF as a program. The program can be defined in any of the following ways:

- At system generation by means of the PROGRAM statement
- In the IDD by means of the DDDL PROGRAM statement
- Under DC/UCF by means of the DCMT VARY DYNAMIC PROGRAM master terminal command

PGM can be used in place of PROGRAM.

USING *control-options*

Identifies the control options to be used.

Expanded syntax for *control-options* is shown above immediately following the LINK syntax.

Multiple parameters in the USING clause must be separated by blanks or commas. The record and control block names must be specified in the same order in which they are defined in the user program. If used, the USING clause must specify at least one record or control block name.

The SQLSSI parameter is used when passing a global cursor from an CA ADS dialog to a user program. SQLSSI is a record that contains the SQL session identifier which is assigned when the dialog's transaction started. This record is copied into the dialog automatically, so the user does not need to add it to the dialog. The user program must have a record in its "linkage" section defined with the SQLSESS datatype.

record-name

Specifies the data that is passed to the named user program.

MAP-CONTROL

Passes the map request block of the original CAADS dialog to the lower level dialog.

The lower level dialog must specify the same map as the calling dialog. The version number and date/time stamp for both maps must be identical. If the maps differ, the application abends.

The keyword MAP_CONTROL may be used in place of MAP-CONTROL.

SUBSCHEMA-CONTROL

Extends a calling dialog's run unit to a lower level dialog. The runtime system ignores any differences between the two dialog's subschemas, schemas, and area ready modes. On return to the calling dialog, the run unit is unconditionally extended upward.

The dialog to which control is extended is not allowed to access a record or set not defined in the original dialog's subschema. Such an attempt causes an abend at runtime.

The keyword SUBSCHEMA_CONTROL may be used in place of SUBSCHEMA-CONTROL.

Usage***Control Passed to a Specified Dialog***

When control is passed to a specified dialog by means of a LINK command, the next lower level in the application thread is implicitly established and a nested structure is created.

Considerations

- The dialog issuing the LINK command becomes the top of the nested structure and remains operative.

If an application response passes control by means of a LINK command, the function from which the response was selected becomes the top of a nested structure.

- A LINK command within a nested structure establishes the top of a lower nested level.
- Dialogs within a nested structure can issue any of the control commands.
A RETURN command cannot pass control higher than the top of the lowest nested level that is operative in the application thread.
- The dialog issuing a LINK command expects control to return to the command following the LINK instruction.
- The issuing dialog's database currencies are saved and are available to the dialog when it regains control, unless the NOSAVE option is used.

When the dialog that issued a LINK NOSAVE command regains control from a lower level dialog or program, the database currencies set depend on whether or not the run unit was extended.

Refer to the LINK command syntax rules that follow this discussion for currency settings of extended and nonextended run units.

Control Passed to a User Program

When a LINK command specifies a user program, control passes outside the CA ADS environment and temporarily suspends CAADS sessions.

Considerations

- The LINK command must explicitly specify any data to be passed to the user program, including the subschema control block, the map request block, and any records used in the program's processing.
- A user program has the option of using the calling dialog's run unit.

If the LINK command does not contain subschema-control in its USING list, the user program cannot access its calling dialog's run unit. The user program can access a database by binding a run unit and establishing its own currencies. This run unit will be bound concurrently with the dialog's run unit.

If the LINK command contains subschema-control in its USING list, the dialog's run unit is passed to the user program. Any database records to be shared with the dialog should be passed in the USING RECORD list.

- A user program must return control to CA ADS by means of a DC RETURN statement.

When the user program issues the DC RETURN statement, the suspended CA ADS session resumes and control passes to the command following the LINK command.

The format of the DC RETURN statement varies based on whether the program has previously issued a DC RETURN statement that specified a next task code other than ADSR, as follows:

- If the program has previously issued a DC RETURN statement that specified a next task code other than ADSR, the DC RETURN statement that returns control to CA ADS must have the following format:

```
DC RETURN NEXT TASK CODE ADSR.
```

DC RETURN NEXT TASK CODE will end the task and rollback any open run unit, whether it was bound by the user program or passed from the calling dialog.

ADSR is the default task code that invokes ADSOMAIN with no input. The task code can be changed by means of the DC/UCF system generation TASK statement.

Note: For more information about specifying the task code for the CA ADS runtime system, see the *CA IDMS System Generation Guide*.

- If the program has not previously issued a DC RETURN statement that specified a next task code other than ADSR, the DC RETURN statement that returns control to CA ADS can have the following format:

```
DC RETURN
```

- A dialog with a standard subschema can link to a dialog with an LRF subschema using subschema control. However, if the lower-level dialog makes an LR call, a status of 0063 is returned; in this case, the status is equivalent to a status of 2008.

To use the LINK command effectively in conjunction with user programs, refer to the online programming techniques presented in the *CA IDMS DML Reference* for the appropriate language.

Example 1: Passing control to a lower level dialog

Dialog UPDATEC, shown in Example 1 of [Application Thread](#) (see page 327), uses the response process listed below to pass control to dialog UPDATEO.

UPDATEO obtains an ORDOR record for the current CUSTOMER, requests modifications, and updates the record in the database. When UPDATEO returns control to dialog UPDATEC, processing resumes with the DISPLAY command that follows the LINK command:

```
LINK TO DIALOG 'UPDATEO'.
DISPLAY MESSAGE TEXT IS
'CUSTOMER ORDER HAS BEEN CHANGED'.
```

Example 2: Passing control to a COBOL program

The following statement from the premap process associated with dialog UPDATEC passes control to the COBOL program LOOKUP. LOOKUP uses the subschema control block and CUSTOMER record buffer from UPDATEC to check the status of the current customer:

```
LINK PROGRAM 'LOOKUP'  
USING (SUBSCHEMA-CONTROL,CUSTOMER).
```

Example 3: Extending the current map session

In this example, ERRCHK is a dialog that contains special error-checking and validating routines. ERRCHK uses the same map as the calling dialog. The LINK command passes current map attributes and data to ERRCHK.

When ERRCHK finds errors, it:

- Sets the appropriate fields in error by modifying error attributes for the map.
- Returns control to the calling dialog. The error attributes are returned along with current map data.

The sample LINK statement that passes control to ERRCHK is:

```
LINK TO DIALOG 'ERRCHK'  
USING (MAP-CONTROL).
```

Example 4: Extending the current run unit

In this example:

- The calling dialog uses subschema EMPSS01. This subschema contains records EMPLOYEE and DEPARTMENT.
- The LINK command unconditionally extends the current run unit to dialog UPDATE, which is a mapless dialog containing update logic for records EMPLOYEE and DEPARTMENT.
- The USING statement bypasses the checking of the subschema and ready modes when passing the run unit.
- Dialog UPDATE updates the database and then returns control to the calling dialog.

The sample LINK statement that passes control to UPDATE is:

```
LINK TO DIALOG 'UPDATE'  
USING (SUBSCHEMA-CONTROL).
```

Example 5: COBOL program that was passed the dialog's subschema-control

The following example shows a LINKed-to COBOL program that was passed the dialog's SUBSCHEMA-CONTROL.

```
ENVIRONMENT DIVISION.  
  PROTOCOL. IDMS-RECORDS MANUAL.  
WORKING-STORAGE SECTION.  
  01 COPY IDMS SUBSCHEMA-NAMES.  
  01 COPY IDMS RECORD <the name of each database record that is  
                        needed but was not passed in the USING list>  
LINKAGE SECTION.  
  01 COPY IDMS SUBSCHEMA-CTRL.  
  01 COPY IDMS RECORD <the name of each database record that is  
                        passed in the LINK command>  
PROCEDURE DIVISION.  
  BIND <the name of each database record that is needed but was  
        not passed in the LINK command>
```

Example 6: BAI program that was passed the dialog's subschema-control

The following example shows a LINKed-to Basic Assembler Language program that was passed the dialog's SUBSCHEMA-CONTROL and DIALOG record ADSO-APPLICATION-GLOBAL-RECORD.

```
ADSPGM TITLE 'PROGRAM CALLED FROM AN ADS DIALOG'
*GETBTMAP RENT EP=GBMPEP1 XA
COPY #APGDS
COPY #CSADS
COPY #TCEDS
COPY #SSCDS
#MOPT CSECT=ADSPGM,ENV=USER
@MODE MODE=IDMSDC,WORKREG=R0,QUOTES=YES,DEBUG=YES
*At entry, R9 contains the address of the TCE
USING TCE,R9
*At entry, R10 contains the address of the CSA
USING csa,R10
ENTRY ADSPGEP1
ADSPGEP1 DS OH
*-----
*Set base register
*-----
LR,R12,R15 BASE THIS MODULE
USING ADSPGEP1,R12
*Registers at entry need not be saved
*At entry, R1 contains the address of the passed parameter list
*Accept ADSO-APPLICATION-GLOBAL-RECORD as a passed parameter
L R6,0(,R1) A(SSCTRL)
USING APG,R6
* Accept SUBSCHEMA-CONTROL as a passed parameter
L R7,4(,R1) A(SSCTRL)
LA R7,0(,R7) CLEAR HIGH-ORDER FLAG
USING SSCTRL,R7
SPACE
#GETSTG TYPE=(USER,SHORT),PLIST=*,X
LEN=12,X
ADDR=(R2),X
INIT=0
* Necessary DML commands here
EJECT
*-----
*Return to caller
*-----
#RETURN
LTORG
END
```


More information:

[CA ADS Runtime System](#) (see page 119)

READ TRANSACTION

Purpose

(CA ADS/Batch only) Terminates the current process, performs a mapin operation, and then selects the next application function or response process to be executed.

Syntax

```
▶▶ READ TRANsaction [ OUTPUT ] . ▶▶
```

Parameters

OUTput

Specifies that the file is to be opened as an input/output file.

Usage

Considerations

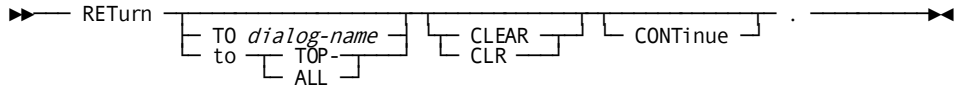
- OUTPUT must be specified in the first READ TRANSACTION command for a VSAM entry-sequenced data set (ESDS) that is to be opened for both input and output.
- OUTPUT is ignored if the file is already opened; if the file is not a VSAM ESDS file, the application abends.
- If the current record's response field selects an immediately executable function that is not the same as the current function, the runtime system passes control to the newly selected function. The next time a mapin operation is performed for the file, the runtime system immediately maps in the record.
- On a mapin operation, the runtime system automatically opens the file being read if the file is not already opened.

RETURN

Purpose

Passes control to a higher level dialog or function in the application thread.

Syntax



Parameters

TO *dialog-name*

Introduces the name of a higher level dialog to which control passes.

Dialog-name can be the name of a variable data field that contains the dialog name or the dialog name itself, enclosed in single quotation marks.

to TOP

Specifies the highest level to which control can pass.

ALL can be used in place of to TOP.

CLEAR

Specifies that record buffers are reinitialized and currencies are released for the dialog receiving control.

CLEAR is ignored if the receiving dialog is at the top of a nested application structure.

CLR can be used in place of CLEAR.

CONTInue

Specifies that control returns to the first command in the premap process of the dialog receiving control.

If CONTINUE is not specified, control returns to the mapout operation of the dialog that receives control. If the receiving dialog is at the top of a nested application structure, CONTINUE is ignored.

Usage

Considerations

- The dialog or function receiving control must be operative.
- The dialog that issues the RETURN command becomes nonoperative as do any operative dialogs or functions on a level between the issuing and receiving dialogs or functions.
- All database currencies established by the dialog issuing the RETURN command are released.
- A RETURN command cannot pass control higher than the top of the lowest level nested application structure created by a LINK command.

- The named dialog must not be higher than the top of the nested application structure in which the issuing dialog participates.
- If the named dialog is operative at more than one higher level, control passes to the lowest level operative dialog with the specified name.
- If the issuing dialog participates in a nested application structure, control returns to the top of the nested structure.
- If the issuing dialog does not participate in a nested structure, control returns to the mainline dialog at the top of the application thread.
- If a RETURN statement does not specify a receiving dialog, control passes to the next higher level dialog or function.
- If the mainline dialog at the top of an application thread issues a RETURN command, the RETURN command is treated as a LEAVE APPLICATION command.
- A dialog that receives control at the top of a nested structure resumes execution at the command that follows the LINK command.
- RETURN can pass control within a nested application structure to any operative dialog that passed control with an INVOKE command.
- If the issuing dialog is not in a nested structure, RETURN can pass control to any higher level operative dialog or function, or directly to the top of the application structure.
- The application developer can specify whether the dialog receiving control resumes execution with its premap process or with its mapout operation.
- The application developer can also request reinitialized record buffers for the dialog that receives control.

Examples

The examples below show the use of the RETURN command in response processes from dialogs used in the two sample applications shown in [Application Thread](#) (see page 327):

Example 1: Using RETURN with the LINK command

In Example 1 of [Application Thread](#) (see page 327), dialog UPDATEC passes control to the dialog UPDATEO by means of a LINK command. The following response process from dialog UPDATEO returns control to the command following the LINK command in dialog UPDATEC:

```
READY USAGE-MODE IS UPDATE.  
MODIFY ORDOR.  
RETURN.
```

Example 2: Using RETURN with the INVOKE command

In Example 2 of [Application Thread](#) (see page 327), dialog UPDATEC passes control to the dialog UPDATEO by means of an INVOKE command. The following response process from dialog UPDATEO returns control to the mainline dialog RDCUST# and reinitializes the record buffers associated with RDCUST#:

```
READY USAGE-MODE IS UPDATE.  
MODIFY ORDOR.  
RETURN TOP CLEAR.
```

Example 3: Transferring control within the same level

In Example 2 of [Application Thread](#) (see page 327), dialog UPDATEO provides the ability to transfer to dialog ADDORDR. ADDORDR prompts the user for new order information. The following response process from dialog ADDORDR adds a new ORDOR record to the database and returns control to the mapout operation of dialog UPDATEC:

```
READY USAGE-MODE IS UPDATE.  
STORE ORDOR.  
RETURN.
```

TRANSFER

Purpose

Passes control to a specified dialog at the same level in the application structure.

Syntax

```
►► TRANSfer [ NOFinish ] to dialog-name . ◄◄
```

Parameters**NOFinish**

Specifies that the current run unit is to be extended.

to *dialog-name*

Either the name of a variable data field that contains the dialog name to which control passes or the dialog name itself, enclosed in single quotation marks.

The load module for the named dialog must be stored in the data dictionary.

Usage

Considerations

- When specified along with the NOFINISH option, TRANSFER can extend the current run unit.
- When TRANSFER is specified without NOFINISH, a dialog that issues a TRANSFER command becomes nonoperative.
- The receiving dialog or function replaces the issuing dialog in the application thread.
- The receiving dialog or function has access to database currencies established by dialogs at higher levels in the application thread and to the contents of global records and of any records whose buffers were established by dialogs at higher levels in the application thread.
- A dialog can transfer control to itself.
- The copy of the dialog receiving control acquires newly initialized record buffers.
- When a dialog transfers control to itself, the FIRST-TIME status is reset.

Examples

The following examples use the TRANSFER statement to pass control to a dialog at the same level.

Example 1: Using the dialog name

In Example 2 of [Application Thread](#) (see page 327), dialog UPDATEO passes control to dialog ADDORDR by means of the following statement:

```
TRANSFER TO 'ADDORDR' .
```

Example 2: Using a variable data field to transfer control

In this example, control passes either to dialog ADDORDR or to dialog ORDCOUNT, depending on the outcome of the OBTAIN command:

```
OBTAIN NEXT ORDOR WITHIN CUST-ORDER.  
IF DB-END-OF-SET  
THEN  
    MOVE 'ADDORDR' TO NEXT-DIALOG.  
ELSE  
    MOVE 'ORDCOUNT' TO NEXT-DIALOG.  
TRANSFER TO NEXT-DIALOG.
```

More information:

- [CA ADS Runtime System](#) (see page 119)
- [Conditional Expressions](#) (see page 245)

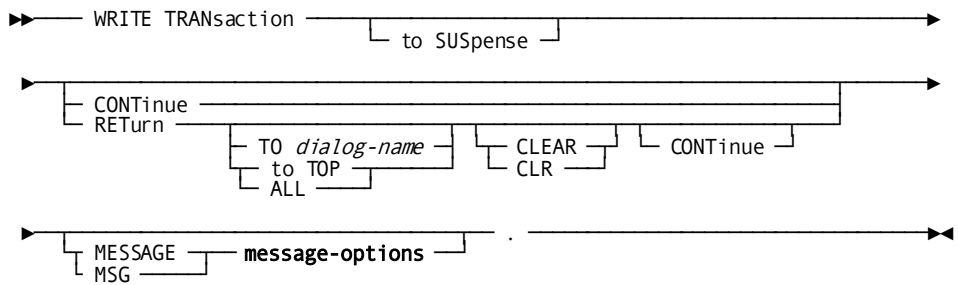
WRITE TRANSACTION

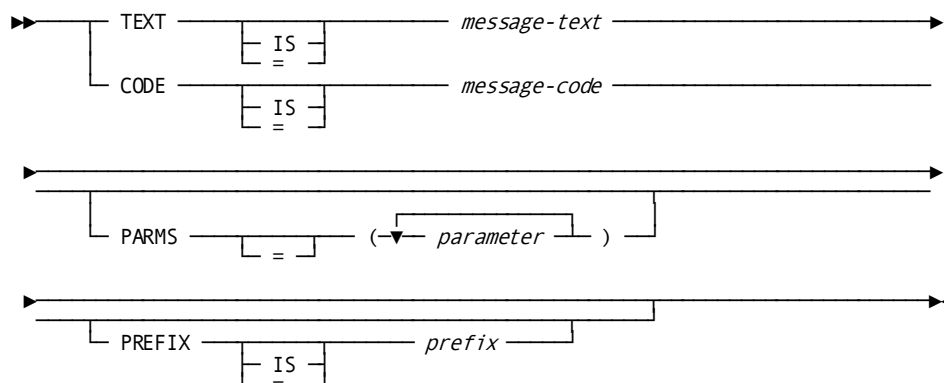
Purpose

(CA ADS/Batch only) Performs the following sequence of functions:

1. Terminates the current process
2. Performs a mapout operation
 - If the dialog's current input file record contains no errors and the keyword SUSPENSE is not included in the command, the mapout writes a record to the dialog's associated output file, according to the output file map definition.
 - If the input file record contains errors or the keyword SUSPENSE is included in the command, the mapout writes the input record to the dialog's suspense file and sends applicable error messages to the log file.
3. Passes control within the application. Control can be passed to:
 - The dialog's premap process or mapin operation
 - A higher level dialog or application function

Syntax



Expansion of Message-Options**Parameters****to SUSPense**

Specifies that the dialog's input record is written to the suspense file even if it does not contain errors. Nothing is written to the dialog's output file.

CONTinue

Specifies that control is passed to the dialog's premap process after mapout operation.

RETurn

Specifies that control is returned to a higher level dialog or application function after mapout operation.

TO *dialog-name*

Either the name of a variable data field that contains the dialog name to which control is passed or the dialog name itself, enclosed in single quotation marks.

to TOP

Specifies the highest level to which control can pass.

ALL can be used in place of to TOP.

CLEAR

Specifies that record buffers are reinitialized and currencies are released for the dialog receiving control.

CLEAR is ignored if the receiving dialog is at the top of a nested application structure.

CLR can be used in place of CLEAR.

CONTINUE

Specifies that control returns to the first command in the premap process of the dialog receiving control. If not specified, control returns to the mapout operation of the dialog that receives control. If the receiving dialog is at the top of a nested application structure, CONTINUE is ignored.

If neither CONTINUE nor RETURN is specified, control passes to the dialog's mapin operation. The runtime system maps the next record into variable storage, then selects the next application function or dialog response process to be executed.

Note: For applications defined using the application compiler, the runtime system first examines the current record's response field. If the field selects an immediately executable function that is not the same as the current function, the runtime system passes control to the selected function. The next time a mapin operation is performed for the file, the runtime system immediately maps in the record.

MESSAGE *message-options*

Identifies message to be displayed.

MSG can be used in place of MESSAGE.

Expanded syntax for *message-options* is shown above immediately following the CONTINUE syntax.

TEXT IS *message-text*

Specifies the text of a message to be displayed in an online map's message field or sent to a batch application and a system log file.

Message-text specifies either the name of a variable data field containing the message text or the text string itself, enclosed in single quotation marks.

The text string can contain up to 240 displayable characters.

IS or = are optional keywords and have no effect on processing.

CODE IS *message-code*

Specifies the message dictionary code of a message to be displayed in an online map's message field or sent to the log file in a batch application.

In a batch application, the message is also sent to the operator, if directed by the destination specified in the dictionary.

Message-code specifies either the name of a variable data field that contains the message code or the 6-digit code itself, expressed as a numeric literal.

IS or = are optional keywords and have no effect on processing.

PARMS = *parameter*

Specifies a replacement parameter for each variable field in the stored message identified by *message-code*.

Up to nine replacement parameters can be specified for a message. Multiple parameters must be specified in the order in which they are numbered and separated by blanks or commas.

Parameter specifies either the name of an EBCDIC or unsigned zoned decimal variable data field that contains the parameter value or the actual parameter value, enclosed in single quotation marks.

The parameter value must contain displayable characters. At run time, each variable data field in a stored message expands or contracts to accommodate the size of its replacement parameter. A replacement parameter can be a maximum of 240 bytes.

PREFIX IS *prefix*

Overrides the default prefix of a dialog and a map.

Prefix must either specify an EBCDIC or unsigned zoned decimal variable data field that contains a 2-character prefix or the 2-character prefix itself, enclosed in single quotation marks

IS or = are optional keywords and have no effect on processing.

Usage*Considerations*

- The named dialog must not be higher than the top of a nested application structure in which the issuing dialog participates.
- If the named dialog is operative at more than one higher level, control passes to the lowest level dialog with the specified name.
- If the write operation results in a physical output-error condition, the application terminates.
- A WRITE TRANSACTION command can be issued in a dialog that is not associated with an output file. In this case, the command is used only to write an input record to the suspense file. If the input record is not in error, nothing is written to the suspense file.
- The WRITE TRANSACTION command also allows specification of a message to be sent to the log file or to the operator's console.
- The destination of the message depends on the routing codes specified using ADSOBSYS or at run time in a control statement.

- If the issuing dialog participates in a nested application structure, control returns to the top of the nested structure.
- If the issuing dialog does not participate in a nested structure, control returns to the mainline dialog at the top of the application thread.
- If a RETURN statement does not specify a receiving dialog or TOP, control passes to the next higher level dialog or function.
- If the mainline dialog at the top of an application thread issues a RETURN command, the RETURN command is treated as a LEAVE APPLICATION command.
- Up to nine replacement parameters can be specified for a message.
- Multiple parameters must be separated by blanks or commas.
- Multiple parameters must be specified in the order in which they occur in the stored message.

Chapter 16: Database Access Commands

This section contains the following topics:

[Overview](#) (see page 363)

[Navigational DML](#) (see page 365)

[Logical Record Facility Commands](#) (see page 433)

Overview

An CA ADS application can access the CA IDMS/DB database by using navigational DML or SQL DML.

Navigational DML

CA ADS navigational DML is used to retrieve and update database or VSAM records and perform database control functions. Navigation DML commands can be used in process logic to store, retrieve, modify, and delete data in a non-SQL defined database, using a standard subschema or a Logical Record Facility (LRF) subschema.

When using LRF, the application developer selects a predefined path that meets the dialog's data requirements and codes simple database requests in dialog process logic. Database navigation is defined in the path, not in the process.

SQL DML

In an CA ADS application, SQL DML can be used to retrieve and update data defined with:

- Records in non-SQL defined databases (associated with an SQL schema)
- Tables in SQL-defined databases

Note: For more information about using SQL DML statements, see the *CA IDMS SQL Self-Training Guide* and the *CA IDMS SQL Programming Guide*.

Navigational DML and LRF commands used in the CA ADS environment are summarized in the following two tables. Documentation of command syntax appears later in this chapter.

Summary of Navigational DML Commands

Command	Purpose
ACCEPT	Moves database keys page information statistics from the database management system to a dialog
BIND PROCEDURE	Establishes communication from a dialog to a DBA-written procedure
COMMIT	Writes checkpoints to the journal file and releases locks held on database records
CONNECT	Connects member records to sets
DISCONNECT	Disconnects member records from sets
ERASE	Erases records from the database
FIND	Locates records in the database
GET	Copies record contents from the database to a dialog's record buffers
KEEP	Places locks on records
MODIFY	Replaces records in the database with the contents of a dialog's record buffers
OBTAIN	Locates records in the database and copies their contents to a dialog's record buffers
READY	Prepares database areas for processing
RETURN DB-KEY	Retrieves index entries without the associated record (used only with the Sequential Processing Facility and with system-owned indexed records)
ROLLBACK	Requests recovery of the database
STORE	Adds a record to the database

Summary of LRF Commands

Command	Purpose
ERASE	Deletes Logical Record Facility record occurrences
MODIFY	Changes field values in Logical Record Facility record occurrences
OBTAIN	Retrieves Logical Record Facility record occurrences
ON	Performs additional processing based on the outcome of conditional testing of Logical Record Facility record access

Command	Purpose
STORE	Stores a new occurrence of a Logical Record Facility record

Navigational DML

Each navigational DML command is presented alphabetically after the overview of navigational database access.

Overview of Navigational Database Access

To use navigational DML commands effectively in a process, the application developer should be familiar with database programming concepts. These concepts are discussed in detail in the *CA IDMS Navigational DML Programming Guide*.

Considerations

The following special considerations apply to accessing the database in the CA ADS environment:

- Before coding database commands, the application developer must be familiar with the characteristics of the subschema associated with the dialog. The subschema specifies the elements, records, sets, and areas available to the dialog. The subschema also includes the default usage modes for the database areas and specifies any restrictions on the use of database commands.
- The default usage mode for a database area can be specified with the FORCE option. In some cases, using the FORCE option enables adding an area to a subschema without recompiling the ADS dialogs that use it. However, we recommend not to use FORCE with ADS applications that use extended run units. In such cases, a forced automatic READY might ready the area in the default usage mode on the 1st dialog, but a lower-level dialog might need to READY the area in a more restrictive mode. This situation can lead to an unexpected failure of the ADS application.

Note: For more information on the limitations of using the FORCE option with ADS dialogs, see the Area Statement section (in the Subschema Statements chapter) in the *CA IDMS Database Administration Guide*.

- Each database command can be coded any number of times within a process.
- If a READY command specifies the same area more than once within a process, the usage mode specified in the last READY statement applies to the specified area for the entire process.

The READY command is executed when the first DML command is encountered. If an invalid (non-zero) error status is returned from the READY or BIND processing, the dialog aborts. Process code cannot intercept these errors.

- At runtime, CA ADS automatically initializes a buffer for each record type associated with the mainline dialog. Subsequent dialogs that access the same record type use the existing record buffer unless a reinitialized buffer for the record is requested by using the Records and Tables screen during dialog compilation.
- To enable proper positioning and movement through the database during the execution of an application, the CA ADS runtime system automatically maintains database keys for the records that are accessed by a dialog as shown in the following table.

Record	Description
Current of run unit	The most recently accessed record occurrence
Current of record type	The most recently accessed occurrence of each record type
Current of set type	The most recently accessed record occurrence (owner or member) of each set
Current of area	The most recently accessed record occurrence in each area

- Database commands use and update currencies, as listed in the currency chart below.

CA ADS saves or releases the currencies established during dialog execution based on the command used to pass control to the next function or dialog.

Database command	Currency updated by successful execution				Successful execution
	Run unit	Record	Set	Area	
ACCEPT*	X	X	X	X	None
IF*	X		X		None
FIND/OBTAIN DB-KEY					All
FIND/OBTAIN CURRENT*	X	X	X	X	All
FIND/OBTAIN WITHIN SET			X		All
FIND/OBTAIN WITHIN AREA				X ²	All

Database command	Currency updated by successful execution				Successful execution
FIND/OBTAIN OWNER			X		All
FIND/OBTAIN CALC					All
FIND/OBTAIN DUPLICATE			X		All
FIND/OBTAIN USING SORT KEY			X		All
GET	X				None
STORE			X ³		All
MODIFY	X				None ⁴
ERASE	X				Nullifies of all record types and sets involved
CONNECT		X	X		Run unit, set
DISCONNECT		X			Nullifies currency of object set; updates current of run unit and area
KEEP *	X	X	X	X	None
COMMIT					None
COMMIT ALL					Nullifies all currencies
ROLLBACK					Nullifies all currencies
ROLLBACK CONTINUE					Nullifies all currencies
FINISH					Nullifies all currencies

Note:

* Uses only one currency as determined by command format.

² Required for NEXT and PRIOR formats only.

³ All in which record type participates as an automatic member.

⁴ Except in the case of a sorted set.

More information:

[Control Commands](#) (see page 325)

[Records and Tables Screen](#) (see page 111)

Use of Native VSAM Data Sets

Native VSAM data sets can be defined in an CA IDMS/DB database schema and accessed by CA ADS database commands as if they were standard database files. CA IDMS/DB supports all three types of VSAM data sets: key sequenced (KSDS), entry sequenced (ESDS), and relative record (RRDS).

Existing VSAM data structures are accessed by equating them to CA IDMS/DB structures in the schema. The dialog issues standard process command statements for the equivalent CA IDMS/DB structures and the DBMS converts these statements to native VSAM access requests for the appropriate VSAM structures.

When a dialog's subschema includes records in a native VSAM file, process code for the dialog is affected in the following ways:

- The set status condition cannot be used with sets defined for native VSAM data records.
- Some database commands are affected, as listed below.

The following table lists considerations that apply to specific database commands when using native VSAM data sets.

Database Commands and Native VSAM Data Sets

Command	Consideration
ACCEPT DB-KEY RELATIVE TO CURRENCY	Next, prior, and owner currency cannot be requested for sets defined for native VSAM records.
CONNECT	The CONNECT command is not allowed because all sets in native VSAM data sets must be defined as mandatory automatic.
DISCONNECT	The DISCONNECT command is not allowed because all sets in native VSAM data sets must be defined as mandatory automatic.
ERASE	ERASE record-name is the only form of the ERASE command that is valid for use with native VSAM data sets. No form of the ERASE command is permitted against records contained in an ESDS.

Command	Consideration
FIND/OBTAIN DB-KEY	The FIND/OBTAIN DB-KEY command cannot be used to access records in a native VSAM KSDS because the database key does not necessarily remain static in a KSDS.
FIND/OBTAIN OWNER	The FIND/OBTAIN OWNER command is not allowed because owner records are not defined in native VSAM data sets.
FIND/OBTAIN WITHIN SET/AREA	When an end-of-set or end-of-area condition occurs, all currencies remain unchanged. The FIRST, LAST, and sequence-vn WITHIN AREA options cannot be used to access spanned data records in a native VSAM data set.
MODIFY	The length of a record in an ESDS file cannot be changed even if the record is variable length. The prime key for a KSDS cannot be modified.
STORE	If the object record is to be stored in a native VSAM RRDS, the DIRECT-DBKEY field must be initialized with the relative record number of the record being stored.

More information:

[Conditional Expressions](#) (see page 245)

Record Locking

Record locks are used to protect the integrity of database records.

Share and Exclusive Locks

Record locks protect object records from concurrent access or update by other run units. Locks can be shared or exclusive:

- **Shared record locks** allow other run units to access but not update the locked record.
- **Exclusive record locks** prohibit other run units from accessing the locked record as long as the lock is maintained.

Implicit and Explicit Record Locks

Record locks can be set implicitly by the DC/UCF central version and explicitly by the application developer, as follows:

- **Implicit record locks** are maintained automatically for every run unit that executes in shared update usage mode. Usage modes are discussed in 'READY' later in this section.
- **Explicit record locks** are set by means of a KEEP command or the KEEP clause of a FIND/OBTAIN command. FIND/OBTAIN is described later in this section.

Long-term explicit record locks are shared or exclusive record locks that are maintained across run units. A long-term lock placed on a record restricts other concurrently executing run units from accessing or updating the record until the lock is explicitly released. Subsequent run units in the same CA ADS application that execute from the same terminal can access and update the locked record, and can upgrade or release the long-term lock.

Note: For more information about record locks, see the *CA IDMS Database Design Guide*.

The following conditions resulting from the use of record locks can cause abnormal termination of an CA ADS application:

- **Too many locks**— Abnormal termination of an CA ADS application occurs if a run unit tries to generate more record locks than the maximum number specified at DC/UCF system generation. To lessen the possibility of abnormal termination because of too many locks, a COMMIT command can be used to release locks.
- **Wait time**— Abnormal termination of an CA ADS application occurs if the internal wait time of a run unit exceeds the wait interval specified at DC/UCF system generation.
- **Deadlock**— Abnormal termination of a run unit occurs when two run units would cause a deadlock by being permitted to wait to set locks. The run-unit that would complete the deadlock terminates, control returns to the issuing task, and a minor code 29 is returned.

An online application can include logic that is invoked if the run unit is terminated because of a db-key deadlock. In this way, the application can maintain the terminal session and save data previously entered on the screen. The application can then ask the user to resubmit the transaction or automatically restart the run unit, establish currency, and try again.

If the run unit is automatically restarted, the following steps should be followed:

1. **Rebind the run unit.** CA ADS automatically starts a new run unit when it encounters the first functional DML statement.
2. **Reestablish currency.** If appropriate currencies are not reestablished before retrying the operation that initially caused the deadlock, a status code of *nn06* (no currency established) will be returned.

Note: For more information about handling the minor code 29, see the *CA IDMS Navigational DML Programming Guide*.

Checking for Deadlock Conditions

Deadlock conditions can be checked for programmatically by using the `ALLOWING` clause when `autostatus` is enabled. The check for a deadlock condition can be made after each service request to the DBMS.

Note: For more information about record contention, see the *CA IDMS Database Design Guide*.

More information:

[Error Handling](#) (see page 277)

[COMMIT](#) (see page 384)

Suppression of Record Retrieval Locks

Specifications can be made during dialog compilation to indicate whether or not database record retrieval locks will be held for dialog run units. Retrieval dialogs that **do not update the database** and **do not pass currencies to update dialogs** can be selectively allowed to access database records without locking those records.

Selectively disabling retrieval locks for dialogs allows:

- Elimination of the overhead of maintaining retrieval locks. This decreases the amount of potential storage and CPU time used by dialogs at runtime.
- Reduction of the number of db-key deadlocks.

Disabling Record Retrieval Locks

To disable record retrieval locks, you must:

1. **Analyze the dialog in the context of the entire application** to ensure that control and currencies are passed appropriately. A dialog with disabled retrieval locks can pass control and currencies only to a dialog or user program that does retrieval based on these currencies.
2. **Verify the status of the system retrieval locks.** If the mandatory retrieval locks are on, disable the locks at system generation time by specifying `RETRIEVAL NOLOCK` in the system generation `SYSTEM` statement.

Note: For more information, see the *CA IDMS System Generation Guide*.

3. **Use the CA ADS dialog compiler or ADSOBCOM** to disable retrieval locking for appropriate dialogs.

Considerations

- To safeguard the database in the absence of retrieval locks, an update user program will be aborted when:
 - The program receives currencies from a retrieval dialog and attempts an update DML call.
 - The program finishes the current run unit and binds another. The abend occurs when control is passed back to CA ADS.
- The update dialog abends if:
 - A higher dialog in the application thread has the `RETRIEVAL NOLOCK` indicator set and system-wide `RETRIEVAL NOLOCKS` are specified.
- An update dialog or program is allowed to update the retrieval dialog's database records in the following cases:
 - The dialog with retrieval locks turned off readies the area in `UPDATE` mode.
 - The update dialog/program *does not* receive currencies when control passes to it.

Updates are allowed because the update dialog/program must ready the database in `UPDATE` mode and establish its own currency. The dialog/program will use record-locking mechanisms and will be assured of having the most up-to-date data.

The control command options that avoid passing currencies when control is passed are the `TRANSFER` command and the `NOSAVE` clause of the `DISPLAY`, `INVOKE`, and `LINK` commands.

More information:

[CA ADS Application Compiler \(ADSA\)](#) (see page 51)

[Control Commands](#) (see page 325)

[Application and Dialog Utilities](#) (see page 621)

Overview of ACCEPT

The ACCEPT command moves database keys, page information, and statistics from the database management system to a dialog's record buffers.

Formats of the ACCEPT Command

The ACCEPT command has three formats, as outlined in the table below.

Format	Description
ACCEPT DB-KEY FROM CURRENCY	Saves the database key and, optionally, the page information of the current record of run unit, record type, set, or area
ACCEPT DB-KEY RELATIVE TO CURRENCY	Saves the database key and, optionally, the page information of the next, prior, or owner record relative to the current record of a set
ACCEPT PAGE-INFO	Saves the page information of the record named.
ACCEPT STATISTICS	Returns runtime database statistics to the dialog

Note: The ACCEPT utility command should not be confused with the ACCEPT database command. The ACCEPT utility command is used to access information about the current DC/UCF task.

More information:

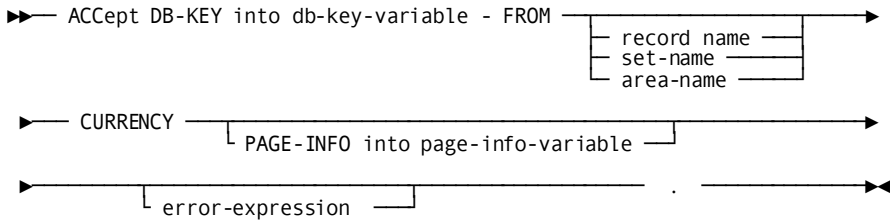
[Utility Commands](#) (see page 509)

ACCEPT DB-KEY FROM CURRENCY

Purpose

Saves the database key and, optionally, the page information of the current record of run unit, record type, set, or area.

Syntax



Parameters

ACCept DB-KEY into *db-key-variable*

Specifies the variable data field to which the database key of the object record is moved.

Db-key-variable is a PICS9(8) COMP SYNC.

Db-key-variable must be a binary full word field that is defined in a record associated with the dialog.

FROM

Specifies the record whose database key is moved to the field identified by *db-key-variable*.

record-name

Saves the database key of the record that is current of the specified record type.

set-name

Saves the database key of the record that is current of the specified set.

area-name

Saves the database key of the record that is current of the specified area.

CURRENCY

Specifies the current record of run unit, record type, set, or area.

If no record, set, or area is specified, CA ADS saves the database key of the record that is current of run unit.

PAGE-INFO into *page-info-variable*

Specifies the variable data field to which the page information of the named record is moved.

page-info-variable

A four-byte field that is defined either as a group field or as a fullword field (PIC S9(8) COMP). This parameter identifies the variable data field to contain the page information for the specified record. Upon successful completion of this statement, the first two bytes of the field contain the page group number and the last two bytes contain a value that may be used for interpreting dbkeys.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Definition

The ACCEPT DB-KEY FROM CURRENCY command is used to move the database key and, optionally, the page information of the current record of run unit, record type, set, or area to a specified location in a dialog's record buffers. A subsequent FIND/OBTAIN DB-KEY command can use the saved database key to access the record directly. FIND/OBTAIN DB-KEY is described later in this section.

Note: You must establish currency before using this statement. If no currency has been established, the DBMS returns 0000 to the ERROR-STATUS field and -1 to the *db-key* field.

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of an ACCEPT DB-KEY FROM CURRENCY command:

Status code	Meaning
0000	The request was executed successfully
1508	The object record is not in the dialog's subschema

Example

The statements in the following example establish a PRODUCT record as current of run unit and save the record's database key in the field SAVE-DB-KEY:

```
MOVE 7690157 TO PROD-NUMBER.
FIND CALC PRODUCT.
ACCEPT DB-KEY INTO SAVE-DB-KEY FROM CURRENCY.
```

More information:

[Error Handling](#) (see page 277)

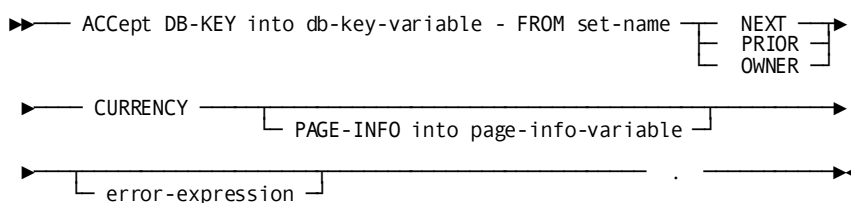
[ACCEPT PAGE-INFO](#) (see page 378)

ACCEPT DB-KEY RELATIVE TO CURRENCY

Purpose

Saves the database key and, optionally, the page information of the next, prior, or owner record relative to the current record of a set.

Syntax



Parameters

ACCEpt DB-KEY into *db-key-variable*

Specifies the variable data field to which the database key of the object record is moved.

Db-key-variable is a PICS9(8) COMP SYNC.

Db-key-variable must be a binary full word field that is defined in a record associated with the dialog.

FROM *set-name*

Specifies the record whose database key is moved to the field identified by *db-key-variable*.

Set-name must be known to the dialog's subschema.

NEXT

Saves the database key of the next record relative to the current record of the specified set.

A request for NEXT CURRENCY cannot be specified unless the object set has prior pointers, which ensure that the next pointer in the prefix of the current record does not point to a logically deleted record.

PRIOR

Saves the database key of the prior record relative to the current record of the specified set.

A request for PRIOR CURRENCY cannot be specified unless the object set has prior pointers.

Note: No indication of an end-of-set condition is possible for an ACCEPT NEXT or ACCEPT PRIOR command. A retrieval command must be issued to determine whether the next or prior record in the specified set is the owner record.

OWNER

Saves the database key of the owner of the current record of the specified set.

A request for OWNER CURRENCY cannot be specified unless the object set has owner pointers. If the current record is the owner of the specified set, a request for OWNER CURRENCY returns the database key of the current record, even if the set does not have owner pointers.

CURRENCY

Specifies the current record of run unit, record type, set, or area.

PAGE-INFO into *page-info-variable*

Specifies the variable data field to which the page information of the named record is moved.

page-info-variable

A four-byte field that is defined either as a group field or as a fullword field (PIC S9(8) COMP). This parameter identifies the variable data field to contain the page information for the specified record. Upon successful completion of this statement, the first two bytes of the field contain the page group number and the last two bytes contain a value that may be used for interpreting dbkeys.

error-expression

Specifies the status codes that are returned to the dialog.

Usage*Definition*

The ACCEPT DB-KEY RELATIVE TO CURRENCY command is used to move the database key and, optionally, the page information of the next, prior, or owner record relative to the current record of set to a specified location in a dialog's record buffers.

This command allows a process to save the database key of a record without accessing the record itself. A subsequent FIND/OBTAIN DB-KEY command can use the saved database key to access the record directly. FIND/OBTAIN DB-KEY is described later in this section.

Note: You must establish currency before using this statement. If no set currency has been established, the DBMS returns 0000 to the ERROR-STATUS field and -1 to the *db-key* field. NEXT, PRIOR, and OWNER CURRENCY cannot be requested for sets defined for native VSAM records.

If autostatus is not in use, a dialog's error-status field indicates the outcome of an ACCEPT DB-KEY RELATIVE TO CURRENCY command:

Status code	Meaning
0000	The request was executed successfully
1506	Currency was not established for the object set
1508	The object record is not in the dialog's subschema

Example

The statements in the following example establish a current ITEM record and save the database key of the owner record of the PRODUCT-ITEM set in the field SAVE-KEY:

```
MOVE 1230407 TO ORD-NUMBER.
FIND CALC ORDOR.
FIND NEXT WITHIN ORDER-ITEM.
ACCEPT DB-KEY INTO SAVE-KEY FROM PRODUCT-ITEM OWNER CURRENCY.
```

More information:

[Error Handling](#) (see page 277)

[ACCEPT PAGE-INFO](#) (see page 378)

ACCEPT PAGE-INFO

Purpose

The ACCEPT PAGE-INFO statement moves the page information for a given record to a specified location in program variable storage. Page information that is saved in this manner is available for subsequent direct access by using a FIND/OBTAIN DB-KEY statement.

Status Codes

If autostatus is not in use, a dialog's error-status field indicates the outcome of an ACCEPT-PAGE-INFO command:

Status code	Meaning
0000	The request has been serviced successfully.
1508	The named record is not in the subschema. The program probably invoked the wrong subschema.

Example

The following example retrieves the page information for the DEPARTMENT record.

```

01 W-PG-INFO.
   02 W-GRP-NUM      PIC S9(4) COMP.
   02 W-DBK-FORMAT  PIC 9(4) COMP.

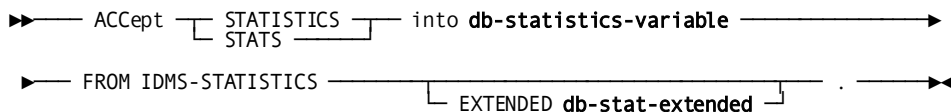
ACCEPT PAGE-INFO INTO W-PG-INFO FOR DEPARTMENT.
    
```

ACCEPT STATISTICS

Purpose

Returns runtime database statistics to the dialog.

Syntax



Parameters

ACCEpt STATISTICS

Introduces the variable data field to which the database key of the object record is moved.

STATS can be used in place of STATISTICS.

into *db-statistics-variable*

The name of the location in the dialog's record buffers where the runtime statistics contained in the CA IDMS statistics block are to be moved.

A fullword aligned, 100-byte system supplied statistics block shown below:

```

01 DB-STATISTICS
  03 DATE-TODAY          PIC X(8) .
  03 TIME-TODAY          PIC X(8) .
  03 PAGES-READ          PIC S9(8)  COMP.
  03 PAGES-WRITTEN       PIC S9(8)  COMP.
  03 PAGES-REQUESTED     PIC S9(8)  COMP.
  03 CALC-TARGET         PIC S9(8)  COMP.
  03 CALC-OVERFLOW       PIC S9(8)  COMP.
  03 VIA-TARGET          PIC S9(8)  COMP.
  03 VIA-OVERFLOW        PIC S9(8)  COMP.
  03 LINES-REQUESTED     PIC S9(8)  COMP.
  03 RECS-CURRENT        PIC S9(8)  COMP.
  03 CALLS-TO-IDMS       PIC S9(8)  COMP.
  03 FRAGMENTS-STORED    PIC S9(8)  COMP.
  03 RECS-RELOCATED      PIC S9(8)  COMP.
  03 LOCKS-REQUESTED     PIC S9(8)  COMP.
  03 SEL-LOCKS-HELD      PIC S9(8)  COMP.
  03 UPD-LOCKS-HELD      PIC S9(8)  COMP.
  03 RUN-UNIT-ID         PIC S9(8)  COMP.
  03 TASK-ID             PIC S9(8)  COMP.
  03 LOCAL-ID            PIC X(8) .
  03 FILLER              PIC X(8) .
  
```

Note: Record DB-STATISTICS is defined in the dictionary when CA IDMS is installed and can be included as a dialog work record. For more information about the CA IDMS statistics block, see the *CA IDMS Database Administration Guide*.

The LOCAL-ID field consists of the 4-byte identifier of the interface in which the run unit originated (in CA ADS, it is always DBDC) and a unique identifier (a fullword binary value) assigned to the run unit by that interface. To display the originating interface identifier and the run-unit identifier, the LOCAL-ID field can be moved to a work field that is defined as follows:

```

01 WORK-LOCAL-ID
  02 WORK-LOCAL-ORIGIN   PIC X(4) .
  02 WORK-LOCAL-NUMBER   PIC S9(8)  COMP.
  
```

Alternatively, the DB-STATISTICS record can be modified to define two subordinate fields for the LOCAL-ID field.

into *db-stat-extended*

The name of the location in the dialog's record buffers where the extended runtime statistics contained in the CA IDMS statistics block are to be moved.

A fullword aligned, 100-byte system supplied statistics block shown below:

```

01 DB-STAT-EXTENDED
   03 SR8-SPLITS           PIC S9(8)  COMP.
   03 SR8-SPAWNS          PIC S9(8)  COMP.
   03 SR8-STORES          PIC S9(8)  COMP.
   03 SR8-ERASES          PIC S9(8)  COMP.
   03 SR7-STORES          PIC S9(8)  COMP.
   03 SR7-ERASES          PIC S9(8)  COMP.
   03 BINARY-SEARCHES-TOTAL PIC S9(8)  COMP.
   03 LEVELS-SEARCHED-TOTAL PIC S9(8)  COMP.
   03 ORPHANS-ADOPTED     PIC S9(8)  COMP.
   03 LEVELS-SEARCHED-BEST PIC S9(4)  COMP.
   03 LEVELS-SEARCHED-WORST PIC S9(4)  COMP.
   03 FILLER              PIC X(60) .
    
```

Note: Record DB-STAT-EXTENDED is defined in the dictionary when CA IDMS is installed and can be included as a dialog work record. For more information about the CA IDMS statistics block, see the *CA IDMS Database Administration Guide*.

Usage

Definition

The ACCEPT STATISTICS command is used to move runtime statistics in the CA IDMS statistics block to a dialog's record buffers. An ACCEPT STATISTICS command does not reset fields in the CA IDMS statistics block. The fields are initialized at the beginning of a run unit. The only acceptable status code returned for an ACCEPT STATISTICS command is 0000.

Example

The statements in the following example:

- Establish currency for the sets in which a new ITEM record will participate as a member
- Store the ITEM record
- Move statistics regarding the stored ITEM record to the SAVE-STATS field in the dialog's record buffers

Sample Statements

```

MOVE IN-PROD-NUMBER TO PROD-NUMBER.
FIND CALC PRODUCT.
MOVE IN-ORD-NUMBER TO ORD-NUMBER.
FIND CALC ORDOR.
STORE ITEM.
ACCEPT STATS INTO SAVE-STATS FROM IDMS-STATISTICS.
    
```

BIND PROCEDURE

Purpose

Establishes communication between a dialog and a DBA-written procedure.

Syntax

```

▶▶ BIND PROCedure for procedure-name TO _____
▶ procedure-control-location _____ . _____
                                     └─ error-expression ─┘
    
```

Parameters

BIND PROCedure for *procedure-name*

Provides the name of the database procedure.

Procedure-name is either the name of an 8-character variable field that contains the procedure name or the procedure name itself enclosed in single quotation marks.

TO *procedure-control-location*

Specifies the location to which the named procedure is bound.

Procedure-control-location specifies a 256-byte, fixed-length area.

When the BIND PROCEDURE command is executed, information specified in the CA IDMS application program information block is copied into *procedure-control-location*. At runtime, this information is copied from *procedure-control-location* back into the CA IDMS application program information block each time the DBMS invokes the procedure. The information passed at runtime is not the information in storage at the time of the procedure call.

error-expression

Specifies status codes that are returned to the dialog.

Usage

Definition

This statement should be used when the application must pass more information to the procedure than that provided by the DBMS. Such instances are unusual.

In most cases, procedures that gain control before or after various database functions are not apparent. After the BIND PROCEDURE command is executed, the DBMS automatically invokes the named procedure for the operations specified in the schema definition.

Note: For more information about database procedures, see the *CA IDMS Database Administration Guide*.

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of a BIND PROCEDURE command:

Status code	Meaning
0000	The request was executed successfully
1408	The named record or procedure was not in the specified subschema.
1418	The procedure was improperly bound to location 0
1472	The available memory to load a module from the load (core-image) library or DDLDCLOUD was not sufficient
1474	An attempt to load a module from the load (core-image) library or DDLDCLOUD failed

Example

In the following example, the BIND PROCEDURE command is used to bind the procedure PROGCHK to the 256-byte area PROC-CTL.

```
BIND PROCEDURE FOR 'PROGCHK' TO PROC-CTL.
```

More information:

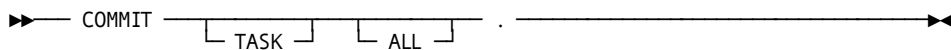
[Error Handling](#) (see page 277)

COMMIT

Purpose

Ends the current recovery unit and makes permanent any changes made to the database data during the current recovery unit.

Syntax



Parameters

TASK

COMMIT TASK writes a checkpoint to the CA IDMS/DB journal file and updates the subschema control block for all database, queue, and scratch records associated with run units that have been implicitly established for the issuing dialog. All record locks except those held on current records are released.

If TASK is not specified, only database records are the objects of the COMMIT command.

ALL

Releases all record locks, including those held on current records, and sets all currencies to null.

Note: The COMMIT command does not release long-term locks held on database records.

Usage

Definition

The COMMIT command is used to write a checkpoint to the CA IDMS/DB journal file and to release record locks held on database, queue, and scratch records. The checkpoints mark the beginning or end of specific database, queue, and scratch area activities within the issuing dialog. The release of record locks lessens the possibility of abnormal termination resulting from too many locks.

The CA ADS runtime system automatically writes a checkpoint to the CA IDMS/DB journal file at the beginning and end of a run unit. Additional checkpoints can be written to the journal file by using the COMMIT command.

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of a COMMIT command:

Status code	Meaning
0000	The request was executed successfully
5031	The request is invalid, possibly due to a logic error in the process
5096	Too many run units exist for the internal run-unit table
5097	An invalid status was received from DBIO. Check the DC/UCF system log for details

More information:

[CA ADS Runtime System](#) (see page 119)

[Error Handling](#) (see page 277)

CONNECT

Purpose

Establishes a record occurrence as a member in a set occurrence.

Participation of records in sets is governed by the membership options defined for each set in the subschema, as shown below.

Membership option	Description
Automatic	Membership is established automatically when a record is stored.
Manual	Membership is not established automatically. A record is established as a member of the set by using the CONNECT command.
Mandatory	Records remain members of the set until they are erased.
Optional	Records remain members of the set until they are erased or disconnected. For information on erasing or disconnecting a record, see 'ERASE' and 'DISCONNECT' later in this section.

Syntax

```

▶▶ CONNECT record-name TO set-name [error-expression] . ▶▶
    
```

Parameters

record-name

Specifies the current occurrence of the named record to be connected with the current occurrence of the set specified by *set-name*.

Record-name must be known to the dialog's subschema.

TO *set-name*

Specifies the set to which the current occurrence of the named record is connected.

Set-name must be known to the dialog's subschema and must be defined as optional automatic, optional manual, or mandatory manual.

The record is connected to the current occurrence of the named set in the order specified for the set in the schema.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of a CONNECT command:

Status code	Meaning
0000	The request was executed successfully
0705	The CONNECT command violates a duplicates-not-allowed option for a CALC, sorted, or index set
0706	Currency was not established for the object record or set
0708	The object record is not in the dialog's subschema
0709	The object record's area was not readied in an update usage mode
0710	The dialog's subschema specifies an access restriction that prohibits connecting the object record to the named set
0714	The CONNECT command cannot be executed because the object set was defined as mandatory automatic
0716	The CONNECT command cannot be executed because the object record is already a member of the named set
0721	An area other than the area of the object record was readied with an incorrect usage mode
0729	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released

Further Considerations

- The object set in a CONNECT command must be defined as optional automatic, optional manual, or mandatory manual.
- The CONNECT command cannot be used with native VSAM data sets because all such sets must be defined as mandatory automatic.
- Before a CONNECT command can be executed, the following conditions must be satisfied:
 - All areas affected either directly or indirectly by the CONNECT command must be readied in an update usage mode.
 - The object record must be established as current of its record type.
 - The applicable set occurrence must be established by the current record of set. If set order is NEXT or PRIOR, the current record of set also determines the position at which the object record is connected within the set.
- After successful execution of the CONNECT command, the object record is current of:
 - The run unit
 - Its record type
 - Its area
 - All sets in which the record currently participates

Example

The statements in the following example establish currency for the ITEM and PRODUCT record types, and connect the current ITEM record to the set occurrence established by the current PRODUCT record:

```
MOVE 'BB' TO ORD-NUMBER.  
FIND CALC ORDOR.  
OBTAIN FIRST WITHIN ORDER-ITEM.  
MOVE ITEM-PROD-NUMBER TO PROD-NUMBER.  
FIND CALC PRODUCT.  
CONNECT ITEM TO PRODUCT-ITEM.
```

More information:

[Error Handling](#) (see page 277)

[READY](#) (see page 423)

DISCONNECT

Purpose

Disconnects a record occurrence from a set occurrence in which it participates as a member.

Membership in the object set must be defined as OPTIONAL in the dialog's schema.

Syntax

```
DISCONNECT record-name FROM set-name [ error-expression ] .
```

Parameters

record-name

Specifies the current occurrence of the named record to be disconnected.

Record-name must be known to the dialog's subschema.

FROM *set-name*

Specifies the set from which the current occurrence of the named record is to be disconnected.

Set-name must be known to the dialog's subschema and must be defined as optional.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of a DISCONNECT command:

Status code	Meaning
0000	The request was executed successfully
1106	Currency was not established for the object record
1108	The specified record is not in the dialog's subschema
1109	The object record's area was not readied in an update usage mode
1110	The dialog's subschema specifies an access restriction that prohibits disconnecting the object record from the named set

Status code	Meaning
1115	The DISCONNECT command cannot be executed because the object set is defined as mandatory
1121	An area other than the area of the object record was readied with an incorrect usage mode
1122	The object record is not currently a member of the specified set
1129	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released

Further Considerations

- The DISCONNECT command cannot be used with native VSAM data sets because all such sets must be defined as mandatory automatic.
- Before a DISCONNECT command can be executed, the following conditions must be satisfied:
 - All areas affected either directly or indirectly by the DISCONNECT command must be readied in an update usage mode.
 - The object record must be established as current of its record type.
- After successful execution of a DISCONNECT command, the object record can no longer be accessed through the set for which membership was canceled.
- A disconnected record can be accessed through any other sets in which it participates as a member or through its CALC key if it has a location mode of CALC.
- A disconnected record is always accessible by means of an area search or through its database key.
- A DISCONNECT command nullifies currency in the object set. However, the next of set and prior of set are maintained, enabling access to continue within the set.
- A disconnected record becomes current of:
 - The run unit
 - Its record type
 - Its area

Example

The statements in the following example establish an ITEM record as current of record type and disconnect the record from the PRODUCT-ITEM set:

```
MOVE 'P8' TO PROD-NUMBER.
FIND CALC PRODUCT.
FIND FIRST ITEM WITHIN PRODUCT-ITEM.
DISCONNECT ITEM FROM PRODUCT-ITEM.
```

More information:

[Error Handling](#) (see page 277)

[CONNECT](#) (see page 386)

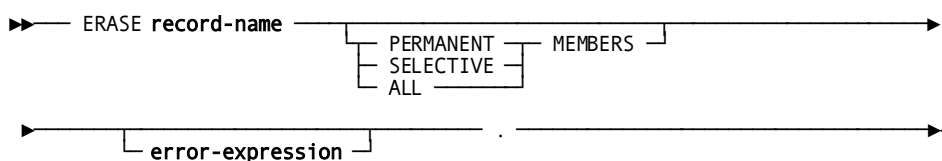
[READY](#) (see page 423)

ERASE

Purpose

Deletes a record from the database.

Syntax



Parameters

record-name

Erases the current occurrence of the named record from the database.

Record-name must be known to the dialog's subschema and must be current of run unit.

Note: Native VSAM users—ERASE *record-name* is the only form of the ERASE statement valid for records in a native VSAM KSDS or RRDS; no form of the ERASE statement is allowed for a native VSAM ESDS.

PERMANENT

Specifies the named record and all mandatory member record occurrences owned by the record to be erased. Optional member records are disconnected.

An erased mandatory member record that is itself the owner of any set occurrences is also treated as the direct object of an ERASE PERMANENT command (that is, all mandatory members in the sets owned by the record are also erased).

SELECTIVE

Specifies the named record and all mandatory member record occurrences owned by the record to be erased. Optional member records are erased if they do not currently participate as members in other set occurrences.

An erased member record that is itself the owner of any set occurrences is also treated as the direct object of an ERASE SELECTIVE command.

ALL

Specifies the named record, all mandatory and optional member record occurrences owned by the record to be erased.

An erased member record that is itself the owner of any set occurrences is also treated as the direct object of an ERASE ALL command.

MEMBERS

Must be specified if the record identified by *record-name* is the owner of any nonempty set occurrences.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Definition

Erasure is a two-step process that first cancels a record's membership in any set occurrences and then releases for reuse the space occupied by the record.

The ERASE command performs the following functions:

- Erases the object record from the database
- Erases all records that are mandatory members of set occurrences owned by the object record
- Disconnects or erases all records that are optional members of set occurrences owned by the object record

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of an ERASE command:

Status code	Meaning
0000	The request was executed successfully
0206	Currency was not established for the object record
0209	The object record's area was not readied in an update usage mode
0210	The dialog's subschema specifies an access restriction that prohibits use of the ERASE command.
0213	Run-unit currency was not established or was nullified by a previous ERASE command

Status code	Meaning
0220	The current record of run unit is not the same type as the specified record
0221	An area other than the area of the object record was readied with an incorrect usage mode
0225	Currency was not established for the object record or set
0229	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released
0230	An attempt was made to erase the owner of a nonempty set
0233	Erasure of the object record is not allowed by the dialog's subschema, or not all sets in which the object record participates are included in the subschema

Further Considerations

- Before an ERASE command can be executed, the following conditions must be satisfied:
 - All areas either directly or indirectly affected by the ERASE command must be readied in an update usage mode.
 - All sets in which the object record participates as owner either directly or indirectly (for example, a set whose owner is a member of a set owned by the object record) and all member record types in those sets must be included in the dialog's subschema.
 - The object record must be established as current of run unit.
- An ERASE command nullifies the CURRENT pointer for all record types involved in the erase and for all sets in which erased records participate. Run-unit and area currencies remain unchanged.
- The next of set and prior of set are maintained when walking the set occurrence of an erased record, whether or not prior pointers have been defined for the sets.
- Erased records are not available for further processing. An attempt to retrieve an erased record results in an error condition.
- Next, prior, and owner pointers are preserved for the last occurrence of each record type erased. This enables access to the next or prior record within the area, or the next, prior, or owner records within the sets in which the erased record participated.

More information:

[Error Handling](#) (see page 277)

[READY](#) (see page 423)

Overview of FIND/OBTAIN

The FIND command is used to locate a record occurrence in the database. The OBTAIN command is used to locate a record and move the data associated with the record to a dialog's record buffers. Because the FIND and OBTAIN command statements have identical formats, they are discussed together.

There are six formats of the FIND/OBTAIN statement, as outlined below.

Formats of the FIND/OBTAIN Statement

Format	Description
FIND/OBTAIN CALC	Accesses a record occurrence by using its CALC key value
FIND/OBTAIN CURRENT	Accesses a record occurrence by using established currencies
FIND/OBTAIN DB-KEY	Accesses a record occurrence by using its database key
FIND/OBTAIN OWNER	Accesses the owner record of a set occurrence
FIND/OBTAIN WITHIN SET/AREA	Accesses a record occurrence based on its logical location within a set or on its physical location within an area
FIND/OBTAIN WITHIN SET USING SORT KEY	Accesses a record occurrence in a sorted set by using its sort key value

Considerations

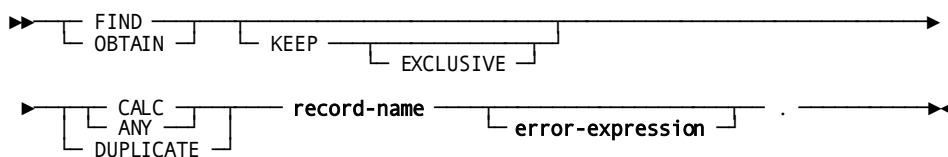
- Locks can be placed on located record occurrences by using the KEEP clause of a FIND/OBTAIN statement. The KEEP clause sets a shared or exclusive lock.
 - **KEEP** places a shared lock on the located record occurrence. Other concurrently executing run units can access but not update the locked record.
 - **KEEP EXCLUSIVE** places an exclusive lock on the located record occurrence. Other concurrently executing run units can neither access nor update the locked record.

FIND/OBTAIN CALC

Purpose

Accesses a record based on the value of the record's CALC key.

Syntax



Parameters

KEEP

Places a shared lock on the object record.

EXCLUSIVE

Places an exclusive lock on the object record.

CALC

Accesses the first or only occurrence of the named record type whose CALC key value matches the value of the CALC data item in the dialog's record buffer.

ANY can be used in place of CALC.

DUPLICATE

Accesses the next occurrence of the named record type with the same CALC key value as the current record of run unit. Use of the DUPLICATE option requires previous access to an occurrence of the same record type by means of the CALC option.

record-name

Specifies the name of the record being accessed.

Record-name must be known to the dialog's subschema.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of a FIND/OBTAIN CALC command:

Status code	Meaning
0000	The request was executed successfully
0306	Currency was not established for the object record (applies to the DUPLICATE option only)

Status code	Meaning
0308	The object record is not in the dialog's subschema
0310	The dialog's subschema specifies an access restriction that prohibits retrieval of the object record
0326	The specified record cannot be found
0329	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released
0331	The object record was not defined with a location mode of CALC
0332	The value of the CALC data item in the dialog's record buffer does not equal the value of the CALC data item in the current record of run unit (applies to DUPLICATE option only)

Further Considerations

- The object record must be stored in the database with a location mode of CALC.
- Before a FIND/OBTAIN CALC command is issued, the CALC key value of the object record must be placed in the applicable field of the dialog's record buffer .
- After successful execution of a FIND/OBTAIN CALC command, the accessed record is current of:
 - The run unit
 - Its record type
 - Its area
 - All sets in which it currently participates as member or owner

Example

The statements in the following example initialize the CALC key field in a dialog's ORDOR record buffer and retrieve the specified occurrence of the ORDOR record:

```
MOVE IN-ORDER-NUMBER TO ORD-NUMBER.
OBTAIN CALC ORDOR.
```

More information:

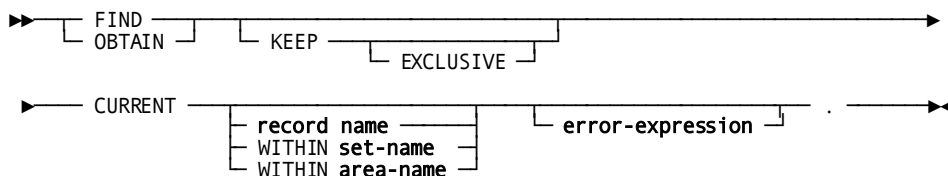
[Error Handling](#) (see page 277)

FIND/OBTAIN CURRENT

Purpose

Accesses a record that is current of run unit, current of the record's record type or area, or current of any set in which the record participates as member or owner.

Syntax



Parameters

KEEP

Places a shared lock on the object record.

EXCLUSIVE

Places an exclusive lock on the object record.

CURRENT

Accesses the record occurrence that is current of run unit.

record-name

Specifies the current occurrence of the named record to be accessed.

WITHIN *set-name*

Specifies the current occurrence of the named set to be accessed.

WITHIN *area-name*

Specifies the current occurrence of the named area to be accessed.

The named record, set, or area must be known to the dialog's subschema.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

FIND/OBTAIN CURRENT is an efficient means of establishing a record as current of run unit before executing a command that uses run-unit currency (for example, ERASE, GET, or MODIFY).

After successful execution of a FIND/OBTAIN CURRENT command, the accessed record is current of:

- Run unit
- Record type
- Area
- All sets in which the record participates as member or owner

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of a FIND/OBTAIN CURRENT command:

Status code	Meaning
0000	The request was executed successfully
0306	Currency was not established for the named record, set, or area
0308	The object record is not in the dialog's subschema
0310	The dialog's subschema specifies an access restriction that prohibits retrieval of the object record
0313	Run-unit currency was not established or was nullified by a previous ERASE command
0323	The named area is not in the dialog's subschema
0329	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released

Example

The statements in the following example establish an ITEM record as current of run-unit before issuing a command that requires run-unit currency:

```
MOVE 'BB' TO ORD-NUM.
OBTAIN CALC ORDOR.
OBTAIN FIRST ITEM WITHIN ORDER-ITEM.
OBTAIN OWNER WITHIN PRODUCT-ITEM.
OBTAIN CURRENT ITEM.
MODIFY ITEM.
```

The object ITEM record becomes current of run unit following the third statement. The fourth statement establishes the owner PRODUCT record as current of run unit. The OBTAIN CURRENT statement reestablishes the ITEM record as current of run unit.

More information:

[Error Handling](#) (see page 277)

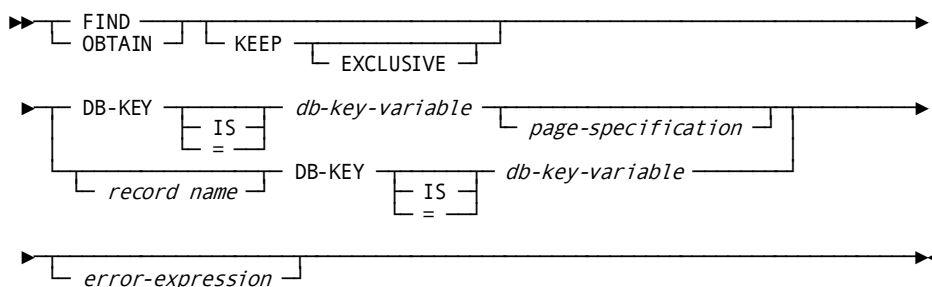
FIND/OBTAIN DB-KEY

Purpose

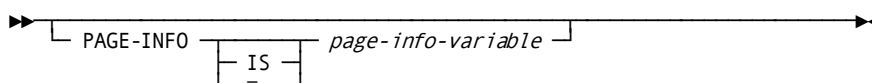
Accesses a record occurrence directly by using a database key that is stored in a field in a dialog's record buffers.

Any record in a dialog's subschema can be accessed in this manner, regardless of its location mode.

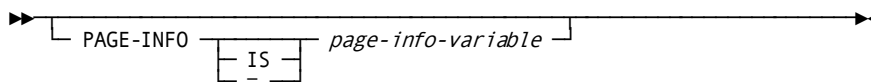
Syntax



Expansion of page-specification



Expansion of *page-specification*



Parameters

KEEP

Places a shared lock on the object record.

EXCLUSIVE

Places an exclusive lock on the object record.

record-name

Specifies the name of the record to be accessed using the database key value contained in *db-key-variable*.

If specified, *record-name* must be known to the dialog's subschema.

DB-KEY IS *db-key-variable*

Specifies the binary fullword in the dialog's record buffers that contains a previously saved database key. If *record-name* is specified, *db-key-variable* must contain the database key of an occurrence of the named record type. If *record-name* is not specified, *db-key-variable* can contain the database key of an occurrence of any record type in the dialog's subschema.

Db-key-variable is a PICS9(8) COMP SYNC.

IS or = are optional keywords and have no effect on processing.

PAGE-INFO

Specifies page information that is used to determine the area with which the dbkey is associated. If not specified, the page information associated with the record that is current of rununit is used.

Note: Page information is only used if the subschema includes areas that have mixed page groups; otherwise, it is ignored.

page-info-variable

A four-byte field that may be defined either as a group field or as a fullword field (PICS9(8) COMP). Identifies the location in variable storage that contains the page information previously saved by the program.

Page information is returned in the PAGE-INFO field in the subschema control area if the subschema includes areas in mixed page groups. Page information may also be returned using an ACCEPT PAGE-INFO statement.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of a FIND/OBTAIN DB-KEY command:

Status code	Meaning
0000	The request was executed successfully
0302	The database key value is inconsistent with the area in which the named record is stored. Either the database key was not initialized properly or the record name is incorrect
0308	The object record is not in the dialog's subschema
0310	The dialog's subschema specifies an access restriction that prohibits retrieval of the named record
0326	The specified record cannot be found
0329	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released
0371	The specified database key does not correspond to a database page

Further Considerations

- FIND/OBTAIN DB-KEY cannot be used to access data records in a native VSAM KSDS.
- After successful execution of a FIND/OBTAIN DB-KEY command, the accessed record is current of the run unit, its record type, its area, and all sets in which it currently participates as member or owner.

Example

The following example illustrates the use of the FIND DB-KEY command to locate an occurrence of the ITEM record whose database key matches the value in a field called SAVED-KEY:

```
FIND ITEM DB-KEY IS SAVED-KEY.
```

More information:

[Error Handling](#) (see page 277)

Further Considerations

- Currency must be established for the object set to execute a FIND/OBTAIN OWNER command.
- FIND/OBTAIN OWNER can be used to retrieve the owner record of any set in a dialog's subschema, whether or not the set has owner pointers.
- The FIND/OBTAIN OWNER command cannot be used with native VSAM data sets because owner records are not defined for such sets.
- When an optional or manual member of a set is accessed, it is not established as current of set if it is not currently connected to the object set. A subsequent attempt to access the owner record locates instead the owner of the current record of set. The IF statement can be used to determine if the accessed record is actually a member of the object set before executing the FIND/OBTAIN OWNER command.
- After successful execution of a FIND/OBTAIN OWNER command, the accessed record is current of:
 - The run unit
 - Its record type
 - Its area
 - All sets in which it currently participates as member or owner
- If the current record of set is the owner record when the command is executed, currency in the object set is not changed.

Example

The statements in the following example illustrate the use of the FIND/OBTAIN OWNER command:

```
MOVE 'CC' TO ORD-NUM.
OBTAIN CALC ORDOR.
OBTAIN LAST ITEM WITHIN ORDER-ITEM.
OBTAIN OWNER WITHIN PRODUCT-ITEM.
```

More information:

[Error Handling](#) (see page 277)

[Conditional Commands](#) (see page 317)

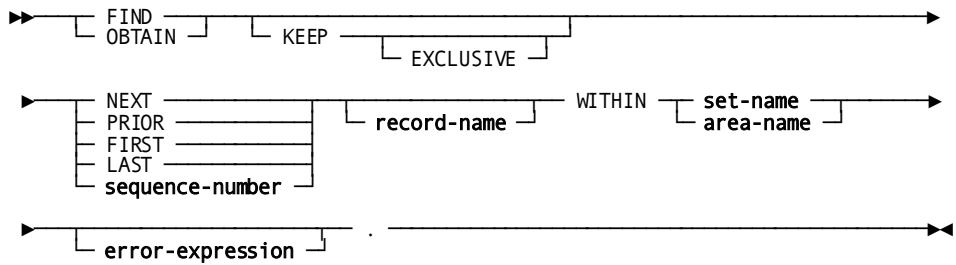
FIND/OBTAIN WITHIN SET/AREA

Purpose

Accesses records logically, based on set relationships, or physically, based on database location.

Records can be accessed serially in a specified set or area, or accessed by selected specific occurrences of a given record type within the set or area.

Syntax



Parameters

KEEP

Places a shared lock on the object record.

EXCLUSIVE

Places an exclusive lock on the object record.

NEXT

Specifies the next record in the specified set or area relative to the current record.

PRIOR

Specifies the prior record in the specified set or area relative to the current record.

FIRST

Specifies the first record in the set or area.

LAST

Specifies the last record in the set or area.

sequence-number

Either the name of a variable data field that contains the sequence number of a record in a set or area or the sequence number itself expressed as a positive or negative integer. The actual sequence number, expressed as a positive or negative integer.

If *sequence-number* is negative, the specified set must have prior pointers.

record-name

Specifies only occurrences of the named record type.

Record-name must be defined as a member of the object set or be contained in the object area.

Record-name must be specified if *set-name* or *area-name* is specified, unless the record or one of the record's elements is named explicitly somewhere in the dialog's process code, or if the record is associated with the dialog as a map record.

WITHIN

Introduces the named set or area to be searched.

set-name

Specifies the set to be searched.

area-name

Specifies the area to be searched.

The named set or area must be known to the dialog's subschema.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of a FIND/OBTAIN WITHIN SET/AREA command:

Status code	Meaning
0000	The request was executed successfully
0304	A sequence number of zero or a variable data field containing a value of zero was specified for the object record
0306	Currency was not established for the named record, set, or area
0307	The end of the set or area was reached, or the set is empty
0308	The object record is not in the dialog's subschema
0310	The dialog's subschema specifies an access restriction that prohibits retrieval of the named record
0318	The record retrieved was not bound. The record or one of the record's elements must be named explicitly somewhere in the dialog's process code, or the record must be associated with the dialog as a map record

Status code	Meaning
0323	The named area is not in the dialog's subschema, or the named record is not in the named area
0326	The object record cannot be found
0329	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released

Further Considerations

- After successful execution of a FIND/OBTAIN WITHIN SET/AREA command, the accessed record is current of:
 - The run unit
 - Its record type
 - Its area
 - All sets in which it currently participates as member or owner
- When accessing records within a set, the following considerations apply:
 - The current record of the specified set determines the set occurrence to be accessed. Set currency must be established before a FIND/OBTAIN WITHIN SET command is executed.
 - The next or prior record within a set is the subsequent or previous record, relative to the current record of the named set in the logical order of the set. The prior record in a set can be retrieved only if the set has prior pointers.
 - The first or last record within a set is the first or last member occurrence, respectively, in terms of the logical order of the set. The record accessed is the same record accessed when the current of set is the owner record and the next or prior record is requested. The last record in a set can be retrieved only if the set has prior pointers.
 - The *n*th occurrence of a record within a set can be accessed by specifying a sequence number that identifies the position of the record in the set. The search begins with the owner of the current of set for the specified set and continues until the *n*th record is located or an end-of-set condition is encountered. If the specified sequence number is negative, the search proceeds in the prior direction in the set. A negative number can be specified only if the set has prior pointers.

- When an end-of-set condition occurs, the owner record occurrence of the set becomes current of:
 - The run unit
 - Its record type
 - Its area

The owner record also becomes current of the set named in the FIND/OBTAIN command. Currency of other sets in which the record participates as member or owner is not changed.

- If OBTAIN is specified and an end-of-set condition occurs, the contents of the owner record are not moved to the dialog's record buffer (that is, OBTAIN is treated as FIND).
- When accessing records within an area, the following considerations apply:
 - The first record occurrence within an area is the one with the lowest database key. The last record occurrence is the one with the highest database key.
 - The next record within an area is the one with the next higher database key relative to the current record of the object area. The prior record is the one with the next lower database key relative to the current of area.
 - The first or last record or the *n*th occurrence of a record in an area must be accessed to establish correct starting position before next or prior records are requested.
- When using a native VSAM set, the following considerations apply:
 - When an end-of-set or end-of-area condition (0307) occurs for a native VSAM set, all currencies remain unchanged.
 - The FIRST, LAST, and *sequence-number* WITHIN *area-name* options cannot be used to access spanned data records in a native VSAM data set.

Example

The statements in the following example illustrate the use of the FIND/OBTAIN WITHIN SET command to retrieve records in an occurrence of the ORDER-ITEM set:

```
MOVE 'BB' TO ORD-NUM.
FIND CALC ORDOR.
OBTAIN FIRST ITEM WITHIN ORDER-ITEM.
OBTAIN NEXT WITHIN ORDER-ITEM.
OBTAIN 5 WITHIN ORDER-ITEM.
OBTAIN NEXT WITHIN ORDER-ITEM.
```

If the fifth ITEM record is the last record in the ORDER-ITEM set, the fourth OBTAIN statement finds the owner ORDOR record.

More information:

[Error Handling](#) (see page 277)

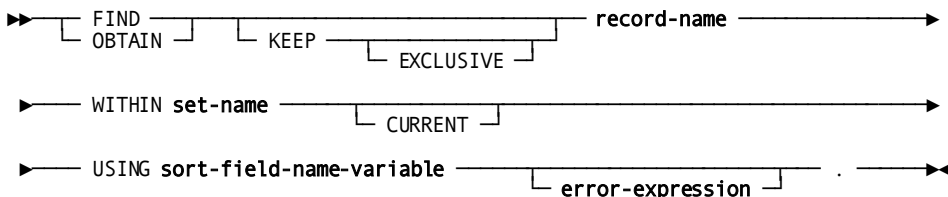
FIND/OBTAIN WITHIN SET USING SORT KEY

Purpose

Accesses a member record in a sorted set.

Sets are sorted in ascending or descending order based on the value of a sort-control element in each member record. The search begins with the current of set or the owner of the current of set and always proceeds through the set in the next direction.

Syntax



Parameters

KEEP

Places a shared lock on the object record.

EXCLUSIVE

Places an exclusive lock on the object record.

record-name

Specifies the record to be accessed.

Record-name must be known to the dialog's subschema and must participate in the set specified by *set-name*.

WITHIN *set-name*

Specifies the set in which the object record participates.

Set-name must be known to the dialog's subschema.

CURRENT

Specifies that the search begins with the current record of the named set.

If the set is sorted in ascending order and the sort key value of the record that is current of set is higher than the sort key value specified by *sort-field-name-variable*, an error condition results. If the set is sorted in descending order and the sort key value of the record that is current of set is lower than the sort key value specified by *sort-field-name-variable*, an error condition results.

If CURRENT is not specified, the search begins with the owner of the current record of the named set.

USING *sort-field-name-variable*

Specifies the sort-control element to be used in searching the sorted set.

Sort-field-name-variable is either the name of the sort-control element in the record specified by *record-name* or the name of a variable data field that contains the sort key value.

Note: The value specified for the *sort-field-name-variable* may only be the name of a single field. If the sort key is comprised of multiple individual fields, the value specified must be that of a group-level element. If the name of a variable data field is coded and represents a group-level element, the associated elementary elements must be in the same sequence as the corresponding fields within the set's schema definition. The data formats for the elementary elements must also match the formats of the corresponding fields within the database record.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Considerations

If *autostatus* is not in use, a dialog's error-status field indicates the outcome of a FIND/OBTAIN WITHIN SET USING SORT KEY command:

Status code	Meaning
0000	The request was executed successfully
0306	Currency was not established for the named set
0308	The named record or set is not in the dialog's subschema
0310	The dialog's subschema specifies an access restriction that prohibits retrieval of the object record
0326	The specified record cannot be found

Status code	Meaning
0329	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released.
0331	No sort-control element is defined for the object record in the dialog's subschema

Further Considerations

- Before issuing a FIND/OBTAIN WITHIN SET USING SORT KEY command, the application developer must place the sort key value of the object record in the applicable field of the dialog's record buffer. If more than one record occurrence has a sort key value equal to the value in the record buffer, the first such record is accessed.
- After successful execution of a FIND/OBTAIN WITHIN SET USING SORT KEY command, the accessed record is current of:
 - The run unit
 - Its record type
 - Its area
 - All sets in which it currently participates as member or owner
- If the object record is not found, next of set and prior of set are maintained, but current of set is nullified.
- Next of set points to the next higher (for ascending sets) or next lower (for descending sets) sort key value.

Example

The statements in the following example establish a current PRODUCT-ITEM set and then retrieve an ITEM record based on the lot number:

```
MOVE 'P8' TO PROD-NUMBER.
FIND CALC PRODUCT.
MOVE 1230427 TO ITEM-LOT-NUMBER.
FIND ITEM WITHIN PRODUCT-ITEM USING ITEM-LOT-NUMBER.
```

More information:

[Error Handling](#) (see page 277)

GET

Transfers the contents of a record occurrence to a dialog's record buffer.

Elements in the object record are moved to the buffer according to the subschema view of the record.

Syntax

```

▶ GET [ record-name ] [ error-expression ] . ▶
    
```

Parameters

record-name

Retrieves the record that is current of run unit. If *record-name* is specified, current of run unit must be an occurrence of the named record type.

Record-name must be specified, unless the record or one of the record's elements is named explicitly somewhere in the dialog's process code, or the record is associated with the dialog as a map record.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Considerations

If *autostatus* is not in use, a dialog's *error-status* field indicates the outcome of a GET command:

Status code	Meaning
0000	The request was executed successfully
0508	The object record is not in the dialog's subschema
0510	The dialog's subschema specifies an access restriction that prohibits retrieval of the object record
0513	Run-unit currency was not established or was nullified by a previous ERASE command
0518	The record retrieved was not bound. The record or one of the record's elements must be named explicitly somewhere in the dialog's process code or the record must be associated with the dialog as a map record
0520	The current record of run unit is not the same type as the named record

Further Considerations

- The GET command operates only on the record that is current of run unit.
- After the successful execution of a GET command, the accessed record remains current of run unit and becomes current of its record type, its area, and all sets in which it participates as member or owner.

Example

The following example illustrates the use of the GET command to move the CUSTOMER record that is current of run unit to the dialog's record buffer:

GET CUSTOMER.

More information:

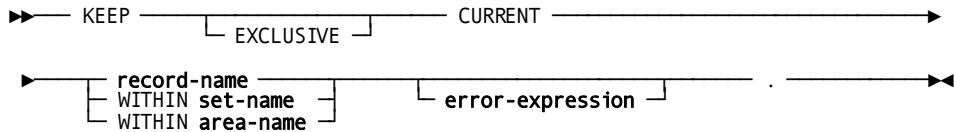
[Error Handling](#) (see page 277)

KEEP

Purpose

Places a shared or exclusive lock on a record occurrence that is current of run unit, record, set, or area.

Syntax



Parameters

EXCLUSIVE

Places an exclusive lock on the object record. If EXCLUSIVE is not specified, the object record receives a shared lock.

CURRENT

Places a lock on the current record of run unit.

record-name

Places the record lock on the current record of the named record type.

WITHIN *set-name*

Places the record lock on the current record of the named set.

WITHIN *area-name*

Places the record lock on the current record of the named area.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Definition

Record locks set with the KEEP command are maintained only for the duration of the run unit or until explicitly released by means of a COMMIT command. The COMMIT command is described earlier in this section.

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of a KEEP command:

Status code	Meaning
0000	The request was executed successfully
0606	Currency was not established for the named record, set, or area
0610	The dialog's subschema specifies a privacy lock that prohibits execution of the KEEP command
0629	Deadlock occurred during locking of target record.

More information:

[Error Handling](#) (see page 277)

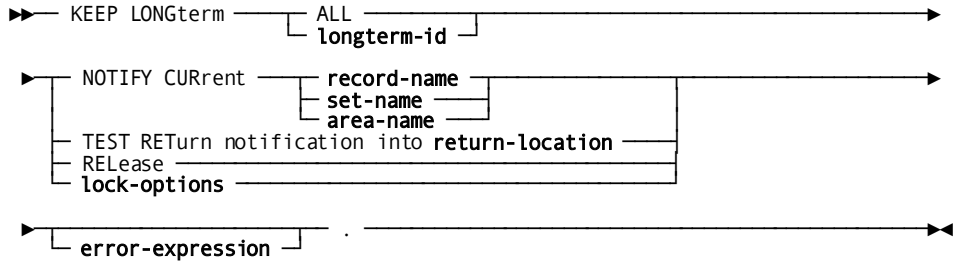
KEEP LONGTERM

Purpose

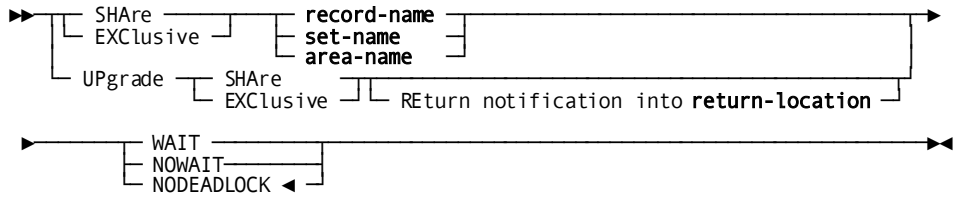
Sets or releases long-term record locks, and monitors database activity across run units.

Information on database activity can be returned to a specified location in a dialog's record buffers.

Syntax



Expansion of Lock-Options



Parameters

ALL

(Only with the RELEASE parameter) Requests release of all long-term locks associated with the current task.

longterm-id

Either the name of a 1- to 16-character variable EBCDIC data field that contains a lock identifier or the 1- to 16-character identifier itself, enclosed in single quotation marks.

Longterm-id can be used by a subsequent KEEP LONGTERM command to upgrade or release the long-term lock or to inquire about the status of database activity for the object record.

NOTIFY CURRENT

Initializes a preallocated area in the dialog's record buffer with the information written by DC/UCF on the database activity for the record identified by *longterm-id*.

record-name

Specifies monitoring of database activity for the record that is current of record type.

set-name

Specifies monitoring of database activity for the record that is current of set.

area-name

Specifies monitoring of database activity for the record that is current of area.

TEST RETURN notification into

Requests that information on database activity for the record identified by *longterm-id* be returned to the location in the dialog's record buffers specified by *return-location*.

In order to specify RETURN NOTIFICATION, a previous KEEP LONGTERM command must have included the NOTIFY CURRENT option.

return-location

The name of a binary fullword variable data field.

RELease

Releases either the long-term lock for the record identified by *longterm-id* or all long-term record locks associated with the current task.

All long-term locks that have not been released by the time the application terminates are released when the user signs off from DC/UCF with a BYE, SIGNON, or SIGNOFF command.

lock-options

Identifies the lock options.

Expanded syntax for *lock-options* is shown above immediately following the KEEP LONGTERM syntax.

error-expression

Specifies the status codes that are returned to the dialog.

SHAre

Places a long-term shared lock on the object record.

EXClusive

Places a long-term exclusive lock on the object record.

Note: The shared or exclusive lock is placed only if the area in which the record is located is readied in an update usage mode.

UPGRADE

Upgrades a longterm lock placed on the record identified by *longterm-id* during execution of a previous process.

REturn notification into

Clause requesting that information on database activity for the record identified by *longterm-id* be returned to the location in the dialog's record buffers specified by *return-location*.

WAIT

(Applies only to SHARE/EXCLUSIVE and UPGRADE) Places the run unit in a wait state if the lock cannot be placed immediately due to an existing lock on the record.

If waiting causes a deadlock, the requesting run unit terminates abnormally.

NOWAIT

(Applies only to SHARE/EXCLUSIVE and UPGRADE) Does not place the run unit in a wait state if the lock cannot be placed immediately due to an existing lock on the record. Control returns to the requesting run unit. The KEEP LONGTERM request is not executed.

NODEADLOCK

(Applies only to SHARE/EXCLUSIVE and UPGRADE) Places the run unit in a wait state. If waiting causes a deadlock, control returns to the requesting run unit and the KEEP LONGTERM request is not executed.

NODEADLOCK is the default when neither WAIT, NOWAIT, or NODEADLOCK is specified.

If WAIT or NODEADLOCK is specified, the run unit waits to place the lock only if the following conditions apply:

- The object record already holds an exclusive lock assigned by a concurrently executing run unit.
- The run unit that assigned the existing lock has readied the area in which the object record is located in an update usage mode.
- The run unit issuing the KEEP LONGTERM request has readied the area in which the object record is located in an update usage mode.

Control returns to the requesting run unit unless all of the above conditions apply.

Usage

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of a KEEP LONGTERM command:

Status code	Meaning
0000	The request was executed successfully
0032	One of the following conditions has occurred: <ul style="list-style-type: none"> ■ the lock id is already in use by the lterm, with a different page group or line index format ■ a #getstg request has failed for a lock control block

Status code	Meaning
0036	A lock manager error has occurred. Check the CV log for other error messages.
0044	A DCL1 error has occurred. Check the CV log for other error messages.
5101	NODEADLOCK was specified in the KEEP LONGTERM request and a deadlock condition occurred
5105	The requested record cannot be found or currency was not established for the object record
5121	Either of the following conditions has occurred: <ul style="list-style-type: none"> ■ The requested long-term id cannot be found ■ A KEEP LONGTERM request has been issued by a nonterminal task
5123	Area not found.
5131	Invalid param list.
5147	Area has not been readied.
5148	Run unit has not been bound.
5149	NOWAIT was specified in the KEEP LONGTERM request and a wait is required.

Further Considerations

When the database performs an action on an object record, one of five bit flags is turned on by the runtime system. If no bit is turned on, no database activity has occurred. The following table shows the bit assignments, their corresponding hexadecimal and decimal values, and the database activity they represent.

Bit assignment	Hexadecimal value	Decimal value	Database action
Fifth bit	X'10'	16	The record was physically deleted.
Fourth bit	X'08'	8	The record was logically deleted.
Third bit	X'04'	4	The record's prefix was modified; that is, a set operation occurred involving the record (for example, CONNECT, DISCONNECT).
Second bit	X'02'	2	The record's data was modified.
First bit	X'01'	1	The record was obtained.

Information about database activity that occurred for an object record is returned to a dialog as a decimal value. The action or combination of database actions taken can be determined by comparing the returned decimal value listed above to a constant. For example:

- If the returned value is 0, no database activity occurred for the record.
- If the returned value is 2, the record's data was modified.
- If the returned value is 6, both the record's data and the record's prefix were modified.
- If the returned value is 8 or greater, the record was deleted.
- If the returned value is 31 (the maximum possible value), all of the above actions occurred for the object record.

Examples

The following examples illustrate the use of `KEEP LONGTERM` to set locks and to monitor database activity:

Example 1: Setting and upgrading a lock

The following example illustrates the use of `KEEP LONGTERM` to set an exclusive lock on the current `CUSTOMER` record in one process and then to upgrade the lock to shared after the record is modified in a subsequent process:

Process A

```

.
.
.
KEEP LONGTERM LOCK-ID EXCLUSIVE CUSTOMER.
.
.
.

```

Process B

```

.
.
.
MODIFY CUSTOMER.
KEEP LONGTERM LOCK-ID UPGRADE SHARE.
.
.
.

```

By upgrading the lock to shared, other concurrently executing run units are allowed to access the `CUSTOMER` record after it has been modified.

Example 2: Monitoring database activity

The following example illustrates the use of KEEP LONGTERM to request monitoring of database activity for the current CUSTOMER record in one process and then, in a subsequent process, to test whether the record was deleted:

Process A

.
. .
.

OBTAIN CALC CUSTOMER.
KEEP LONGTERM LOCK-ID NOTIFY CURRENT CUSTOMER.

.
. .
.

Process B

.
. .
.

KEEP LONGTERM LOCK-ID TEST RETURN NOTIFICATION INTO DB-ACTIV.
IF DB-ACTIV GE 8
THEN
 INVOKE 'ORDCHECK' .

.
. .
.

More information:

[Error Handling](#) (see page 277)

MODIFY

Purpose

Replaces element values of a record occurrence in the database with new element values defined in the dialog's record buffer.

Syntax

►► MODIFY record-name error-expression . ◀◀

Parameters

record-name

Specifies the current occurrence of the named record to be modified with the values in the dialog's record buffer.

The named record must be known to the dialog's subschema.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of a MODIFY command:

Status code	Meaning
0000	The request was executed successfully
0805	Modification of the record violates a duplicates-not-allowed specification for a CALC record, sorted set, or index set
0806	Currency was not established for the object set
0809	The object record's area was not readied in an update usage mode
0810	The dialog's subschema specifies an access restriction that prohibits modification of the named record
0813	Run-unit currency was not established or was nullified by an ERASE command
0820	The current record of run unit is not the same type as the named record
0821	An area other than the area of the object record was readied with an incorrect usage mode
0825	No current record of set type was established
0829	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released
0833	Not all sorted sets in which the object record participates are included in the dialog's subschema
0855	An invalid length was defined for a variable-length record
0883	The length of a record in a native VSAM ESDS was changed or a prime key in a native VSAM KSDS was modified

Further Considerations

- The following conditions must be satisfied before a MODIFY command is executed:
 - All areas affected either directly or indirectly by the MODIFY command must be readied in an update usage mode.
- More information:**
 - Usage modes are discussed in [READY](#) (see page 423).
- The values of all elements defined for the object record in the dialog's subschema must be in the dialog's record buffer. If the MODIFY command is not preceded by an OBTAIN or GET command, the application developer must initialize the applicable values.
- The object record must be established as current of run unit.
- After successful execution of a MODIFY command, the modified record is current of:
 - The run unit
 - Its record type
 - Its area
 - All sets in which it participates as member or owner
- The following special considerations apply to the modification of CALC and sort-control elements:
 - If the modification of a CALC or sort-control element violates a duplicates-not-allowed specification in the dialog's schema, the MODIFY command is not executed and an error condition results.
 - When a CALC-control element is modified successfully, the object record can be accessed by using its new CALC key value. The database key of the object record does not change.
 - If a sort-control element is to be modified, the sorted set in which the object record participates must be included in the dialog's subschema.
 - When a sort-control element is modified successfully, any set occurrence in which the object record currently participates as a member is examined. If necessary, the object record is disconnected and reconnected in the set occurrence to maintain the sorted set order.
- The following special considerations apply to the modification of records in native VSAM data sets:
 - The length of a record in an ESDS cannot be changed even if the records are variable length.
 - The prime key of a KSDS cannot be modified.

Example

The statements in the following example retrieve an occurrence of the CUSTOMER record by using its CALC key, update the value of the CUST-NAME element in the dialog's record buffer, and then modify the record occurrence in the database:

```
MOVE IN-CUST-NUMBER TO CUST-NUMBER.
OBTAIN CALC CUSTOMER.
MOVE NEW-CUST-NAME TO CUST-NAME.
MODIFY CUSTOMER.
```

More information:

[Error Handling](#) (see page 277)

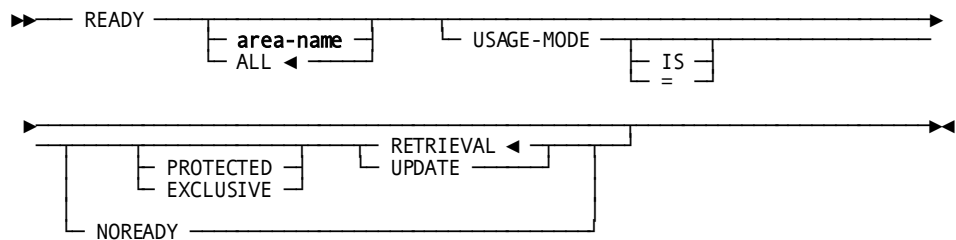
READY

Purpose

Overrides the usage mode specified in a dialog's subschema for one or more database areas.

Database areas are readied when a run unit begins (that is, immediately before the execution of the first database-access command issued by a process).

Syntax



Parameters

area-name

Readies the named area in the specified usage mode.

ALL

Readies all areas in the specified usage mode.

If neither *area-name* nor ALL is specified, all areas defined in the dialog's subschema are readied in the usage mode specified in the subschema.

USAGE-MODE

Specifies the usage mode in which the object areas are readied.

IS or = are optional keywords and have no effect on processing.

PROTECTED

Prevents concurrent update of the object areas.

EXCLUSIVE

Prevents concurrent use of the object areas.

If neither PROTECTED nor EXCLUSIVE is specified, the usage mode is qualified as shared.

RETRIEVAL

Readies the object areas for retrieval only.

RETRIEVAL is the default when neither RETRIEVAL or UPDATE is specified.

UPDATE

Readies the object areas for both retrieval and update.

NOREADY

Indicates that the area or areas named are not to be readied.

Usage

Overview of Usage Modes

Usage modes restrict runtime operations. Database areas can be readied in a retrieval or update usage mode.

- When an area is readied in a **retrieval usage mode**, the run unit cannot issue CONNECT, DISCONNECT, ERASE, MODIFY, or STORE commands for records in the area.
- When an area is readied in an **update usage mode**, the run unit can issue all database commands for records in the area.

Usage modes can be qualified as protected, exclusive, or shared.

- A database area readied in a **protected** retrieval or update usage mode cannot be updated by a concurrently executing run unit. A run unit cannot ready a database area in a protected usage mode if another run unit has readied the same area in an update usage mode.
- A database area readied in an **exclusive** retrieval or update usage mode cannot be used in any way by a concurrently executing run unit. A run unit cannot ready a database area in an exclusive usage mode if another run unit has readied the area in any usage mode.
- A database area readied in a **shared** retrieval or update usage mode can be accessed by multiple run units concurrently. If neither protected nor exclusive is specified, the usage mode is qualified as shared.

Considerations

- CA ADS automatically readies all areas defined in a dialog's subschema in the usage mode (if any) specified in the subschema, or in a shared retrieval usage mode, unless the NOREADY option is specified.
- If you add an area to a subschema with default usage mode with the FORCE option, this area is readied even if the dialog is not recompiled. However, we recommend not to perform this action for ADS applications that use extended run units.

Note: For more information on the limitations of using the FORCE option with ADS dialogs, see the Area Statement section (in the Subschema Statements chapter) in the *CA IDMS Database Administration Guide*.

- If the same area is named in more than one READY command in a process, the usage mode specified by the last READY command coded in the process applies to the named area for the entire process.
- An area cannot be readied in an update usage mode if the area includes any records that participate in a set whose members or owner are in an area readied in a retrieval usage mode.
- If a READY command results in a usage mode conflict for an area, the dialog issuing the READY command is placed in a waitstate until the area is available in the requested usage mode.

Example 1: Readying an Area

The following example illustrates the use of the READY command:

```
READY ORDOR-REGION
USAGE-MODE IS PROTECTED UPDATE.
```

Example 2: Specifying That an Area Not be Readied

The following example illustrates the use of the NOREADY option. In this example, the area CUSTOMER-REGION is readied in shared update while the area ORDOR-REGION is not readied.

```
READY CUSTOMER-REGION
USAGE-MODE IS SHARED UPDATE.
READY ORDOR-REGION
USAGE-MODE IS NOREADY.
```

More information:

[CA ADS Runtime System](#) (see page 119)

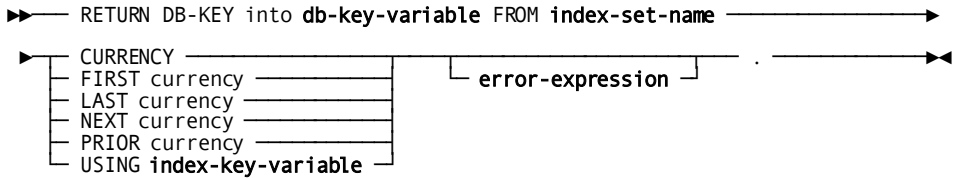
RETURN DB-KEY

Purpose

Retrieves an index entry without retrieving the associated record.

Note: This command applies only to CA IDMS system-owned indexed records.

Syntax



Parameters

RETURN DB-KEY into

Clause introducing the return of the database key, to the record associated with the specified index entry, to the location identified by *db-key-variable*.

db-key-variable

A numeric variable data field in the dialog's record buffers that can hold a binary fullword value.

Db-key-variable is a PICS9(8) COMP SYNC.

FROM index-set-name

Specifies the index set associated with the index entry being retrieved.

Note: *Index-set-name* must be known to the dialog's subschema.

CURRENCY

Retrieves the entry that is current of index.

FIRST currency

Retrieves the first entry in the index.

LAST currency

Retrieves the last entry in the index.

NEXT currency

Retrieves the entry following the current of index.

PRIOR currency

Retrieves the entry preceding the current of index.

USING *index-key-variable*

Retrieves the first index entry whose symbolic key matches the contents of *index-key-variable*.

Index-key-variable is the name of a variable data field that is not more than 256 bytes in length.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

The RETURN DB-KEY command is used as follows:

- The value of the symbolic key in the index entry is returned to the symbolic key field for the associated record in the dialog's record buffer.
- The database key that points to the record associated with the index entry is returned to a specified field in the dialog's record buffers.
- The record referenced in the RETURN DB-KEY command must be referenced in a prior DML call in the dialog's run unit.

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of a RETURN DB-KEY command:

Status code	Meaning
0000	The request was executed successfully

Status code	Meaning
1707	The end of the index was reached. Currency is set on the index owner. The DBMS returns the owner's db-key When 1707 is returned for an SPF index, currency remains on the last entry of the index. No db-key is returned.
1725	Index currency was not established with an OBTAIN command
1726	The index entry cannot be found
1729	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released

Further Considerations

- After successful execution of a RETURN DB-KEY command, the retrieved index entry is current of index.
- If an end-of-set condition is encountered, currency is set to the last or first entry in the index, based on whether next or prior currency has been requested.
- If a specified entry cannot be found, currency is set between the two entries that are higher and lower than the specified value.
- If the specified value is higher or lower than all index entries, currency is set after or before the highest or lowest entry in the index.

More information:

[Error Handling](#) (see page 277)

ROLLBACK

Purpose

Requests recovery of the part of a run unit that falls between two checkpoints (a recovery unit).

Syntax



Parameters

TASK

Specifies that database, queue, and scratch areas are recovered.

If TASK is not specified, only database areas are recovered.

CONTINUE

Rolls back the issuing run unit (ROLLBACK CONTINUE) or all run units associated with the issuing task (ROLLBACK TASK CONTINUE), but does not terminate the run unit. Database access can be resumed without issuing BIND and READY statements.

Usage

Definition

ROLLBACK performs the following functions:

- Writes an ABRT checkpoint to the CA IDMS/DB journal file.
- Nullifies all currencies.
- Terminates database activities within the process and, if no database commands are issued after the ROLLBACK command, relinquishes control over database areas. If other database commands are issued after the ROLLBACK command, the database areas are readied again automatically in the applicable usage modes.

Considerations

After successful execution of a ROLLBACK command, database, queue, and scratch areas are restored to the most recent checkpoint.

The only acceptable status code returned for a ROLLBACK command is 0000.

STORE

Purpose

Adds records to the database.

Syntax

►► STORE record-name error-expression . ◀◀

Parameters

record-name

Specifies the name of the object record occurrence to be moved from the dialog's record buffer to the database.

Record-name must be known to the dialog's subschema.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Definition

The STORE command moves the object record occurrence from the dialog's record buffer to the database and connects it to an occurrence of each set for which the record is defined as an automatic member. The STORE command performs the following functions:

- Acquires space in the database and a database key for a new record occurrence
- Transfers the values of the record elements from the dialog's record buffer to the object record occurrence in the database
- Connects the object record to all sets for which it is defined as an automatic member

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of a STORE command:

Status code	Meaning
0000	The request was executed successfully
1202	The suggested DIRECT-DBKEY value is not within the page range for the object record
1205	Storage of the record violates a duplicates-not-allowed specification for a CALC record, sorted set, or index set
1208	The object record is not in the dialog's subschema
1209	The object record's area was not readied in an update usage mode
1210	The dialog's subschema specifies an access restriction that prohibits storage of the named record
1211	The object record cannot be stored because of insufficient space
1212	The record cannot be stored because no database key is available
1221	An area other than the area of the object record was readied with an incorrect usage mode
1225	A current of set was not established for each set to which the object record is to be connected
1229	A run-unit deadlock condition occurred. DBMS aborted and rolled back the run unit. All resources associated with the task are released.
1233	Not all sets in which the object record participates as an automatic member are included in the dialog's subschema.

Status code	Meaning
1255	An invalid length was defined for a variable-length record.
1261	The record cannot be stored because of broken chains in the database.
1287	The owner and member records for a set to be updated are not in the same page group or do not have the same dbkey radix point.

Further considerations

- A record occurrence is stored in the database based on the location mode specified in the schema definition of the record:
 - **CALC**— The object record is placed on or near a database page that is calculated by CA IDMS from a control element (the CALC key) in the record.
 - **VIA**— The object record is placed as close as possible to its owner record occurrence if owner and member record occurrences share a common database page range. If owner and member record occurrences do not share a common page range, the object record is placed in the same relative position in its own page range as that in which the owner record is placed in its page range.
 - **DIRECT**— The object record is placed on or near a database page that is identified by a value moved by the application developer to the DIRECT-DBKEY field.
- Before a STORE command can be executed, the following conditions must be satisfied:
 - All areas affected either directly or indirectly by the STORE command must be readied in an update usage mode.
 - All control elements (that is, CALC and sort keys) must be initialized.
 - If the object record has a location mode of DIRECT, the DIRECT-DBKEY field must be initialized with a suggested database key value or a null database key value of -1.
 - If the object record is to be stored in a native VSAM RRDS, the DIRECT-DBKEY field must be initialized with the relative record number that represents the location within the data set where the record is to be stored.
 - All sets in which the object record is defined as an automatic member and the owner record of each of those sets must be included in the dialog's subschema.
 - If the object record has a location mode of VIA, currency must be established for the owner of the set through which the record is stored, regardless of whether the record is an automatic or manual member of the set.

- Currency must be established for all set occurrences for which the object record is defined as an automatic member. A STORE command connects the object record to a set occurrence, based on set order, as follows:
 - If the object record is defined as a member of a set that is ordered FIRST, the object record is connected right after the owner to become the first member of the set. If the set is ordered LAST, the object record is connected as the last member of the set.
 - If the object record is defined as a member of a set that is ordered NEXT or PRIOR, the record that is current of set establishes the set occurrence to which the object record is connected and determines the record's position within the set.
 - If the object record is defined as a member of a sorted set, the process must establish currency on the set by getting currency on the set's owner. Then, the process can store the object record. CA IDMS/DB automatically connects the object record to the correct position in the set in order to maintain the proper set sequence.

The sort key of the object record is compared with the sort key of the record that is current of set to determine if the object record can be inserted in the set by movement in the next direction. If it can, current of set remains unchanged and the object record is connected. If it cannot, current of set is repositioned at the owner record occurrence (not necessarily the current occurrence of the owner record type) and movement proceeds in the next direction until the object record can be properly connected.

- After successful execution of a STORE command, the object record becomes current of:
 - The run unit
 - Its record type
 - Its area
 - All sets in which it participates as owner or automatic member

Example

The statements in the following example store a new ITEM record in the database and connect it to the correct occurrences of the ORDER-ITEM and PRODUCT-ITEM sets:

```
MOVE IN-PROD-NUMBER TO PROD-NUMBER.  
FIND CALC PRODUCT.  
MOVE IN-ORD-NUMBER TO ORD-NUMBER.  
FIND CALC ORDOR.  
STORE ITEM.
```


More information:

[Error Handling](#) (see page 277)

[READY](#) (see page 423)

Logical Record Facility Commands

In CA ADS, Logical Record Facility (LRF) commands are used to retrieve and update data that is defined in a Logical Record Facility subschema.

Overview of LRF Database Access

To enable use of LRF, DBAs predefine the paths that a dialog can use to access specific views of data in the database. Logic to navigate the database is contained in the path definition.

Given the dialog's data requirements, the programmer selects the appropriate LRF path and then codes database requests in the form of LRF commands within the dialog's process logic. At runtime, CA IDMS/DB locates the requested data using the specified path.

Components of LRF

LRF processes commands associated with logical records. When a dialog issues an LRF command, LRF selects an appropriate path based on the information in the command statement. LRF uses field values in the record buffer that is established for the logical record to update the database.

Logical records are defined in a subschema by the database administrator (DBA). Each logical record is composed of fields selected from one or more subschema records or roles that are typically accessed together.

Logical Record Facility paths are also defined in the subschema. Each path is a group of database access instructions that perform the processing necessary to satisfy an LRF request. One or more paths are associated with each logical record in the subschema.

The predefined **conditions affecting logical record access** include:

- **Restrictions** on the commands that can be issued for each logical record
- **Selection** criteria that can be specified by a WHERE clause in each command for each logical record
- **Path statuses** returned by LRF to indicate the result of each command

To use Logical Record Facility commands effectively, the application developer must be familiar with the processing characteristics of the logical records that are defined in the subschema.

Note: For more information about using Logical Record Facility, see the *CA IDMS Logical Record Facility*.

Process code within a single dialog cannot reference more than one logical record that includes fields from a given subschema record.

WHERE Clause

A WHERE clause is used to specify criteria for selection of one or more occurrences of a logical record that is the object of an ERASE, MODIFY, OBTAIN, or STORE command. A WHERE clause is also used to direct LRF to a particular logical record path.

Considerations

- A WHERE clause is specified in the form of an expression that consists of one or more conditions to be tested.
 - Multiple conditions are combined with the logical operators AND and OR.
 - The logical operator NOT can precede a single condition or a compound condition that is enclosed in parentheses. NOT specifies the opposite of the condition.
- A test condition is expressed as a comparison or a keyword.
- A logical record occurrence is selected only if the entire expression evaluates as true.
- Operators in a conditional expression are evaluated one at a time, from left to right, in order of precedence.

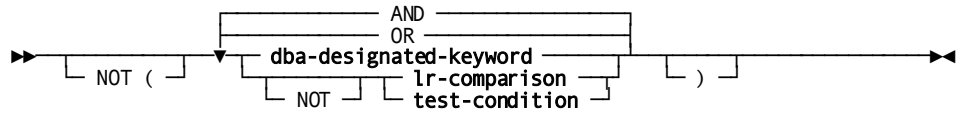
Note: The default order of precedence is the same as that described for other conditional expressions discussed in [Conditional Expressions](#) (see page 245).

Conditional Expression

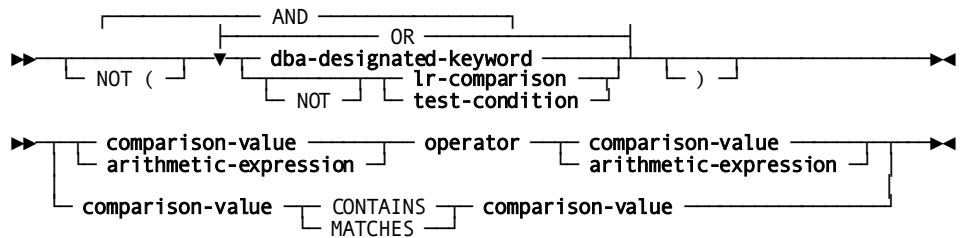
Purpose

The conditional expression of the WHERE clause is used when the process command syntax specifies *lr-conditional-expression*.

Syntax



Syntax: Logical Record Expression



Parameters

NOT

Specifies that the opposite of the 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.

dba-designated-keyword

Specifies a keyword, defined in the subschema by the database administrator (DBA), that directs LRF to a particular logical record path. The selected path must be associated with the object logical record.

lr-comparison

Specifies a comparison expression that establishes criteria used to select occurrences of the object logical record.

Syntax for the comparison expression is shown later in this chapter.

test-condition

Specifies a condition to be tested, such as command status or cursor position.

AND

Specifies that the expression is true only if the outcome of both of the conditions being tested is true.

OR

Specifies that the expression is true if the outcome of either one or both of the conditions being tested is true.

More information:

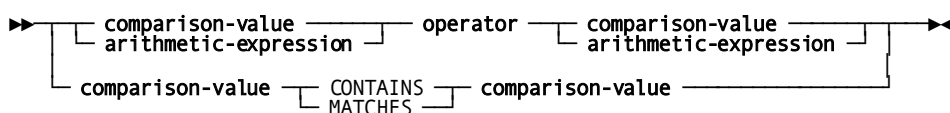
[Conditional Expressions](#) (see page 245)

Comparison Expression

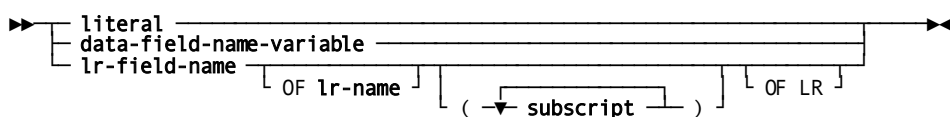
Purpose

Used to compare two values or to compare two character strings to determine if the first string matches or contains the second string.

Syntax



Expansion of Comparison-Value



Expansion of Comparison-Value

Parameters

comparison-value

Specifies the value to be compared.

Expanded syntax for *comparison-value* is shown above immediately following the comparison expression syntax.

arithmetic-expression

Specifies an arithmetic expression, according to the rules presented in [Arithmetic Expressions](#) (see page 171).

operator

The comparison operators are:

Operator	Synonym	Meaning
EQ	=	Equal
NE		Not equal to
GT	>	Greater than

Operator	Synonym	Meaning
LT	<	Less than
GE		Greater than or equal to
LE		Less than or equal to

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, and both operands must be EBCDIC or unsigned zoned decimal data types. If the right operand is not entirely contained in the left operand, the outcome of the comparison is false.

MATCHES

Compares the left operand to the right operand, one character at a time, beginning with the leftmost character in each operand. The right operand can contain mask characters, as follows:

- @ -- Matches any alphabetic character
- # -- Matches any numeric character
- * -- Matches any character

Any other character in the right operand matches only itself in the left operand.

literal

A user-supplied variable, expressed as a numeric constant, or the character string itself, enclosed in single quotation marks.

data-field-name-variable

Specifies the name of a variable data field, according to the rules presented in [Variable Data Fields](#) (see page 285).

lr-field-name

Specifies the name of a field in a Logical Record Facility record known to the subschema associated with the dialog.

OF lr-name

Specifies the name of the record that contains the field referenced by *lr-field-name*.

This clause is required only if the named field is not unique among the records known to the dialog.

subscript

Specifies the applicable occurrence of the field referenced by *lr-field-name*. This can be a variable field containing the applicable occurrence, the occurrence itself, or an expression.

This clause applies only if the named field is defined as a multiply-occurring field.

OF LR

Specifies that the value of the named field at the time that the request is issued is used throughout processing of the request.

If the value of the field changes during processing, LRF continues to use the original value. If OF LR is not specified and the value of the field changes during processing of the request, the new value in the dialog's record buffer is used for any further processing of the request.

Usage

Considerations

Both the left and right operands must be EBCDIC or unsigned zoned decimal data types. 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.

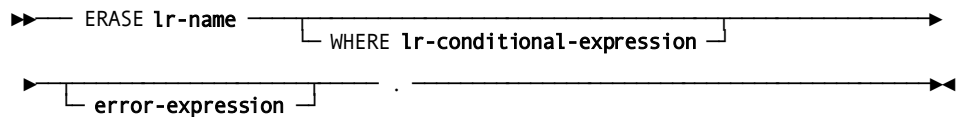
ERASE

Purpose

Deletes record occurrences.

The execution of an ERASE command does not necessarily result in the deletion of all or any of the database records used to create the object Logical Record Facility database-access record. The path selected to service the ERASE request performs only the database-access operations specified in the subschema.

Syntax



Parameters

lr-name

Specifies occurrences of the logical record used for database access.

lr-name must be known to the dialog's subschema.

WHERE *lr-conditional-expression*

Specifies the selection criteria to be applied to the logical record request.

Syntax for *lr-conditional-expression* is described earlier in this section.

error-expression

Specifies the status codes that are returned to the dialog.

Example

The ERASE command in the following example deletes the occurrence of CUST-ORDER-LR with the specified customer and order numbers:

```
ERASE CUST-ORDER-LR
  WHERE CUST-NUMBER EQ '1234567890'
  AND ORD-NUMBER EQ '7654321'
  AND DELETE-ORDER.
```

The DBA-designated keyword, DELETE-ORDER, directs processing to a path that retrieves the applicable occurrence of the CUST-ORDER-LR logical record and deletes the specified order information from the database.

More information:

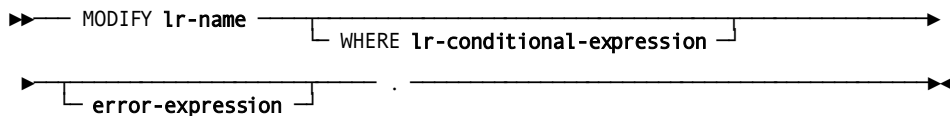
[Error Handling](#) (see page 277)

MODIFY

Purpose

Modifies field values in a record occurrence.

Syntax



Parameters

lr-name

Specifies the name of the logical record.

lr-name must be known to the dialog's subschema.

WHERE *lr-conditional-expression*

Specifies the selection criteria to be applied to the logical record request.

Syntax for *lr-conditional-expression* is described earlier in this section.

error-expression

Specifies the status codes that are returned to the dialog.

Example

The statements in the following example update an occurrence of the logical record CUST-ORDER-LR by specifying a new customer name and a new required date for the associated order:

```
OBTAIN FIRST CUST-ORDER-LR
  WHERE CUST-NUMBER EQ '1234567890'
  AND ORD-NUMBER EQ '7654321'.
MOVE NEW-CUST-NAME TO CUST-NAME.
MOVE NEW-DATE-REQ TO ORD-DATE-REQ.
MODIFY CUST-ORDER-LR.
```

More information:

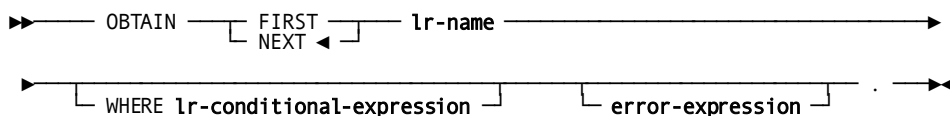
[Error Handling](#) (see page 277)

OBTAIN

Purpose

Retrieves logical record occurrences.

Syntax



Parameters

FIRST

Retrieves the first occurrence of the named logical record that meets the selection criteria specified in the WHERE clause.

NEXT

Retrieves the next occurrence of the named logical record that meets the selection criteria specified in the WHERE clause.

NEXT is the default when neither FIRST or NEXT is specified.

If the same selection criteria were not specified in a previous OBTAIN command, OBTAIN NEXT is equivalent to OBTAIN FIRST.

lr-name

Specifies the name of the logical record

lr-name must be known to the dialog's subschema.

WHERE *lr-conditional-expression*

Specifies the selection criteria to be applied to the logical record request.

Syntax for *lr-conditional-expression* is described earlier in this section.

error-expression

Specifies the status codes that are returned to the dialog.

Usage***Definition***

Data from object logical-record fields is transferred to the buffer established in the dialog's record buffers. The OBTAIN command can be issued iteratively to retrieve a series of record occurrences that meet the criteria specified in a WHERE clause.

Example

The statements in the following example retrieve all occurrences of the logical record CUST-ORDER-LR for customer 1234567890:

```
OBTAIN FIRST CUST-ORDER-LR
  WHERE CUST-NUMBER EQ '1234567890'.
ON LR-NOT-FOUND THEN INVOKE 'CUSTCHEK'.
ON LR-FOUND
  REPEAT.
    OBTAIN NEXT CUST-ORDER-LR
      WHERE CUST-NUMBER EQ '1234567890'.
    .
    .
    .
  END.
DISPLAY.
```

More information:

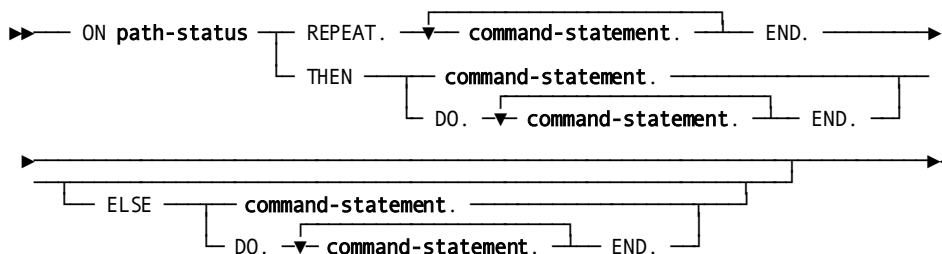
[Error Handling](#) (see page 277)

ON Command

Purpose

Indicates additional processing to be performed when a specified path status is returned by the Logical Record Facility following the execution of an LRF command.

Syntax



Parameters

path-status

Tests whether Logical Record Facility returned the named path status. *Path-status* specifies a 1- to 16-character DBA-defined or standard path status defined for the path selected to service the previous logical record request.

REPEAT *command-statement*

Specifies the commands to be executed as long as LRF returns the named path status.

REPEAT begins a processing loop; END terminates the loop. Each command is executed sequentially before the path status is tested again.

Command-statement can be any valid CA ADS process command, including another logical record command.

THEN *command-statement*

Specifies the commands to be executed if LRF returns the named path status.

Note: Multiple command statements must be preceded by DO and followed by END.

Command-statement can be any valid CA ADS process command, including another logical command.

ELSE *command-statement*

Specifies the commands to be executed if LRF returns the named path status.

Command-statement can be any valid CA ADS process command, including another logical record command.

Note: Multiple command statements must be preceded by DO and followed by END.

A given ON command statement can include only one ELSE clause, and that ELSE clause must match the most recent ON command not associated with an ELSE clause.

Usage*Path Statuses*

A path status, in the form of a 1- to 16-character unquoted string, indicates the result of an LRF request. LRF can return either a path status defined by the DBA in the subschema associated with the dialog or one of the standard path statuses. The standard path statuses are:

- **LR-FOUND** indicates that the logical record request was executed successfully. When LR-FOUND is returned, the dialog's error-status field contains 0000.
- **LR-NOT-FOUND** indicates that the object record cannot be found either because no such record exists or because all occurrences of the record have already been retrieved. When LR-NOT-FOUND is returned, the dialog's error-status field contains 0000.
- **LR-ERROR** indicates that a logical record request was issued incorrectly or that an error occurred in the processing of the path selected to service the request. When LR-ERROR is returned, the dialog's error-status field contains one of the status codes listed below.

Status code	Meaning
2001	The requested logical record was not found in the subschema (The path DML statement, EVALUATE, returns 0000 if true and 2001 if false)
2008	The object record is not in the dialog's subschema, or the specified request is not permitted for the named record
2010	The dialog's subschema prohibits access to logical records
2040	The WHERE clause in an OBTAIN NEXT command directed LRF to a different processing path than did the WHERE clause in the preceding OBTAIN command for the same logical record
2041	The request's WHERE clause cannot be matched to a path in the dialog's subschema

Status code	Meaning
2042	The logical record path for the request specifies return of the LR-ERROR status to the process
2043	Bad or inconsistent data was encountered in the logical record buffer during evaluation of the request's WHERE clause
2044	The request's WHERE clause does not include data required by the logical record path
2045	A subscript value in a WHERE clause is either less than zero or greater than its maximum allowed value
2046	One of the following conditions occurred during the evaluation of a WHERE clause: <ul style="list-style-type: none"> ■ Arithmetic overflow (fixed point, decimal, or exponent) ■ Arithmetic inflow (exponent) ■ Divide exception (fixed point, decimal, or floating point) ■ Significance exception
2063	The request's WHERE clause contains a keyword that exceeds the 16-character maximum
2072	The request's WHERE clause is too long to be evaluated in the available work area

Considerations

- One or more process commands can be specified to be executed once or iteratively, based on the returned path status. If an iterative sequence is used, the path status must change during processing to prevent uncontrolled looping.
- ON commands can be nested to any level and can be included in IF and WHILE command structures.
- When coding ON commands, indentation should be used wherever possible to make the statement more readable and to ensure that the required clauses are properly matched.

Examples

The following examples test the path status before performing additional processing.

Example 1: Displaying messages when a record is not found

The statements in the following example display messages based on the path status returned after an attempt is made to retrieve a CUST-ORDER-LR logical record:

```
OBTAIN CUST-ORDER-LR
  WHERE CUST-NUMBER EQ '1234567890'.
ON NO-CUSTOMER
THEN
  DISPLAY MSG TEXT IS 'CUSTOMER NOT ON FILE'.
ON NO-ORDER
THEN
  DISPLAY MSG TEXT IS 'CUSTOMER HAS NO ORDERS'.
ON LR-NOT-FOUND
THEN
  DISPLAY MSG TEXT IS 'RECORD NOT FOUND'.
```

Example 2: Retrieving a record after a specified record

The statements in the following example retrieve VENDOR-LR logical records as long as the path status returned after the previous retrieval is VENDOR-CODE-010:

```
OBTAIN FIRST VENDOR-LR.
ON VENDOR-CODE-010
  REPEAT.
    OBTAIN NEXT VENDOR-LR.
    .
    .
    .
  END.
DISPLAY.
```

More information:

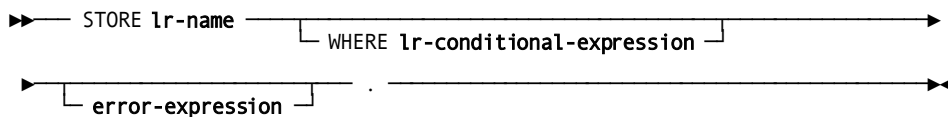
[Conditional Commands](#) (see page 317)

STORE

Purpose

Stores new occurrences of logical records.

Syntax



Parameters

lr-name

Specifies the name of the Logical Record Facility record.

lr-name must be known to the dialog's subschema.

WHERE *lr-conditional-expression*

Specifies the selection criteria to be applied to the logical record request.

Syntax for *lr-conditional-expression* is described earlier in this section.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Considerations

The execution of a STORE command does not necessarily result in new occurrences of all or any of the database records used to create the object logical record. The path selected to service the STORE request performs only the database access operations specified in the subschema.

For example, CUST-ORDER-LR comprises fields from the CUSTOMER, PRODUCT, ORDOR, and ITEM records. A new CUST-ORDER-LR logical record is stored for each new customer order; however, only new occurrences of the ORDOR and ITEM records are actually added to the database. The CUSTOMER and PRODUCT records already exist in the database.

Example

The statements in the following example store a new occurrence of the logical record CUST-ORDER-LR for customer 1234567890. The DBA-designated keywords NEW-ORDER and NEW-ITEM direct LRF to the logical record paths that store new order and new item information, respectively.

```
MOVE ORDER-NEW TO ORDOR.  
STORE CUST-ORDER-LR  
  WHERE CUST-NUMBER EQ '1234567890' AND NEW-ORDER.  
  .  
  .  
  .
```

```
MOVE ITEM-NEW TO ITEM.  
MODIFY CUST-ORDER-LR  
  WHERE NEW-ITEM.
```

More information:

[Error Handling](#) (see page 277)

Chapter 17: Map Commands

This section contains the following topics:

[Overview](#) (see page 449)

[Map Modification Commands](#) (see page 450)

[Attributes Command](#) (see page 450)

[CLOSE](#) (see page 454)

[MODIFY MAP](#) (see page 455)

[Pageable Maps](#) (see page 464)

Overview

Online maps (CA ADS) and file maps (CA ADS Batch) are created and stored in the data dictionary using the CA IDMS mapping facility. Map modification commands change the copy of the map maintained for a particular dialog, not the stored map definition.

Note: For more information about maps and map attributes, see the *CA IDMS Mapping Facility Guide*.

Map Commands

CA ADS map commands are used to adjust maps to meet the processing requirements of individual dialogs at run time. Pageable map commands are used to create, retrieve, and modify detail occurrences of a pageable map.

The map modification and pageable map commands are summarized in the following table. Each command is presented alphabetically later in this section.

Summary of Map Modification and Pageable Map Commands

Type	Command	Description
Map modification commands	Attributes	Modifies the display intensity or the protected/unprotected specification of one or more map data fields, providing an alternative format to the MODIFY MAP command for these attributes
	CLOSE	Closes the dialog input and output file maps (batch only)
	MODIFY MAP	Modifies a map's write control character (WCC) options and specifies attributes of one or more map data fields

Type	Command	Description
Pageable map detail commands	GET DETAIL	Retrieves a modified detail occurrence
	PUT DETAIL	Creates or modifies a detail occurrence

Map Modification Commands

Map modification commands are used to change a map to meet processing requirements of individual dialogs at run time. Single or multiple attributes can be changed globally or on a field-specific basis. Requested map modifications can be designated as temporary or permanent. Temporary changes apply only to the next display of the map. Permanent changes apply as long as the dialog remains operative in the application thread.

Pageable Map Considerations

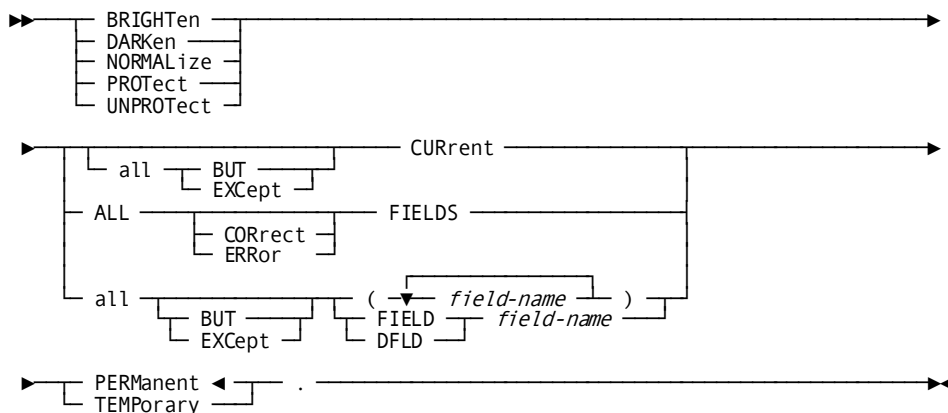
For a pageable map, the following considerations apply:

- Permanent map modifications to detail area map fields modify only detail occurrences referenced by subsequent PUT DETAIL commands.
- Temporary map modifications to detail area map fields modify only the detail occurrence referenced by the next PUT DETAIL command. If temporary modifications are to apply in subsequent PUT DETAIL commands, the appropriate map modification commands must be repeated.
- Temporary map modifications to header and footer map data fields apply only to the first display of the map following the map modification. Temporary map modifications to fields in a detail occurrence apply only to the first display of that occurrence following map modification.

Attributes Command

Purpose

Modifies a map attribute for one or more map fields.

Syntax**Parameters****BRIGHTen**

Displays the specified map fields at brighter-than-normal intensity. A brightened field appears highlighted on the terminal screen.

DARKen

Displays the specified map fields at darker-than-normal intensity. Characters in a darkened field do not appear on the terminal screen.

NORMALize

Displays the specified map fields at normal intensity.

PROTECT

Enables the input protect attribute for the specified map fields. The user cannot enter, modify, or delete data in the specified fields.

UNPROTECT

Disables the input protect attribute for the specified map fields. The user can enter, modify, or delete data in the specified fields.

CURrent

Modifies the current map data field only. The current map data field is determined by the most recent map modification command or map field status condition test:

- **Map modification command**— The current field is the last field named in an explicit list of map fields or the last map field modified in an implicit list.

An implicit list results from specifying the FOR ALL BUT clause or the FOR ALL CORRECT/ERROR FIELDS clause in the map modification command. An implicit list of map fields is ordered in the sequence in which the fields are defined in the map that is, top to bottom, left to right).

- **Map field status condition test**— The current field is the last field tested in the explicit or implicit list of fields. An implicit list results from specifying the ALL/ANY/NONE/SOME FIELDS clause in the map field status condition test.

The runtime system tests the fields in an explicit list from left to right, and tests the fields in an implicit list in the order in which the fields are defined in the map (that is, top to bottom, left to right).

Note that a status condition test ends at the first map field that determines the result of the test. For example, a test is run to determine if any fields in a list are truncated; the test stops at the first field that is truncated, and that field becomes the current map field.

all BUT

Modifies all map data fields except the current field.

EXCEPT can be used in place of BUT.

ALL FIELDS

Means that the attribute coded in this command will be applied to all data fields on the map, unless either of the optional positional parameters CORRECT and ERROR are specified.

CORRECT

Means that the attribute coded in this command will be applied during the next error display to all data fields that have NOT been marked as 'IN ERROR'.

ERROR

Means that the attribute coded in the command will be applied during the next error display to all data fields that have been marked as 'IN ERROR'.

If CORRECT or ERROR is not specified, all map data fields are modified.

all BUT

Introduces the fields to be modified.

The optional keyword BUT modifies all map data fields except the field or fields specified by *field-name*.

EXCEPT can be used in place of BUT.

FIELD *field-name*

Specifies the map data field to be modified.

DFLD can be used in place of FIELD.

PERManent

Specifies permanent modification.

The modification applies to each display of the map associated with the current dialog as long as the dialog remains operative in the application thread. In a pageable map, a modification to a map data field of a detail line occurrence applies throughout the map paging session.

PERMANENT is the default when neither TEMPORARY or PERMANENT is specified.

TEMPorary

Specifies temporary modification.

The modification applies only to the next display of the map associated with the current dialog. In a pageable map, a modification to a map data field of a detail line occurrence applies only to the next time the detail line occurrence is displayed on the screen during the map paging session.

Usage*Definition*

The attributes command provides an alternative format to the MODIFY MAP command for modification of display intensity or protected status of one or more map data fields. Only one attribute can be specified in a single attribute command.

The MODIFY MAP command can be used to modify multiple attributes. MODIFY MAP is discussed later in this section.

Example

The statements in the following example make up part of a response process that adds a new CUSTOMER record occurrence to the database. If the user enters a customer number that is already assigned, the screen is redisplayed with the CUST-NUMBER field in bright intensity:

```
FIND CALC CUSTOMER.  
IF DB-STATUS-OK  
THEN  
  DO.  
    BRIGHTEN FIELD CUST-NUMBER TEMPORARY.  
    DISPLAY MESSAGE TEXT IS  
    'CUSTOMER NUMBER ALREADY ASSIGNED. '  
  END.  
ELSE  
  STORE CUSTOMER.  
  DISPLAY MESSAGE TEXT IS  
  'CUSTOMER HAS BEEN ADDED. '
```

More information:

[Conditional Expressions](#) (see page 245)

CLOSE

Purpose

(CA ADS Batch only) Closes the dialog input and output file maps.

Syntax

```
▶▶ CLOSE [ BOTH | INPUT | OUTPUT ] file MAPs . ▶▶
```

Parameters**BOTH**

Specifies the dialog's input and output file maps.

BOTH can be specified even if the dialog has only an input or an output file map.

BOTH is the default when no other option is specified.

INPUT

Specifies the dialog's input file map.

OUTPUT

Specifies the dialog's output file map.

Usage*Considerations*

- The runtime system automatically closes the files if an application terminates with files still open:
 - A CLOSE command logically closes a file only if other dialogs using different maps have accessed the same file.
 - A CLOSE command must be issued for each map to physically close a file.
- The CLOSE command is required when closing a file before the application terminates, as in the following cases:
 - The application has been reading from or writing to a file and is required to start over at the beginning of the file.

- An output file to which records were written is to be read as an input file.
- A run-unit commit is performed by a COMMIT command or at the end of a run unit.

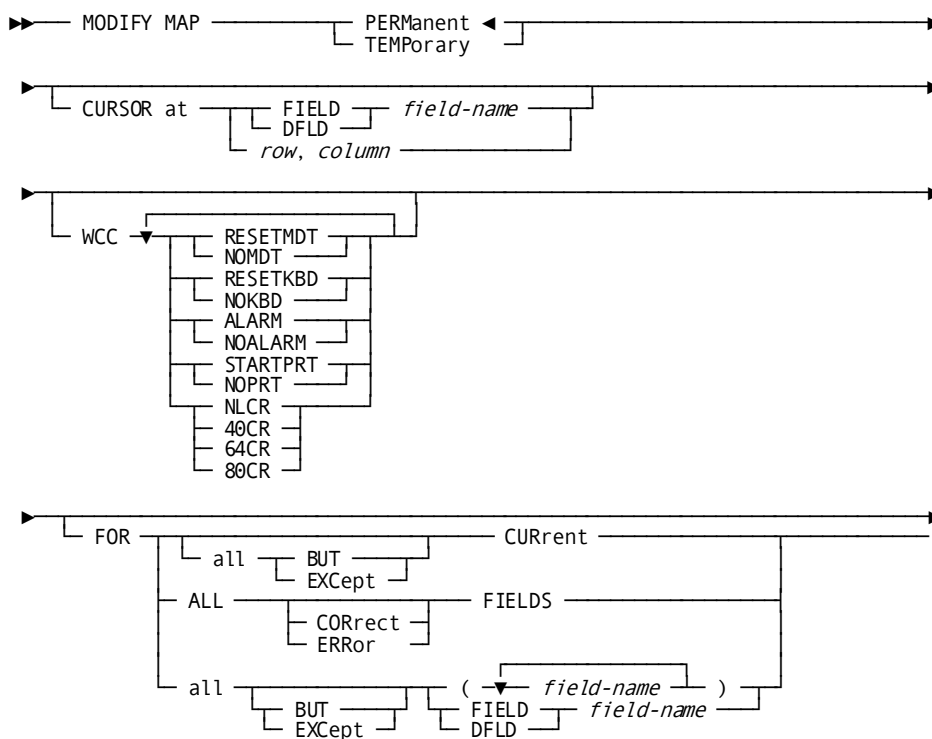
If a COMMIT command is issued, but not all files used in the application are closed, the runtime system either takes no action, sends a warning message to the log, or abends the application, as specified at system generation or at run time. The default action is abend.

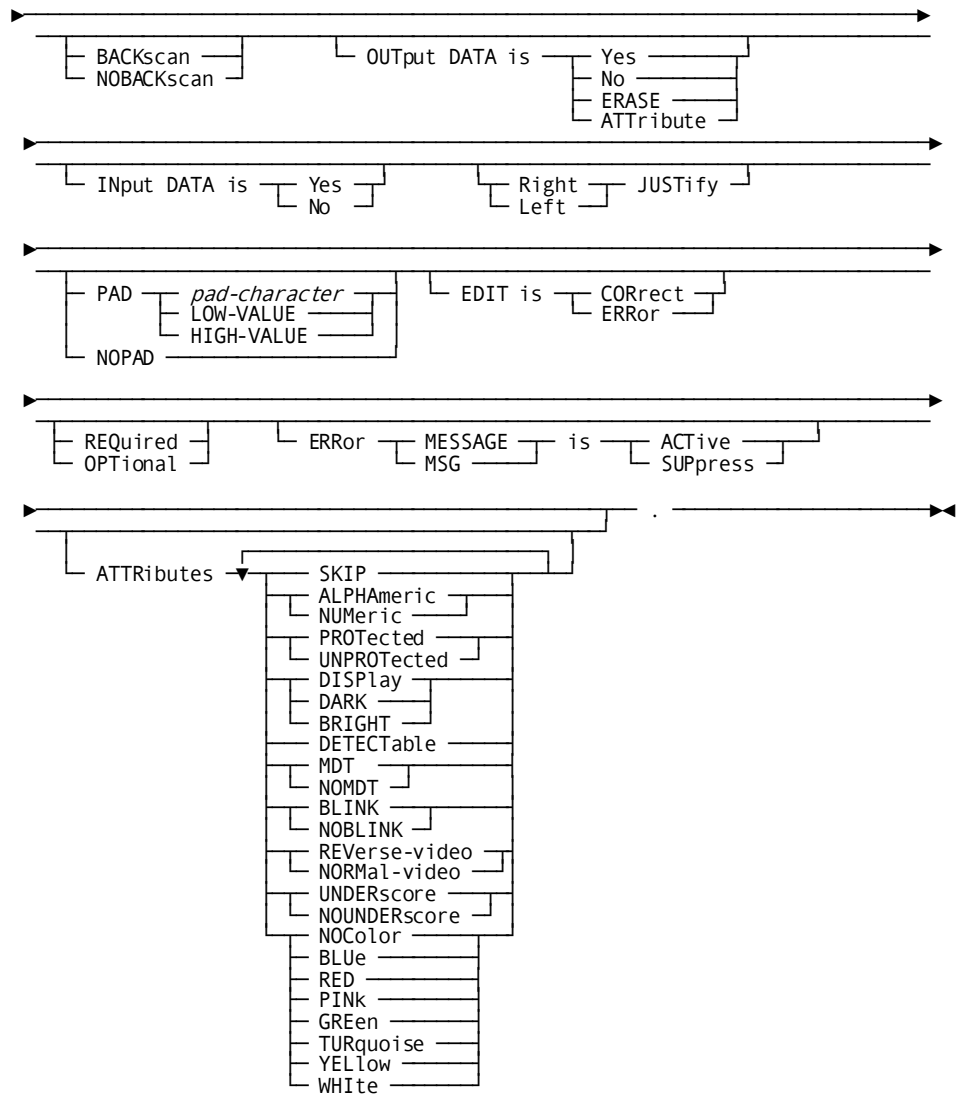
MODIFY MAP

Purpose

Modifies a map write control character (WCC) options and specifies attributes for map data fields.

Syntax





Parameters

PERManent

Specifies permanent modification.

The modifications apply to each display of the map as long as the dialog remains operative in the application thread. In a pageable map, a modification to a map data field of a detail line occurrence applies throughout the map paging session.

PERMANENT is the default when neither TEMPORARY nor PERMANENT is specified.

TEMPorary

Specifies temporary modification.

The modifications apply only to the next display of the map. In a pageable map, a modification to a map data field of a detail line occurrence applies only to the next time the detail line occurrence is displayed on the screen during the map paging session.

CURSOR

Specifies the cursor position on the terminal screen when the map is displayed.

FIELD *field-name*

Positions the cursor at the beginning of the named map field.

Field-name specifies a data field in the map associated with the current dialog.

DFLD can be used in place of FIELD.

row

Either the name of a variable data field that contains the row number or the row number itself, expressed as a numeric constant.

column

Either the name of a variable data field that contains the column number or the column number itself, expressed as a numeric constant.

The specified row and column numbers must be 1- to 16-digit unsigned integers and must be valid for the terminal in use. The row and column specifications must be separated by a blank or a comma.

WCC

Modifies the write control character (WCC) specifications for the map associated with the current dialog.

RESETMDT

The modified data tags (MDTs) are turned off.

NOMDT

The modified data tags (MDTs) are not turned off.

An MDT marks a data field for transmission to the dialog whether or not it is modified by the user.

RESETKBD

The keyboard is unlocked when the map is displayed.

NOKBD

The keyboard remains locked when the map is displayed.

ALARM

If installed, the terminal's audible alarm will sound when the map is displayed.

NOALARM

Even if installed, the terminal's audible alarm will not sound when the map is displayed.

STARTPRT

The contents of the terminal buffer are printed when the map is displayed.

NOPRT

The contents of the terminal buffer are not printed when the map is displayed.

Note: This specification is meaningful only when a 3280-type printer is in use.

NLCR

No line formatting is performed on the printer output. The printer advances to a new line only when the new line (NL) and carriage return (CR) characters occur.

40CR

Printer output is formatted into 40 characters per line.

64CR

Printer output is formatted into 64 characters per line.

80CR

Printer output is formatted into 80 characters per line.

Note: This specification is meaningful only if the STARTPRT option above is specified.

If the MODIFY MAP command is used to alter any WCC option, *all* WCC options are overridden by the command. Unspecified WCC options default, as follows:

- RESETMDT/NOMDT defaults to NOMDT.
- RESETKBD/NOKBD defaults to NOKBD.
- ALARM/NOALARM defaults to NOALARM.
- STARTPRT/NOPRT defaults to NOPRT.
- NLCR/40CR/64CR/80CR has no default.

FOR

Specifies the map data fields being modified.

all BUT CURrent

Modifies all map data fields except the current field.

EXCEPT can be used in place of BUT.

ALL FIELDS

Introduces which map data fields are to be modified.

CORrect

Modifies all map data fields set to be correct by the automatic error-handling facility or the dialog.

ERRor

Modifies all map data fields set to be correct by the automatic error-handling facility or the dialog.

If CORRECT or ERROR is not specified, all map data fields are modified.

all BUT

Introduces the fields to be modified.

The optional keyword BUT modifies all map data fields except the field or fields specified by *field-name*.

EXCEPT can be used in place of BUT.

FIELD *field-name*

Specifies the map data field to be modified.

DFLD can be used in place of FIELD.

BACKscan

The contents of the designated map fields are displayed without trailing blanks. Characters remaining from the previous display of the map may appear in any unused positions.

NOBACKscan

The contents of the designated map fields are displayed with trailing blanks.

OUTput DATA is

Clause introducing selections which determine whether data from the dialog's record buffers and attribute specifications are transmitted to the designated map fields when the map is displayed. Attribute specifications include all attributes that can be specified in conjunction with the ATTRIBUTES keyword of the MODIFY MAP command.

Yes

Data and attribute specifications are transmitted.

No

Data is not transmitted. Data remaining from the previous display of the map appears in the designated map fields. Attribute specifications for a designated map field are transmitted only if one of the following conditions is met:

- The map being displayed is different than the map previously displayed.
- The designated map field is in error.

ERASE

Data is not transmitted, and data remaining from the previous display of the map is erased from the designated map fields. Attribute specifications are transmitted.

ATtribute

Attribute specifications are transmitted, but data is not. Data remaining from the previous display of the map appears in the designated map fields.

INput DATA is

Clause introducing selections which determine whether data entered in the specified map fields is transmitted to the dialog's record buffers.

Yes

Data in the designated map fields is transmitted to the dialog's record buffers.

No

Data in the designated map fields is not transmitted to the dialog's record buffers.

JUSTify

Introduces how data entered in the designated map fields is justified before it is transmitted to the dialog's record buffers.

Note: This specification is meaningful for nonnumeric fields only.

Right

Data in the designated map fields is right justified.

Left

Data in the designated map fields is left justified.

PAD *pad-character*

Specifies whether data entered in the designated map fields is padded before it is transmitted to the dialog's record buffers.

Data is padded on the left (if RIGHT JUSTIFY is specified) or on the right (if LEFT JUSTIFY is specified) with the specified pad character.

Pad-character is either the name of a variable data field that contains the pad character or the actual pad character, enclosed in single quotation marks.

NOPAD

Data in the designated map fields is not padded.

EDIT is

Specifies whether an error flag is set for the designated map fields.

ERRor

An error flag is set for the designated map fields.

CORrect

No error flag is set for the designated map fields.

Note: Error flags cannot be set permanently.

On a mapout operation, if any field is flagged to be in error, then for all fields both correct and incorrect) only attribute bytes are transmitted; no data is moved from program variable storage to the screen.

There is one exception to the above rule: on the initial display of a map by an CA ADS dialog, all literals and data fields are transmitted even if a field is in error.

REQuired

The user must enter data in the designated map fields.

OPTional

The user can enter data in the designated map fields, as applicable.

ERRor MESSAGE is

Specifies display or suppression of an error message associated with a field.

ACTive

Enables display of an error message.

A message is usually enabled after `ERROR MESSAGE SUPPRESS` is specified within a `MODIFY MAP PERMANENT` specification.

SUPpress

Disables display of an error message associated with a field.

When the map is redisplayed because of errors, the message defined for the map field will not be displayed even if the field contains edit errors.

Note: Autoedit errors detected on map in for detail fields within a pageable map cannot be suppressed unless you turn off autoedit.

ATTRIBUTES

Applies 3270- and 3279-type terminal display attributes to the designated map fields.

SKIP

Causes repositioning of the cursor over the designated map fields to the next unprotected field.

SKIP automatically assigns the NUMERIC and PROTECTED attributes (see below) to the designated map fields.

ALPHAMeric

The user can enter any data type characters.

Note: ALPHAMERIC cannot be specified if SKIP (see above) is specified.

NUMeric

The user can enter only numeric data type characters.

PROTected

The designated map fields are input protected. The user cannot enter, modify, or delete data.

UNPROTected

The designated map fields are not input protected. The user can enter, modify, or delete data.

Note: UNPROTECTED cannot be specified if SKIP (see above) is specified.

DISPlay

The designated map fields are displayed at normal intensity.

DARK

The designated map fields are displayed at darker-than-normal intensity.

Characters in a darkened field do not appear on the terminal screen. DARK cannot be specified if DETECTABLE (see below) is specified.

BRIGHT

The designated map fields are displayed at brighter-than-normal intensity. A brightened field appears highlighted on the terminal screen.

DETECTable

Specifies that the designated map fields are detectable by selector light pen.

Note: DETECTABLE cannot be specified if DARK (see above) is specified.

MDT

Modified data tags (MDTs) are turned on for the designated map fields when the map is displayed.

NOMDT

Modified data tags (MDTs) are not turned on for the designated map fields when the map is displayed.

BLINK

(3279-type terminals only) The designated map fields are displayed with blinking characters.

Note: BLINK cannot be specified if either REVERSE-VIDEO or UNDERSCORE (see below) is specified.

NOBLINK

(3279-type terminals only) Blinking characters are suppressed for the designated map fields.

REverse-video

(3279-type terminals only) The designated map fields are displayed with dark characters on a light background.

Note: REVERSE-VIDEO cannot be specified if either BLINK (see above) or UNDERSCORE (see below) is specified.

NORMal-video

(3279-type terminals only) The designated map fields are displayed with light characters on a dark background.

UNDERscore

(3279-type terminals only) The designated map fields are underscored.

Note: UNDERSCORE cannot be specified if either BLINK or REVERSE-VIDEO (see above) is specified.

NOUNDERscore

(3279-type terminals only) The designated map fields are not underscored.

NOColor

(3279-type terminals only) The designated map fields are displayed with the default color of the terminal.

BLUE/RED/PINK/GREEN/TURquoise/YELLOW/WHITE

(3279-type terminals only) The designated map fields are displayed with one of the seven available color attributes.

Usage*Considerations*

- Multiple attributes to be modified can be specified in a single MODIFY MAP command. All indicated modifications apply to all specified map data fields in the command.

If multiple attributes are specified, they must be separated by commas or blanks.

- The following rules apply to attributes and WCC options that are omitted from a MODIFY MAP command:
 - If an attribute that is not a WCC option is omitted, the attribute remains as defined at map compilation time or as set by a previous modification designated as PERMANENT.
 - If any WCC option is altered by the MODIFY MAP command, *all* WCC options are overridden by the command. Unspecified WCC options are assigned the default values listed in the syntax rules below.
- The ERROR MESSAGE clause of the MODIFY MAP statement allows suppression of a default error message and display of a more appropriate message. For example, the following error message can be displayed for a part-number field in an order entry application:

THE SPECIFIED PART CANNOT BE MAILED

Note: Pageable maps cannot have the error message suppressed on map in.

Example

The following statements are part of a response process that adds a new CUSTOMER record occurrence to the database. The CUST-NUMBER field is required when adding a customer. If the user does not enter a customer number, an error flag is set for the CUST-NUMBER field and the field is made required:

```
IF CUST-NUMBER EQ SPACES
THEN
DO.
  MODIFY MAP TEMPORARY FOR FIELD CUST-NUMBER EDIT ERROR.
  MODIFY MAP PERMANENT FOR FIELD CUST-NUMBER REQUIRED.
  DISPLAY MESSAGE TEXT IS
  'CUSTOMER NUMBER REQUIRED WHEN ADDING CUSTOMER. ' .
END.
ELSE
  MODIFY MAP TEMPORARY FOR FIELD CUST-NUMBER EDIT CORRECT OPTIONAL.
```

Pageable Maps

A pageable map is a map that contains multiple occurrences of a set of map fields. Each occurrence of the multiply-occurring set is called a **detail occurrence**.

A pageable map can contain more detail occurrences than can fit on the user's screen at one time. The runtime system stores detail occurrences in the order in which they are created by pageable map commands, and divides them into pages, based on the number of occurrences that can fit on the screen. One page of occurrences can be displayed on the screen at a time.

An example of a pageable map is one that displays information about a department and lists all the employees within the department. The set of map fields related to employee information occurs once for each employee to be listed. These detail occurrences of employee information are created at run time by pageable map commands and can be displayed to the user one page at a time.

Areas of a Pageable Map

A pageable map is divided into three areas.

Header Area

The **header area** (optional) is located across the top of the screen and contains one or more rows of map fields associated with header information. The header area information is displayed whenever the map is displayed.

Detail Area

The **detail area** (required) is located across the middle of the screen and contains the detail occurrences. Detail occurrence map fields are defined in the detail area only once. At run time, the number of detail occurrences that are displayed in the detail area depends on the space available on the screen after accounting for the header and footer information.

Footer Area

The **footer area** (optional) is located across the bottom of the screen and contains one or more rows of map fields associated with footer information. The footer information is displayed whenever the map is displayed.

For example, a pageable map used to display a department record and all associated employee records might contain the following information:

- **Header area**— The title of the map and department information
- **Footer area**— A message field, the map page, and information about how to page through the map
- **Detail area**— Detail occurrences of employee information

[DEPT. ID: _____		
	EMP. ID: _____	LAST NAME: _____	ACTION CODE: _____
	START DATE: _____	MESSAGE: _____	
[PAGE: _____		

Map Paging Session

A **map page** refers to the header and footer map fields and to a page of detail occurrences.

When a pageable map is displayed, the page of occurrences that appears in the detail area is determined by the current value of the \$PAGE system-supplied data field. For example, given a screen that can hold ten occurrences, if \$PAGE equals 1, occurrences 1 through 10 are displayed; if \$PAGE equals 2, occurrences 11 through 20 are displayed; and so forth. Actions taken by the user and commands issued by premap and response processes can modify the value of \$PAGE.

Beginning a Map Paging Session

A **map paging session** begins when a dialog associated with a pageable map begins execution. A map paging session ends when the application terminates or when a dialog passes control to another dialog under any of the following conditions:

- The dialog receiving control is associated with a different pageable map than the dialog that initiated the map paging session
- The dialog receiving control has different map paging dialog options than the dialog that initiated the map paging session
- The dialog that initiated the map paging session issues a TRANSFER command
- The dialog receiving control is at a level higher than the dialog that initiated the map paging session

Note: The first two conditions do not apply when the receiving dialog is not associated with a pageable map. In such cases, the map paging session continues, provided that the third or fourth condition is not met.

If none of the above conditions is met, the map paging session continues. Detail occurrences created during the session can be added to, displayed, and modified by dialogs associated with the pageable map. If the map paging session terminates, the runtime system deletes all detail occurrences created during the session.

One or more dialogs can be associated with the same pageable map in a given map paging session. During a map paging session, premap and response process commands can create, display, retrieve, and modify detail occurrences.

Considerations

The following considerations apply:

- **Detail occurrences are created** by PUT NEW DETAIL process commands. Detail occurrences are built from the values stored in the variable data fields to which the detail occurrence fields map.

The runtime system stores detail occurrences in the order in which they are created and divides them into pages, based on the number of detail occurrences that can fit on the screen at one time. A detail occurrence is displayed on the screen only when the map page to which the occurrence belongs is displayed.

- **A dialog process displays a map page to the terminal** as a result of either of the following actions:
 - **A PUT NEW DETAIL command is issued** that creates the first detail occurrence of the second map page. The runtime system automatically displays the first map page, allowing the user to enter information.

The process that issues the PUT NEW DETAIL command continues to execute and can create additional detail occurrences. The process must issue a DISPLAY command to terminate processing. The runtime system does not process information entered during a pseudo-converse until the DISPLAY command is issued.

Note: In this case, the DISPLAY command does not send information to the terminal. Header and footer variable data fields should be primed before the first map page is displayed. If the map contains a message field in the header or footer area, any text for the message field should be specified once by issuing a PUT NEW DETAIL command before the first map page is displayed.

- **A DISPLAY command is issued**, except when the map has already been displayed as a result of a PUT NEW DETAIL command. The map page displayed is determined by the current value of \$PAGE.
- **The user can modify map data fields on the screen**, including header and footer data fields and detail occurrence fields of the current map page. Restrictions that apply include those specified in the map definition (such as the PROTECT specification), in the dialog definition (that is, the paging mode dialog option, UPDATE/BROWSE), and by process commands (such as the MODIFY MAP command).
- **The user can make a paging request** to specify the next map to be displayed by performing one of the following actions:
 - Pressing the control key associated with paging forward one page. The system generation default paging-forward key is PF8.
 - Pressing the control key associated with paging backward one page. The system generation default paging-backward key is PF7.
 - Changing the \$PAGE map field (if one is defined for the map) and pressing a control key other than the paging-forward key, paging-backward key, [Clear], [PA1], [PA2], or [PA3]
- **The user presses a control key**, including the paging-forward or paging-backward key, and the runtime system performs the following processing:
 - **Updates map data fields**— The runtime system updates its internal representation of the header and footer map data fields and updates detail occurrence fields to reflect changes made by the user. No updates are performed if [Clear], [PA1], [PA2], or [PA3] are pressed; these control keys do not transmit data.

Map field attributes set temporarily by the user or by map modification commands are reset. Attributes set permanently in the map definition or by map modification commands remain set.

- **Updates \$PAGE**— If a paging request was made, the runtime system updates \$PAGE as follows:
 - Adds 1 to \$PAGE if the paging forward key was pressed and the current map page is not the last map page.
 - Subtracts 1 from \$PAGE if the paging backward key was pressed and the current map page is not the first map page.
 - Moves the value entered in the \$PAGE map field to \$PAGE if the \$PAGE map field was changed and the control key pressed was not the paging forward key, the paging backward key, [Clear], [PA1], [PA2], or [PA3]. If the value entered in the \$PAGE field is less than the first map page or greater than the last map page, \$PAGE is set to the first or last page number.

\$PAGE determines the next map page to be displayed.
- **Determines the flow of control**— In a session that is not a map-paging session, the runtime system always attempts to initiate a function or response process when the user presses a control key. In a map paging session, the runtime system either attempts to initiate a function or response process, or instead displays the same or another map page. The action taken by the runtime system depends on the paging-type dialog option (NOWAIT/WAIT/RETURN), on whether a paging request was made, and whether any map field's modified data tag was set, as shown in the following table.

Flow of Control in a Map Paging Session

Paging Type	Paging request ¹		Nonpaging request	
	No MDT set	Any MDT set ²	No MDT set	Any MDT set ²
NOWAIT	Displays the requested map page	Displays the requested map page	Initiates a function or response process ³	Redisplays the same map page
WAIT	Displays the requested map page	Initiates a function or response process ³	Initiates a function or response process ³	Initiates a function or response process ³
RETURN	Initiates a function or response process ³	Initiates a function or response process ³	Initiates a function or response process ³	Initiates a function or response process ³

Notes:

¹ A paging request occurs when the user presses a control key associated with paging forward or backward or modifies the \$PAGE field, if one is defined for the map. If [Clear], [PA1], [PA2], or [PA3] is pressed, any modification to \$PAGE is ignored and is not considered as a paging request. If a paging request is not made, refer to the Nonpaging request columns.

² If the control key pressed is [Clear], [PA1], [PA2], or [PA3], refer to the No MDT set column under the applicable Paging/Nonpaging request column.

³ The function or response is selected as described in [Runtime flow of control](#) (see page 135).

If the same or another map page is displayed, the user can modify map fields, make a paging request, and press a control key, as described above.

If a function or response process is initiated, the internal representations of the header and footer fields are mapped into their associated variable data fields.

- **Detail occurrences are retrieved** by GET DETAIL process commands. A GET DETAIL command locates the occurrence to be retrieved, then moves the occurrence's fields into the variable data fields to which the fields map.
- **Only modified detail occurrences can be retrieved.** A detail occurrence is considered to be modified if it has the following two characteristics:
 - Contains one or more map fields whose modified data tags (MDTs) are set at the time of the most recent pseudo-converse.
 - Has yet to be retrieved since the most recent pseudo-converse. Once a modified detail occurrence has been retrieved, it is no longer considered to be modified.

Note that if a modified detail occurrence is not retrieved following a pseudo-converse, it is not automatically considered to be modified following a subsequent pseudo-converse. The detail occurrence must once again have the two characteristics listed above.

A detail occurrence that is not a modified detail occurrence cannot be retrieved by dialog process code.

- **Detail occurrence fields are modified in dialog processes** by PUT CURRENT DETAIL commands. A PUT CURRENT DETAIL command modifies the detail occurrence referenced by the most recent GET DETAIL or PUT DETAIL command.
- **Additional detail occurrences can be created** by PUT NEW DETAIL commands. New occurrences are stored at the end of the set of detail occurrences.

- **Detail occurrences cannot be deleted by process commands.** Detail occurrences are deleted as follows:
 - If the backpage dialog option (described below) is NO, detail occurrences of previous map pages are deleted when a new map page is displayed.
 - At the end of a map paging session, all detail occurrences are deleted.

More information:

[Variable Data Fields](#) (see page 285)

Map Paging Dialog Options

Map paging dialog options define parameters for a map paging session. Specification of options for a dialog are made during dialog definition. The map paging dialog options NOWAIT, BACKPAGE NO, and UPDATE cannot be specified together.

The following table lists available map paging dialog options.

Map Paging Dialog Options

Option	Parameter	Description
Paging type	NOWAIT WAIT RETURN	Specifies the runtime flow of control when the user presses a control key, as described in the previous table.
Backpage	BACKPAGE YES	Allows the user to display a previous map page. The runtime system maintains the resources that describe the detail occurrences of previous pages.
	BACKPAGE NO	Prohibits the user from displaying a previous map page. The runtime system deletes all previous pages of detail occurrences when a new map page is displayed. The lowest page number is the first page that has not been deleted. ¹
Paging mode	UPDATE	Specifies that the terminal operator can modify map data fields, subject to restrictions specified in the mapping facility and by map modification process commands.

Option	Parameter	Description
	BROWSE	Specifies that the user can modify only the \$PAGE and \$RESPONSE fields of the map. Map fields can still have their MDTs set in the map definition or by map modification commands.

Note:

¹ On mapin from the terminal when backpageing is not allowed, if \$PAGE has been set to a value greater than the current map page, the runtime system flags all map pages below \$PAGE for deletion. When the map is displayed again, these flagged pages are deleted, even if \$PAGE has been modified to a lower value in the interim.

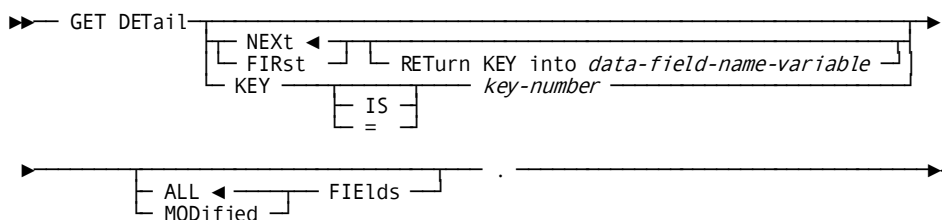
GET DETAIL

Purpose

Retrieves a modified detail occurrence of a pageable map.

A GET DETAIL command can retrieve all the fields of a modified detail occurrence or only those fields whose MDTs are turned on.

Syntax



Parameters

NEXT

Retrieves the first modified detail occurrence that follows the detail occurrence referenced by the preceding pageable map command.

The preceding pageable map command must follow the most recent pseudo-converse.

FIRst

Retrieves the first modified detail occurrence of the pageable map.

Note that the GET DETAIL FIRST command can be used repeatedly to retrieve all the modified detail occurrences of a pageable map. The first GET DETAIL FIRST command retrieves the first modified detail occurrence. Once retrieved, the occurrence is no longer considered as modified, and the second modified detail occurrence becomes the first modified detail occurrence. This modified detail occurrence can be retrieved by a subsequent GET DETAIL FIRST command, and so forth.

An end-of-data condition results if no pageable map command precedes the GET DETAIL command, if the preceding pageable map command resulted in an end-of-data or detail-not-found (see the KEY IS parameter below) condition or if the GET DETAIL command cannot find a modified detail occurrence before reaching the end of the set of detail occurrences.

RETurn KEY into *data-field-name-variable*

Specifies the numeric variable field into which the runtime system moves the binary fullword value (if any) associated with the detail occurrence being retrieved. A value is associated with a detail occurrence by specifying the KEY IS parameter in a PUT DETAIL command.

If no value is associated with the detail occurrence, *data-field-name-variable* is set to zero. *Data-field-name-variable* does not have to be a binary fullword.

KEY is *key-number*

Specifies the modified detail occurrence to be retrieved based on the numeric key value associated with the detail occurrence. A key value is associated with a detail occurrence by specifying the KEY IS parameter in a PUT DETAIL command.

Key-number is either the numeric variable data field or the numeric literal itself.

The runtime system finds the first detail occurrence associated with the key value specified by *key-number*. If the detail occurrence is a modified detail occurrence, it is retrieved. If the occurrence is not a modified detail occurrence, or if no detail occurrence with the specified key value is found, a detail-not-found condition is set.

ALL

Specifies all the fields of the modified detail occurrence to be retrieved.

ALL is the default when neither ALL or MODIFIED is specified.

MODified

Specifies only those fields whose MDTs are turned on to be retrieved.

If MODIFIED is specified, variable data fields that map to nonretrieved fields retain their previous values.

Usage

Considerations:

- The GET DETAIL command causes the runtime system to move the following:
 - The retrieved fields into the variable data fields to which they map
 - The page number of the retrieved detail occurrence into the \$PAGE system-supplied data field
 - The numeric key value (if any) associated with the occurrence into a specified field (optional)
- A GET DETAIL command can retrieve only a modified detail occurrence. A detail occurrence is considered modified if it has the following characteristics:
 - Contains one or more map fields whose modified data tags (MDTs) are turned on at the time of the most recent pseudo-converse.
 - Has yet to be retrieved. Once a modified detail occurrence has been retrieved, it is no longer considered modified.
- A detail occurrence that is not a modified detail occurrence cannot be retrieved by dialog process code.

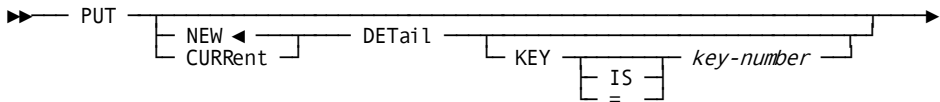
PUT DETAIL

Purpose

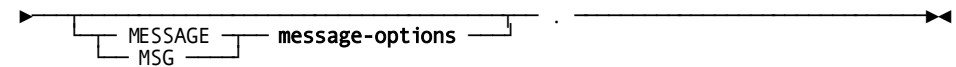
The PUT DETAIL command:

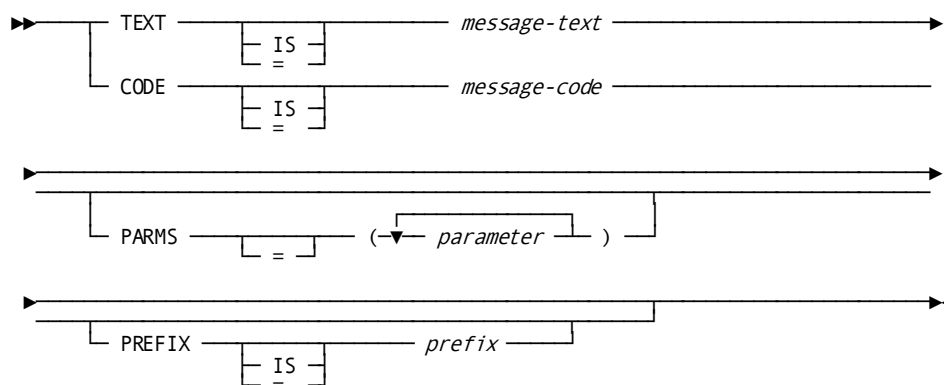
- creates or modifies a detail occurrence of a pageable map
- specifies a numeric value to be associated with the occurrence
- specifies a message to appear in the message field of the occurrence.

Syntax



Expansion of Message-Options





Parameters

NEW DETail

Creates a detail occurrence which is stored at the end of the set of detail occurrences.

NEW is the default when neither NEW or CURRENT is specified.

CURRent DETail

Modifies the detail occurrence referenced by the most recent pageable map command.

After a pseudo-converse, a pageable map command must be issued to establish currency on a detail occurrence before a PUT CURRENT DETAIL command can be issued. If currency is not established, CA ADS abnormally terminates the dialog.

KEY is *key-number*

Specifies the numeric value to be associated with the detail occurrence being created or modified.

Key-number is either the name of a variable data field or the number itself, expressed as a numeric constant.

Key-number replaces the numeric value (if any) previously associated with the detail occurrence. The numeric value is not displayed at the terminal, but is stored along with the detail occurrence as a binary full word.

The KEY parameter can be used to store the database key of a subschema record associated with a detail occurrence. A GET DETAIL command can later retrieve the database key when it retrieves the detail occurrence, facilitating the retrieval of the subschema record.

MESSage

Introduces the text or code of a message.

MSG can be used in place of MESSAGE.

message-options

Identifies message to be displayed.

Expanded syntax for *message-options* is shown above immediately following the PUT DETAIL syntax.

TEXT is *message-text*

Specifies the text of a message to be displayed in an online map's message field or sent to a batch application and a system log file.

Message-text specifies either the name of a variable data field containing the message text or the text string itself, enclosed in single quotation marks.

The text string can contain up to 240 displayable characters.

CODE is *message-code*

Specifies the message dictionary code of a message to be displayed in an online map's message field or sent to the log file in a batch application.

In a batch application, the message is also sent to the operator, if directed by the destination specified in the dictionary.

Message-code specifies either the name of a variable data field that contains the message code or the 6-digit code itself, expressed as a numeric literal.

PARMS = *parameter*

Specifies a replacement parameter for each variable field in the stored message identified by *message-code*.

Up to nine replacement parameters can be specified for a message. Multiple parameters must be specified in the order in which they are numbered and separated by blanks or commas.

Parameter specifies either the name of an EBCDIC or unsigned zoned decimal variable data field that contains the parameter value or the parameter value itself, enclosed in single quotation marks.

The parameter value must contain displayable characters. At run time, each variable data field in a stored message expands or contracts to accommodate the size of its replacement parameter. A replacement parameter can be a maximum of 240 bytes.

PREFIX is *prefix*

Overrides the default prefix of a dialog and a map.

Prefix specifies an EBCDIC or unsigned zoned decimal variable data field that contains a 2-character prefix or the 2-character prefix itself, enclosed in single quotation marks.

Creating or Modifying a Detail Occurrence of a Pageable Map

After a PUT DETAIL command is executed, the map fields of a created or modified occurrence contain the values of the variable data fields to which they map. The created or modified occurrence appears on the user's screen when the map page to which it belongs is displayed.

Storage

The amount of storage available at run time to hold detail occurrences is specified at system generation with the PAGING STORAGE clause of the OLM statement. By default, the available storage is 10K bytes. If a PUT DETAIL command would cause storage overflow, the detail occurrence is not created and the \$MAXIMUM-DETAILS-PUT map paging condition is set. The \$MAXIMUM-DETAILS-PUT condition can be tested.

Note: For more information about calculating the storage required by a pageable map, see the *CA IDMS Mapping Facility Guide*.

More information:

[Conditional Expressions](#) (see page 245)

Specifying a Numeric Value Associated with an Occurrence

A numeric value, such as a database key, can be associated with a created or modified detail occurrence.

This value is not displayed to the user, but can be retrieved by a GET DETAIL command.

Specifying a Message to Appear in the Message Field of an Occurrence

The text of a message or a code associated with a message that has already been defined in the message dictionary can be specified in a PUT DETAIL command. When the dialog is executed, the runtime system moves the appropriate message to the message field in the dialog's map.

A message field is defined by the \$MESSAGE map field.

Note: For more information, see the *CA IDMS Mapping Facility Guide*.

If no message field is defined for the detail area of the pageable map, the runtime system places the message in the header or footer message field or, if neither the header nor the footer has a message field, the runtime system ignores the message. If more than one message is placed in the header or footer message field, the messages are concatenated up to the length of the message field.

Considerations

The following considerations apply to specifying a message code in a PUT DETAIL command:

- Each system-supplied message in the data dictionary message area (DDLDCMSG) is identified by a six-digit code prefixed by the letters DC. For example, a request for message 987654 retrieves message DC987654.
User-defined messages added to the message dictionary can have a prefix other than DC and digits in the range 900001 through 999999.
- Each message in the message dictionary can be assigned a severity code. The severity code specifies the action CA ADS takes when a message is retrieved. The following table lists the severity codes and their associated actions.

Severity code	Action
0	Processes the PUT DETAIL command
1	Snaps all CAADS resources and processes the PUT DETAIL command
2	Snaps all system areas and processes the PUT DETAIL command
3	Snaps all CAADS resources and terminates CA ADS with a task abend code of D002
4	Snaps all system areas and terminates CA ADS with a task abend code of D002
5	Terminates CA ADS with a task abend code of D002
8	Snaps all system areas and terminates the DC system with an operating system abend code of 3996
9	Terminates the DC/UCF system with an operating system abend code of 3996

- A message in the message dictionary can contain one or more variable fields that are replaced with application-specific values at run time. In a PUT DETAIL command, the application developer can use the PARMs parameter to code replacement parameters for each variable field in a specified message.

Within the message definition in the dictionary, symbolic parameters are identified by an ampersand (&) followed by a two-digit numeric identifier. These identifiers can appear in any order. The position of the replacement values in the PARMs parameter must correspond directly to the two-digit numeric identifiers in the message; the first value corresponds to &01, the second to &02, and so forth. For example, assume that the stored message text is as follows:

```
THIS IS TEXT &01 AND &03 OR &02
```

The PARMs parameter reads PARMs=('A','B','C'). The resulting text would read as follows:

```
THIS IS TEXT A AND C OR B
```

- If the message is defined in the dictionary with more than one text line, only the first line appears in the map's message field.

Example

The following example illustrates the map and the premap and response processes of a dialog that:

- Lists the employees in a department one page at a time
- Allows the user to modify employee information and delete employees
- Updates the database based on the user's entries
- Redisplays the map with appropriate messages and allows the user to make further modifications

The **paging type** in this example is WAIT. If the user makes a paging request and no MDTs are set, the runtime system displays the requested page. If the user makes a nonpaging request or if any MDTs are set, the runtime system initiates the response process. The response process is associated with the control keys ENTER, FWD (paging forward), and BWD (paging backward).

The **pageable map** associated with the dialog is shown in the screen that follows. The following considerations apply to the detail area map fields:

- The fields are defined once. At run time, the number of occurrences of these fields that are displayed to the user at any one time depends on the number of occurrences that can fit on the screen between the header and footer areas.

- At run time, the fields map to variable data fields through detail occurrences:
 - PUT NEW DETAIL commands create detail occurrences from associated variable data fields.
 - When a map page is displayed, the detail occurrences for the page are displayed.
 - When the user presses a control key, the appropriate detail occurrence fields are updated.
 - GET DETAIL commands can retrieve modified detail occurrences into associated variable data fields.
- The fields map to work record data fields and not directly to EMPLOYEE database record elements. This facilitates the update of EMPLOYEE database records in the response process.

DEPT. ID: _____ ¹			
EMP. ID: _____ ²	LAST NAME: _____ ³	ACTION CODE: _____ ⁴	
	START DATE: _____ ⁵	MESSAGE: _____ ⁶	
PAGE: _____ ⁷			
8			

Note:

1. Maps to DEPT-ID of DEPARTMENT database record
2. Maps to WK-EMP-ID through detail occurrence
3. Maps to WK-EMP-LNAME through detail occurrence
4. Maps to WK-ACTION through detail occurrence
5. Maps to WK-EMP-START-DATE through detail occurrence
6. Maps to \$MESSAGE through detail occurrence
7. Maps to \$PAGE system-supplied data field
8. Maps to WK-MESSAGE

Sample Premap Process

The **premap process** shown below performs the following:

- Obtains a DEPARTMENT record based on a CALC key passed from another dialog or function
- Obtains all associated EMPLOYEE records
- Creates a detail occurrence for each retrieved record
- Displays the first map page at the terminal

```
OBTAIN CALC DEPARTMENT.  
IF DB-REC-NOT-FOUND  
  THEN  
    DO.  
      MOVE 'DEPARTMENT NOT FOUND' TO WK-MESSAGE.  
      DISPLAY.  
    END.  
MOVE SPACES TO WK-ACTION.  
OBTAIN FIRST EMPLOYEE WITHIN DEPT-EMPLOYEE.  
WHILE 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.  
IF $PAGE-READY  
  THEN  
    DO.  
      MOVE 'MORE EMPLOYEES EXIST FOR THIS DEPT' TO WK-MESSAGE.  
      DISPLAY.  
    END.  
OBTAIN NEXT EMPLOYEE WITHIN DEPT-EMPLOYEE.  
END.  
MOVE 'ALL EMPLOYEES DISPLAYED FOR THIS DEPT' TO WK-MESSAGE.  
DISPLAY.
```

Sample Response Process

The **response process** shown below performs the following:

- Retrieves each modified detail occurrence.
- Updates the EMPLOYEE database accordingly.
- Modifies each retrieved detail occurrence:
 - Moves a confirming message to the message field
 - Initializes the action code
 - Protects the fields if the associated database record is deleted
- Redisplay the map. The value of \$PAGE is saved at the beginning of the response process and is restored at the end in order to display the page requested by the user. During the response process, \$PAGE is modified by GET DETAIL commands.

```
READY USAGE-MODE UPDATE.  
MOVE $PAGE TO WK-PAGE.  
GET DETAIL FIRST RETURN KEY WK-KEY.  
WHILE NOT $END-OF-DATA  
  REPEAT.  
    OBTAIN EMPLOYEE DB-KEY IS WK-KEY.  
    IF WK-ACTION EQ 'DEL'  
      THEN  
        DO.  
          ERASE EMPLOYEE.  
          PROTECT (WK-EMP-ID WK-EMP-LNAME  
                  WK-EMP-START-DATE WK-ACTION) PERMANENT.  
          MOVE SPACES TO WK-ACTION.  
          PUT CURRENT DETAIL TEXT 'DELETED'.  
        END.  
      ELSE  
        DO.  
          MOVE WK-EMP-LNAME TO EMP-LNAME.  
          MOVE WK-EMP-START-DATE TO EMP-START-DATE.  
          MODIFY EMPLOYEE.  
          MOVE SPACES TO WK-ACTION.  
          PUT CURRENT DETAIL TEXT 'MODIFIED'.  
        END.  
      GET DETAIL NEXT RETURN KEY WK-KEY.  
    END.  
  MOVE WK-PAGE TO $PAGE.  
  DISPLAY.
```

Chapter 18: Queue and Scratch Management Commands

This section contains the following topics:

[Overview](#) (see page 483)
[Queue Records](#) (see page 485)
[DELETE QUEUE](#) (see page 486)
[GET QUEUE](#) (see page 488)
[PUT QUEUE](#) (see page 491)
[Scratch Records](#) (see page 494)
[DELETE SCRATCH](#) (see page 496)
[GET SCRATCH](#) (see page 498)
[PUT SCRATCH](#) (see page 502)

Overview

CA ADS queue and scratch management commands are used to control the allocation and access of queue and scratch records. Queue and scratch records are work records stored in the data dictionary that allow data to be passed from one CA IDMS/DC or DC/UCF (DC/UCF) task to another.

Note: During the execution of an CA ADS application, each pseudo-converse is a new task.

Queue Records

Queue records are stored in the data dictionary queue area (DDLDCRUN). Use of queue records allows data to be passed from one DC/UCF task or batch application to another.

Scratch Records

Scratch records are temporarily maintained in the data dictionary scratch area (DDLDCSCR). Under CA ADS Batch, scratch records can be stored in and retrieved from a scratch file allocated by the site. Use of scratch records allows data to be passed between tasks or dialogs.

Queue and Scratch Management Commands

Queue and scratch management commands are summarized in the following table. Each command is discussed later in this section.

Type	Command	Description
Queue management	DELETE QUEUE	Deletes one or all queue records in a specified queue.
	GET QUEUE	Transfers the contents of a queue record to a specified location in a dialog's record buffers and, optionally, deletes the record from the queue.
	PUT QUEUE	Stores a queue record in the data dictionary and assigns a queue id.
Scratch management	DELETE SCRATCH	Deletes one or all scratch records associated with a specified scratch area. In CA ADS Batch, one or all scratch records associated with a specified scratch file are deleted.
	GET SCRATCH	Transfers the contents of a scratch record to a specified location in a dialog's record buffers and, optionally, deletes the record. In CA ADS Batch, the contents of a scratch record are transferred to a specified location and a scratch file is assigned to the record.
	PUT SCRATCH	Stores or replaces a scratch record in the data dictionary and assigns a scratch area id. In CA ADS Batch, a scratch record is stored or replaced in the scratch file and assigned a scratch area id.

More information:

[CA ADS Runtime System](#) (see page 119)

Queue Records

Overview

Queue records are available to all tasks running under DC/UCF, as well as to batch programs. Records in a queue established by one task are available to subsequent tasks running on the same logical terminal, or to concurrent or subsequent tasks running on any other terminal. Queue records are saved across system shutdowns and are recovered across a system crash.

Because queue records are available to concurrent tasks running on other terminals, the records can be used to pass data from one application to another. Additionally, queue records provide a convenient means of storing data for subsequent processing.

Storing a Queue Record

A queue record is stored in the data dictionary as a member occurrence in a set owned by a queue header record. All records associated with a particular queue header are referred to collectively as a queue. The queue is identified by a queue id. Requests to access a queue record can use the queue id to specify the queue in which the object record participates. If a request to store a queue record specifies an unknown queue id, a queue is created with the specified id.

When a queue record is stored, DC/UCF can return a queue record identifier to a specified location in a dialog's record buffers. The queue record identifier can then be used to access the queue record.

Currencies

The CA ADS runtime system maintains currencies for each queue accessed by a task. If concurrently executing tasks access the same queue, each task has its own queue currency. A request for a particular queue record can identify the record by the queue id, by the queue record id, by the position of the record within the queue, or by the relationship of the object record to the record that is current of queue for the requesting task.

Queue records remain in the data dictionary until explicitly deleted or until the retention period specified for the queue has expired. When all records associated with a given queue header have been deleted, the header record is also deleted and the queue no longer exists.

Considerations

- An exclusive lock is placed on a queue record when the record is retrieved or stored, thereby preventing concurrently executing tasks from accessing the same record. Queue record locks are released when the task terminates or when a COMMIT command with the TASK keyword is executed.

Because no other task can access a locked queue record, a concurrently executing task that attempts to access the record must wait until the lock is released. To minimize such waits, queue access should be as brief as possible.
- Queue currencies are not saved when a task terminates. Each task must establish its own currencies. The following considerations apply:
 - Queue currencies are lost each time a DISPLAY command is executed.
 - Queue currencies are lost across a system shutdown or a system crash.
- All queue management command clauses must be coded in the same order in which they appear in the syntax.
- Queue management commands are allowed in CA ADS Batch only if the application is running under the central version.

More information:

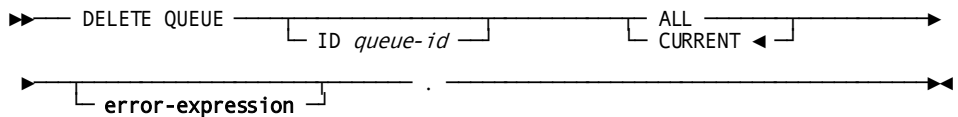
[Database Access Commands](#) (see page 363)

DELETE QUEUE

Purpose

Deletes a queue or queue record.

Syntax



Parameters

ID *queue-id*

Specifies the queue or queue record associated with *queue-id* to be deleted.

Queue-id is the name of a variable data field that contains a queue id or the 1- to 16-character id itself, enclosed in single quotation marks.

If *queue-id* is not specified, a null queue id (that is, 16 blanks) is assumed.

ALL

Deletes all records, including the queue header record, in the queue specified by *queue-id*.

CURRENT

Deletes the record that is current of queue for the requesting task.

CURRENT is the default when you specify neither CURRENT or ALL.

error-expression

Specifies the status codes that are returned to the dialog.

Usage*Considerations*

If autostatus is not in use, a dialog's error-status field indicates the outcome of a DELETE QUEUE command:

Status Code	Meaning
0000	The request was executed successfully
4404	The requested header record cannot be found
4405	The requested queue record cannot be found
4406	Currency was not established for the object queue record
4407	An I/O error occurred during processing
4431	The CA ADS internal parameter list was invalid

Example

The following example illustrates the use of the DELETE QUEUE command to delete the current record from queue CUSTQ:

```
DELETE QUEUE ID 'CUSTQ'.
```

More information:

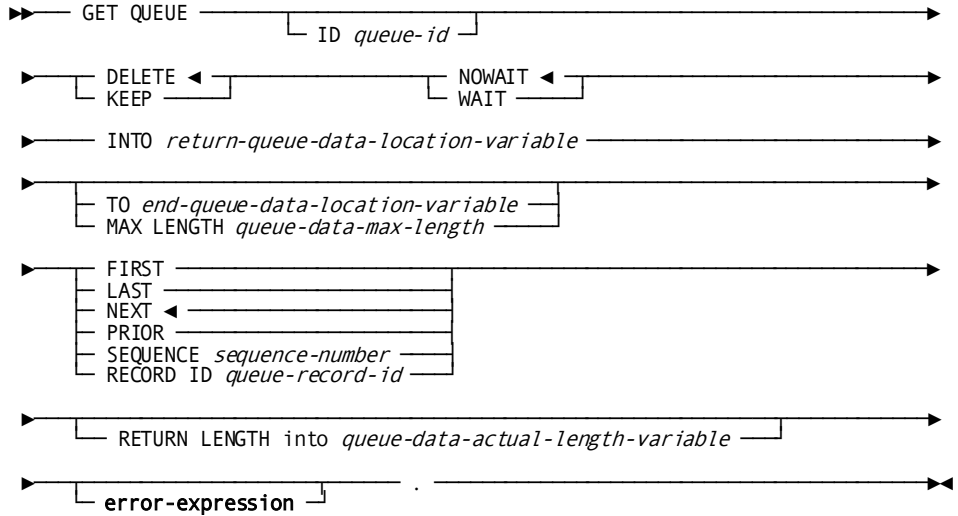
[Error Handling](#) (see page 277)

GET QUEUE

Purpose

Transfers the contents of a queue record to a specified location in a dialog's record buffers.

Syntax



Parameters

ID *queue-id*

Specifies the *queue-id* to be retrieved.

Queue-id is either the name of a variable data field that contains a queue id or the 1- to 16-character queue id itself, enclosed in single quotation marks.

If *queue-id* is not specified, a null queue id (that is, 16 blanks) is assumed.

DELETE

Deletes the record from the queue after it is passed to the requesting task. If the record is truncated, the truncated data may be lost permanently.

DELETE is the default when you specify neither DELETE or KEEP.

KEEP

Retains the record in the queue after it is passed to the requesting task.

NOWAIT

Continues task execution in the event of a nonexistent queue. NOWAIT is the default when you specify neither NOWAIT or WAIT.

WAIT

Suspends task execution until the requested queue exists.

INTO *return-queue-data-location-variable*

Specifies the location to which the requested queue record is transferred.

Return-queue-data-location-variable is the name of a variable data field in the dialog's record buffers.

TO *end-queue-data-location-variable*

Specifies the end of the buffer area allocated for the requested queue record.

End-queue-location-variable is either the name of a dummy byte field or the name of a variable data field that contains a data item not associated with the requested queue record. The field specified by *end-queue-data-location* must immediately follow the last byte of the buffer area allocated for the requested queue record.

MAX LENGTH *queue-data-max-length*

Specifies the length of the buffer area allocated for the requested queue record.

Queue-data-max-length is either the name of a variable data field that contains the length of the buffer area allocated for the requested queue record or the length itself, expressed as a numeric constant.

If neither *TO end-queue-data-location-variable* nor **MAX LENGTH *queue-data-max-length*** is specified, the length of the location is the length of *return-queue-data-location-variable*.

FIRST

Obtains the first record in the queue that is specified by *queue-id*.

LAST

Obtains the last record in the queue that is specified by *queue-id*.

NEXT

Obtains the record that follows the current record of the queue specified by *queue-id*.

NEXT is the default when you specify no other queue record to be obtained.

If currency is not established, NEXT is equivalent to FIRST.

PRIOR

Obtains the record that precedes the current record in the queue specified by *queue-id*.

If currency is not established, PRIOR is equivalent to LAST.

SEQUENCE *sequence-number*

Obtains the *n*th record in the queue specified by *queue-id*.

Sequence-number is either the name of a variable data field that contains the sequence number or the sequence number itself, expressed as a numeric constant.

RECORD ID *queue-record-id*

Obtains the record identified by *queue-record-id*.

Queue-record-id is either the name of a numeric variable data field that contains the system-assigned queue record id or the queue record id itself, expressed as a numeric constant.

Queue-record-id cannot be a doubleword binary field. The runtime system converts the queue record id to a binary fullword for internal storage.

RETURN LENGTH into *queue-data-actual-length-variable*

Returns the untruncated length of the obtained queue record to the location specified by *queue-data-actual-length-variable*.

Queue-data-actual-length-variable is the name of a numeric field in the dialog's record buffers.

error-expression

Specifies the status codes that are returned to the dialog.

Usage*Considerations*

- If the queue record is larger than the allocated buffer area, the record is truncated as necessary. Deletion of the record from the queue after the transfer is complete can be specified.
- If autostatus is not in use, a dialog's error-status field indicates the outcome of a GET QUEUE command:

Status Code	Meaning
0000	The request was executed successfully
4404	The requested header record cannot be found
4405	The requested queue record cannot be found
4407	An I/O error occurred during processing
4419	The dialog's storage location is too small for the requested queue record. The record was truncated accordingly

Status Code	Meaning
4431	The CA ADS internal parameter list was invalid. In CA ADS, this is usually due to using a RECORD ID parameter that includes a queue-record-id that contains all zeros.
4432	The derived length of the queue record data area is negative

Example

The following example illustrates the use of the GET QUEUE command. The data in the last record in queue CUSTQ is transferred to the location in the dialog's record buffers identified by CUSTWORK. The record is deleted from the queue:

```
GET QUEUE ID 'CUSTQ' INTO CUSTWORK MAX LENGTH REC-LENGTH LAST.
```

More information:

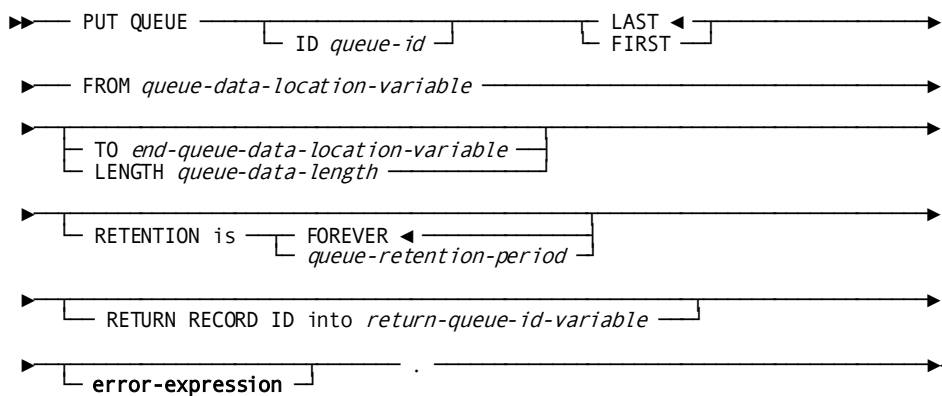
[Error Handling](#) (see page 277)

PUT QUEUE

Purpose

Stores a queue record in the data dictionary.

Syntax



Parameters

ID *queue-id*

Stores a record in the queue identified by *queue-id*.

Queue-id is either the name of a variable data field that contains a queue id or the 1- to 16-character queue id itself, enclosed in single quotation marks.

If *queue-id* is not specified, a null queue id (that is, 16 blanks) is assumed.

LAST

Stores a record at the end of the queue.

LAST is the default when you specify neither LAST or FIRST.

FIRST

Stores a record at the beginning of the queue.

FROM *queue-data-location-variable*

Specifies the location of the data to be stored in the queue record.

Queue-data-location-variable is the name of a variable data field in the dialog's record buffers.

TO *end-queue-data-location-variable*

Specifies the end of the buffer area that contains the queue record data.

End-queue-data-location-variable is the name of a variable data field that contains a data item not associated with the queue record data.

The field specified by *end-queue-data-location-variable* must immediately follow the last byte of the buffer area that contains the queue record data.

LENGTH *queue-data-length*

Specifies the length, to be specified in bytes, of the buffer area that contains the data to be stored in the queue record.

Queue-data-length is either the name of a variable data field that contains the length or the length itself, expressed as a numeric constant.

If neither TO *end-queue-data-location-variable* nor LENGTH *queue-data-length* is specified, the length of the location is the length of *queue-data-location-variable*.

RETENTION

Introduces the number of days, in the range 0 through 255, that the queue is to be retained.

A retention period of 255 is equivalent to FOREVER.

FOREVER

Retains the queue until all queue records associated with the queue are explicitly deleted.

FOREVER is the default when the queue's retention period is not otherwise specified.

queue-retention-period

The name of a variable data field that contains the retention period or the retention period itself, expressed as a numeric constant.

RETURN RECORD ID into *return-queue-id-variable*

Returns a system-assigned queue record id to the location specified by *return-queue-id-variable*.

The queue record id is returned as a binary fullword and is converted, as appropriate, when it is moved to *return-queue-id-variable*

Return-queue-id-variable is the name of a numeric variable data field in the dialog's record buffers.

Return-queue-id-variable cannot be a doubleword binary field. The system-assigned queue record id can subsequently be used to retrieve or delete the associated queue record.

error-expression

Specifies the status codes that are returned to the dialog.

Usage*Considerations*

If autostatus is not in use, a dialog's error-status field indicates the outcome of a PUT QUEUE command:

Status Code	Meaning
0000	The request was executed successfully
4407	The queue upper limit has been reached or an I/O error occurred during processing
4431	The CA ADS internal parameter list was invalid

Example

The following example illustrates the use of the PUT QUEUE command to store the data in CUSTWORK in a queue record associated with queue CUSTQ:

```
PUT QUEUE ID 'CUSTQ' FROM CUSTWORK LENGTH REC-LENGTH  
RETURN RECORD ID INTO REC-ID.
```

More information:

[Error Handling](#) (see page 277)

Scratch Records

Scratch records allow a task to pass information to subsequent tasks, thereby providing data continuity among tasks. The scratch records are used only for temporary storage of data and are not saved across a system shutdown or a system crash.

Scratch Area ID

A scratch area is identified by an eight-character name. Requests to access a scratch record can use the scratch area id to specify the area with which the object record is associated. If a request to store a scratch record specifies an unknown scratch area id, a scratch area is created with the specified id.

Scratch records are also assigned numeric identifiers either by the application developer or by the system. Records in a scratch area are arranged sequentially in ascending order, according to the value of the scratch record identification. System-assigned identifiers are sequenced last in a scratch area.

All scratch management command clauses must be coded in the same order in which they appear in the syntax.

CA ADS Usage

Scratch records are common to all tasks running on the same logical terminal. The records stored by one task are available to subsequent tasks running on the same terminal.

A request to store a scratch record places a record of the requested length in the data dictionary. A database key pointer to the scratch record is placed in a scratch area associated with the requesting task. Scratch records remain in the data dictionary until explicitly deleted, until a signoff from DC/UCF occurs, or until the system is shut down or crashes.

Currencies are maintained for each scratch area associated with a task. Scratch area currencies are passed from one task to the next. A request for a particular scratch record can identify the record by the scratch area id, by the scratch record id, by the position of the record within the scratch area, or by the relationship of the object record to the record that is current of the scratch area.

Considerations

- Scratch records associated with one terminal are not available to tasks associated with other terminals.
- Any number of scratch records can be associated with a single scratch area, and any number of scratch areas can be associated with a task.
- When all records associated with a given scratch area have been deleted, the scratch area is also deleted.
- During the execution of an CA ADS application, each pseudo-converse is a new task.

More information:

[CA ADS Runtime System](#) (see page 119)

CA ADS Batch Considerations

Information can be written to temporary scratch files at dialog execution time and passed between dialogs within the same job step in a given CA ADS Batch application. A request to store a scratch record places a record of the requested length in a temporary work file. Records can be accessed in any order from this file. The scratch file need not be defined to the data dictionary.

Scratch records remain in the temporary file until they are explicitly deleted, until the job step is completed, until a signoff occurs, or if the system is shut down or crashes.

Using Scratch Files

To use scratch files:

1. **Include process-language SCRATCH statements in dialog process modules.** At dialog execution time, these statements store, retrieve, and delete scratch records.
Syntax for SCRATCH statements in CA ADS Batch dialogs is the same as for CA ADS dialogs. SCRATCH statement syntax is presented later in this section.
2. **Define the external name for a scratch file** in the DMCL module for scratch (SCRDMCL).
3. **Initialize a data set for the scratch file the first time the file is used** by using the FORMAT utility.

4. **Include FORMAT job control language statements** immediately before control statements for the CA ADS Batch application.
5. **Specify the ddname/filename for the scratch file** in the CA ADS Batch job to make the initialized scratch file available to the application.

Note: For more information about the FORMAT utility and its input parameters, see the *CA IDMS Utilities Guide*.

Considerations

- The scratch file cannot be used to communicate between CA ADS Batch job steps.
- The same scratch file can be used in several CA ADS Batch job steps without reinitializing the file. A PUT SCRATCH command must be used before any GET SCRATCH commands.
- Central version must be used to run CA ADS Batch.

More information:

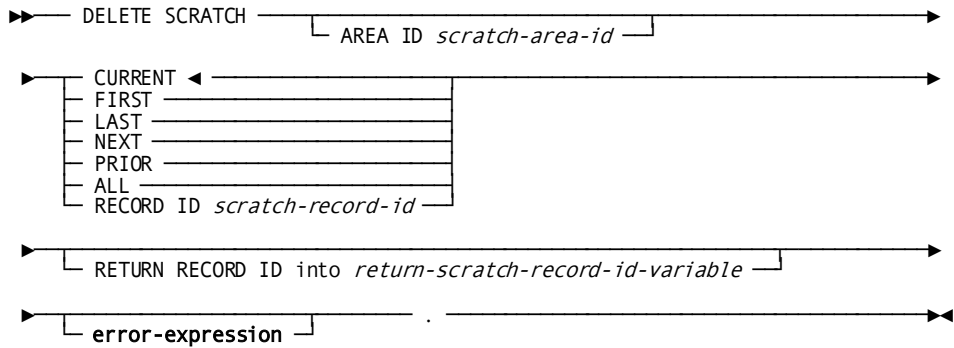
[Application and Dialog Utilities](#) (see page 621)

DELETE SCRATCH

Purpose

Deletes one or all records associated with a particular scratch area id.

Syntax



Parameters

AREA ID *scratch-area-id*

Specifies the area in the data dictionary scratch area to be deleted.

Scratch-area-id is either the name of a variable data field that contains a scratch area id or the 1- to 8-character scratch area id itself, enclosed in single quotation marks.

If *scratch-area-id* is not specified, a null scratch area id (that is, eight blanks) is assumed.

CURRENT

Deletes the record that is current of the scratch area specified by *scratch-area-id*.

CURRENT is the default when you specify no other scratch record to be deleted.

FIRST

Deletes the first record in the scratch area specified by *scratch-area-id*.

LAST

Deletes the last record in the scratch area specified by *scratch-area-id*.

NEXT

Deletes the record that follows the current record of the scratch area specified by *scratch-area-id*.

If currency is not established, NEXT is equivalent to FIRST.

PRIOR

Deletes the record that precedes the current record of the scratch area specified by *scratch-area-id*.

If currency is not established, PRIOR is equivalent to LAST.

ALL

Deletes all records in the scratch area specified by *scratch-area-id*.

RECORD ID *scratch-record-id*

Deletes the record identified by *scratch-record-id*.

Scratch-record-id is either the name of a variable data field that contains the scratch record id or the scratch record id itself, expressed as a numeric constant.

RETURN RECORD ID into *return-scratch-record-id-variable*

Returns the id of the last scratch record deleted to the location specified by *return-scratch-record-id-variable*.

Return-scratch-record-id-variable is the name of a numeric variable data field in the dialog's record buffers.

Return-scratch-record-id-variable cannot be a doubleword binary field.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Considerations If autostatus is not in use, a dialog's error-status field indicates the outcome of a DELETE SCRATCH command:

Status Code	Meaning
0000	The request was executed successfully.
4303	The requested scratch area cannot be found.
4305	The requested scratch record cannot be found.
4307	An I/O error occurred during processing.
4331	The CA ADS internal parameter list is invalid.

Example

The following example illustrates the use of the DELETE SCRATCH command to delete all of the records in scratch area CUSTAREA:

```
DELETE SCRATCH AREA ID 'CUSTAREA' ALL.
```

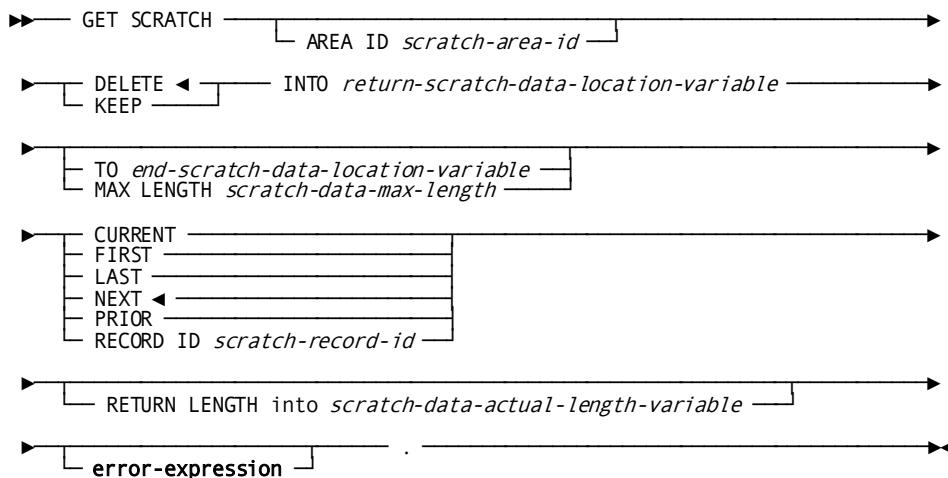
More information:

[Error Handling](#) (see page 277)

GET SCRATCH

Purpose

Transfers the contents of a scratch record to a specified location in a dialog's record buffers.

Syntax**Parameters****AREA ID *scratch-area-id***

Specifies an area in the scratch area to be retrieved.

Scratch-area-id is either the name of a variable data field that contains a scratch area id or the 1- to 8-character scratch area id itself, enclosed in single quotation marks.

If *scratch-area-id* is not specified, a null scratch area id (that is, eight blanks) is assumed.

DELETE

Deletes the record from the scratch area after it is passed to the requesting task.

If the record is truncated, the truncated data may be lost permanently.

DELETE is the default when you specify neither DELETE or KEEP.

KEEP

Retains the record in the scratch area after it is passed to the requesting task.

INTO *return-scratch-data-location-variable*

Specifies the location to which the requested scratch record is transferred.

Return-scratch-data-location-variable is the name of a variable data field in the dialog's record buffers.

TO *end-scratch-data-location-variable*

Specifies the end of the buffer area allocated for the requested scratch record.

End-scratch-data-location-variable is the name of a dummy byte field or the name of a variable data field that contains a data item not associated with the requested scratch record.

The field specified by *end-scratch-data-location-variable* must immediately follow the last byte of the buffer area allocated for the requested scratch record.

MAX LENGTH *scratch-data-max-length*

Specifies the length of the buffer area allocated for the requested scratch record.

Scratch-data-max-length is the name of a variable data field that contains the length or the length itself, expressed as a numeric constant.

If neither **TO *end-scratch-data-location-variable*** nor **MAX LENGTH *scratch-data-max-length*** is specified, the length of the location is the length of *return-scratch-data-location-variable*.

CURRENT

Obtains the record that is current of the scratch area specified by *scratch-area-id*.

FIRST

Obtains the first record in the scratch area specified by *scratch-area-id*.

LAST

Obtains the last record in the scratch area specified by *scratch-area-id*.

NEXT

Obtains the record that follows the current record of the scratch area specified by *scratch-area-id*.

NEXT is the default when you specify no other scratch record to be obtained.

If currency is not established, NEXT is equivalent to FIRST.

PRIOR

Obtains the record that precedes the current record of the scratch area specified by *scratch-area-id*.

If currency is not established, PRIOR is equivalent to LAST.

RECORD ID *scratch-record-id*

Obtains the record identified by *scratch-record-id*.

Scratch-record-id is either the name of a variable data field that contains the scratch record id or the scratch record id itself, expressed as a numeric constant.

RETURN LENGTH into *scratch-data-actual-length-variable*

Returns the untruncated length of the obtained scratch record to the location specified by *scratch-data-actual-length-variable*.

Scratch-data-actual-length-variable is the name of a numeric field in the dialog's record buffers.

error-expression

Specifies the status codes that are returned to the dialog.

Usage***Considerations***

If the scratch record is larger than the allocated buffer area, the record is truncated as necessary.

If autostatus is not in use, a dialog's error-status field indicates the outcome of a GET SCRATCH command:

Status Code	Meaning
0000	The request was executed successfully
4303	The requested scratch area cannot be found
4305	The requested scratch record cannot be found
4307	An I/O error occurred during processing
4319	The dialog's storage location is too small for the requested scratch record. The record was truncated accordingly
4331	The CA ADS internal parameter list was invalid
4332	The derived length of the scratch record data area is negative.

Example

The following example illustrates the use of the GET SCRATCH command to copy the last record in scratch area CUSTAREA to a location in the dialog's record buffers identified by CUSTWORK. The record is retained in the scratch area for later access:

```
GET SCRATCH AREA ID 'CUSTAREA' KEEP LAST
INTO CUSTWORK MAX LENGTH REC-LENGTH.
```

More information:

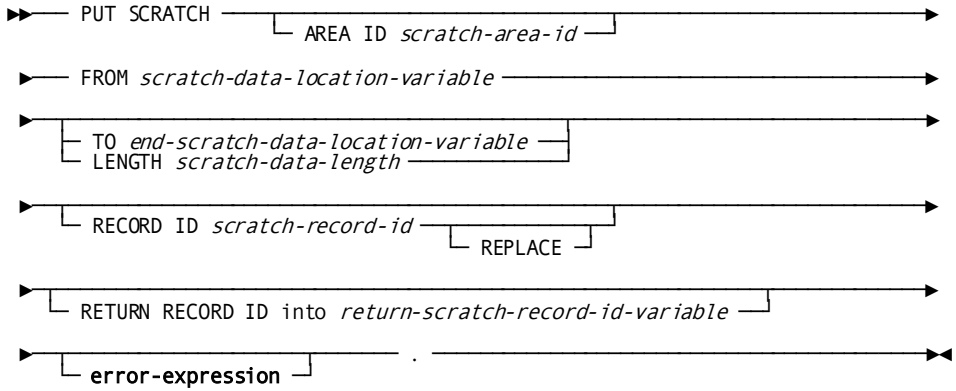
[Error Handling](#) (see page 277)

PUT SCRATCH

Purpose

Stores or replaces a scratch record in the scratch area.

Syntax



Parameters

AREA ID *scratch-area-id*

Specifies the area in the scratch area where the record will be stored.

Scratch-area-id is either the name of a variable data field that contains a scratch area id or the 1- to 8-character scratch area id itself, enclosed in single quotation marks.

If *scratch-area-id* is not specified, a null scratch area id (that is, eight blanks) is assumed.

FROM *scratch-data-location-variable*

Specifies the location of the data to be stored in the queue record.

Scratch-data-location-variable is the name of a variable data field in the dialog's record buffers.

TO *end-scratch-data-location-variable*

Specifies the end of the buffer area that contains the data to be stored in the scratch record.

End-scratch-data-location-variable is either the name of a dummy byte field or the name of a variable data field that contains a data item not associated with the scratch record data.

The field specified by *end-scratch-data-location-variable* must immediately follow the last byte of the buffer area that contains the scratch record data.

LENGTH *scratch-data-length*

Specifies the length, to be specified in bytes, of the buffer area that contains the data to be stored in the scratch record.

Scratch-data-length is either the name of a variable data field that contains the length or the length itself, expressed as a numeric constant.

If neither *TO end-scratch-data-location-variable* nor *LENGTH scratch-data-length* is specified, the length of the location is the length of *scratch-data-location-variable*.

RECORD ID *scratch-record-id*

Assigns an id to the scratch record being stored.

Scratch-record-id is either the name of a variable data field that contains the scratch record id or the scratch record id itself, expressed as a numeric constant.

The scratch record id can subsequently be used to retrieve or delete the associated scratch record.

The scratch record id is stored as a binary fullword.

REPLACE

Replaces the scratch record identified by *scratch-record-id* with the scratch record being stored.

RETURN RECORD ID into *return-scratch-record-id-variable*

Returns a system-assigned scratch record id to the location specified by *return-scratch-record-id-variable*.

Return-scratch-record-id-variable is the name of a variable data field in the dialog's record buffers.

Return-scratch-record-id-variable cannot be defined as a doubleword binary field.

The scratch record id can subsequently be used to retrieve or delete the associated scratch record.

The system assigns a scratch record id if one is not specified in the RECORD ID parameter.

error-expression

Specifies the status codes that are returned to the dialog.

Usage

Considerations

If autostatus is not in use, a dialog's error-status field indicates the outcome of a PUT SCRATCH command:

Status Code	Meaning
0000	The request to add a scratch record was executed successfully
4307	An I/O error occurred during processing
4317	The request to replace a scratch record was executed successfully
4322	The request to store a scratch record cannot be executed because the scratch record id already exists within the scratch area and the REPLACE option was not specified
4331	The CA ADS internal parameter list was invalid
4332	The derived length of the scratch record data location is negative

Example

The following example illustrates the use of the PUT SCRATCH command:

```
PUT SCRATCH AREA ID 'CUSTAREA' FROM CUSTWORK LENGTH REC-LENGTH  
RETURN RECORD ID INTO REC-ID.
```

More information:

[Error Handling](#) (see page 277)

Chapter 19: Subroutine Control Commands

This section contains the following topics:

- [Overview](#) (see page 505)
- [CALL](#) (see page 505)
- [DEFINE](#) (see page 506)
- [GOBACK](#) (see page 507)

Overview

CA ADS subroutine control commands are used to define and call subroutines within a process.

The subroutine control commands are listed in the following table. Each command is discussed later in this section.

Subroutine Control Commands

Command	Description
CALL	Passes control to a predefined subroutine
DEFINE	Establishes an entry point for a subroutine and defines subroutine processing
GOBACK	Terminates subroutine processing and returns control to the command following the associated CALL command

CALL

Purpose

Passes control to a predetermined subroutine.

Syntax

► CALL *subroutine-name* ◀

Parameter

subroutine-name

Specifies the 1- to 8-character name of the subroutine to which control is passed. The subroutine name is defined by the DEFINE command, described below.

Usage

When the CA ADS runtime system encounters a CALL command, processing control passes to the beginning of the named subroutine. Processing continues through the subroutine until CA ADS encounters a GOBACK command, or a control command

If no GOBACK or control command occurs before the end of the subroutine, the runtime system automatically returns control to the command that immediately follows the associated CALL command.

Considerations

- A CALL statement can occur in the body of a process or within a subroutine definition.
- Subroutine calls can be nested up to ten levels.
- The called subroutine must be defined by using the DEFINE command, described later in this section, and must be coded later in the process than the CALL command.

More information:

[Control Commands](#) (see page 325)

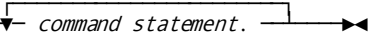
[GOBACK](#) (see page 507)

DEFINE

Purpose

Establishes an entry point for a subroutine and to define the subroutine processing. At runtime, a subroutine is executed when it is named in a CALL statement.

Syntax

▶— DEFINE subroutine *subroutine-name* — . —  *command statement.* —▶

Parameters

subroutine-name

Specifies the 1- to 8-character name of the subroutine being defined.

Subroutine-name must be unique within the process.

command-statement

Specifies the process commands that define the subroutine.

Command-statement can be any process command statement except DEFINE SUBROUTINE.

Usage*Considerations*

- Each command statement must be terminated with a period (.).
- Any number of subroutine definitions can be coded at the end of a process.
- A subroutine definition is terminated by the occurrence of another subroutine definition or by the end of the process code.
- DEFINE is the only process command that can follow a subroutine definition.
- Subroutine order and physical placement within the process code are important. The physical placement of multiple subroutines depends on the code within each subroutine. All called subroutines must be physically coded lower than the calling subroutine.

Example:

```
CALL SUBROUTINE-A.
....
CALL SUBROUTINE-B.
....
DEFINE SUBROUTINE-A.
    (within this code which is subroutine-d)
DEFINE SUBROUTINE-D.
    (within this code which is subroutine-b)
DEFINE SUBROUTINE-B.
```

GOBACK**Purpose**

Terminates subroutine processing.

Syntax

▶▶ GOBACK — . —————▶▶

Usage

At run time, GOBACK returns processing control to the command following the CALL that passed control to the subroutine.

Considerations

- A GOBACK command can be coded wherever logically appropriate within the body of a subroutine.
- A GOBACK command is automatically generated by CA ADS to ensure that GOBACK is the last command in the subroutine.

Example

The following example uses the CALL, DEFINE, and GOBACK commands to illustrate the use of a subroutine within a process:

```
FIND CALC CUSTOMER.
IF DB-REC-NOT-FOUND
THEN
  DO.
    STORE CUSTOMER.
    CALL UPDMAIL.
    DISPLAY MSG TEXT 'CUSTOMER ADDED'.
  END.
ELSE
  DO.
    MODIFY CUSTOMER.
    CALL UPDMAIL.
    DISPLAY MESSAGE TEXT 'CUSTOMER CHANGED'.
  END.
DEFINE SUBROUTINE UPDMAIL.
  MOVE 1 TO SB.
  WHILE SB LE 3
  REPEAT.
    MOVE CUST-INT(SB) TO MAIL-INT.
    FIND CALC MAILIST.
    CONNECT CUSTOMER TO MAILIST.
    ADD 1 TO SB.
  END.
GOBACK.
```

Chapter 20: Utility Commands

This section contains the following topics:

[Overview](#) (see page 509)

[ABORT](#) (see page 510)

[ACCEPT](#) (see page 513)

[INITIALIZE RECORDS](#) (see page 515)

[SNAP](#) (see page 516)

[TRACE](#) (see page 518)

[WRITE PRINTER](#) (see page 519)

[WRITE TO LOG/OPERATOR](#) (see page 523)

Overview

CA ADS utility commands are used to reinitialize record buffers, transmit data to be printed, and provide information about the current task.

The utility commands are summarized in the following table.

Summary of Utility Commands

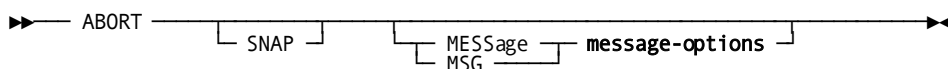
Command	Description
ABORT	Abnormally terminates an application
ACCEPT	Retrieves task-related information
INITIALIZE RECORDS	Reinitializes dialog record buffers
SNAP	Requests a snap dump of selected memory areas
WRITE PRINTER	Transmits data from a dialog to an CA IDMS/DC or DC/UCF print queue
WRITE TO LOG/OPERATOR	Sends a message to the log file or to the operator's console (CA ADS Batch only)

ABORT

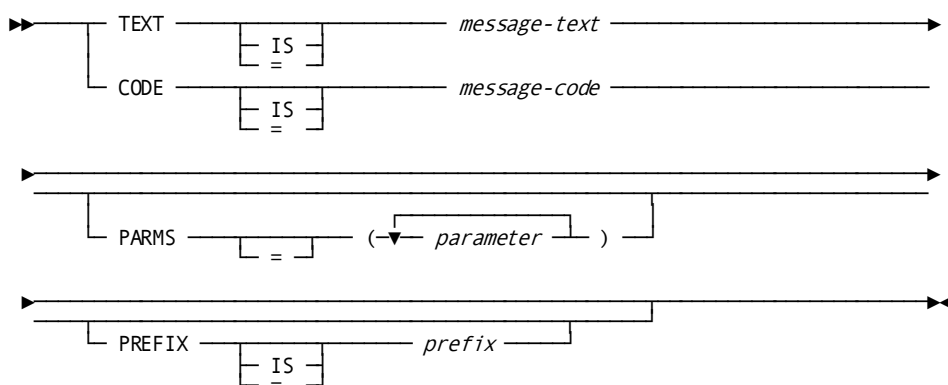
Purpose

Terminates the execution of the current task.

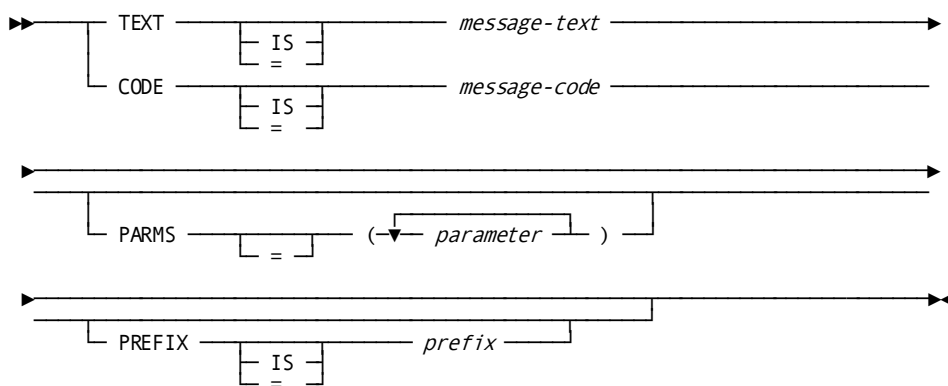
Syntax



Expansion of message-options



Expansion of message-options



Parameters

SNAP

Writes a formatted snap dump to the CA IDMS/DC or DC/UCF (DC/UCF) log. Snap dumps can be printed by means of the PRINT LOG utility.

Note: For more information about PRINT LOG, see the *CA IDMS Utilities Guide*.

MESSAge

Specifies a message to be displayed on the Dialog Abort Information screen and written to the system log. If a MESSAGE clause is not specified, system message DC174020 is used (text of this message can be changed by using IDD):

ADS/ON-LINE ABORT. USER SPECIFIED ABORT WITH NO MESSAGE CODE/TEXT

MSG can be used in place of MESSAGE.

Note: For more information about altering message text using IDD, see the *CA IDMS IDD Quick Reference Guide*, under the MESSAGE command.

TEXT IS *message-text*

Specifies the text of a message to be sent to the system log or, if batch, sent to the console and batch log file.

This can be either the name of a variable data field containing the message text or the text string itself, enclosed in single quotation marks.

The text string can contain up to 240 displayable characters.

CODE IS *message-code*

Specifies the message dictionary code of a message to be displayed in an online map's message field or sent to the log file in a batch application.

This can be either the name of a variable data field that contains the message code or the 6-digit code itself, expressed as a numeric literal.

In a batch application, the message is also sent to the operator, if directed by the destination specified in the dictionary.

PARMS = *parameter*

Introduces a replacement parameter for each variable field in the stored message identified by *message-code*. The parameter can be either the name of an EBCDIC or unsigned zoned decimal variable data field that contains the parameter value or the actual parameter value, enclosed in single quotation marks.

Up to nine replacement parameters can be specified for a message. Multiple parameters must be specified in the order in which they are numbered and separated by blanks or commas.

The parameter value must contain displayable characters. At run time, each variable data field in a stored message expands or contracts to accommodate the size of its replacement parameter. A replacement parameter can be a maximum of 240 bytes.

PREFIX IS *prefix*

Overrides the default prefix of a dialog and a map. *Prefix* specifies an EBCDIC or unsigned zoned decimal variable data field that contains a 2-character prefix or the 2-character prefix itself, enclosed in single quotation marks

Usage

Considerations

- When a dialog issues an ABORT command, CA ADS abnormally terminates the current task and returns control to DC/UCF. A snap dump of all memory areas maintained for the current CA ADS runtime session at the time of the abort can be requested.
- The CA ADS runtime system provides a diagnostic screen that displays information about an abnormally terminated dialog. The diagnostic screen is enabled for an installation by means of the DIAGNOSTIC SCREEN clause of the system generation ADSO statement.

Note: For more information about the ADSO statement, see the *CA IDMS System Generation Guide*.

If the diagnostic screen is not enabled when an ABORT command is issued, a system error message (DC466019) is displayed. If a message code is specified in the ABORT command and the dictionary message specifies a destination of log, the message is also sent to the system log.

- Up to nine replacement parameters can be specified for a message.
- Multiple message parameters must be separated by blanks or commas.
- Message parameters must be specified in the order in which they occur in the stored message.
- Within the message definition in the dictionary, symbolic parameters are identified by an ampersand (&) followed by a two-digit numeric identifier. These identifiers can appear in any order. The position of the replacement values in the PARMs parameter must correspond directly to the two-digit numeric identifiers in the message; the first value corresponds to &01, the second to &02, and so forth. For example, assume that the stored message text is as follows:

```
THIS IS TEXT &01 AND &03 OR &02
```

The PARMs parameter reads PARMs=('A','B','C'). The resulting text would read as follows:

```
THIS IS TEXT A AND C OR B
```

Example

The following example illustrates the use of the ABORT command:

```
ADD ACC-BAL TO TOT-BAL.  
ADD 1 TO CONTROL-CTR.  
IF CONTROL-CTR < 100  
THEN  
    INVOKE 'CEXDR008' .  
ELSE  
    ABORT SNAP MSG TEXT CUST-NUM.
```

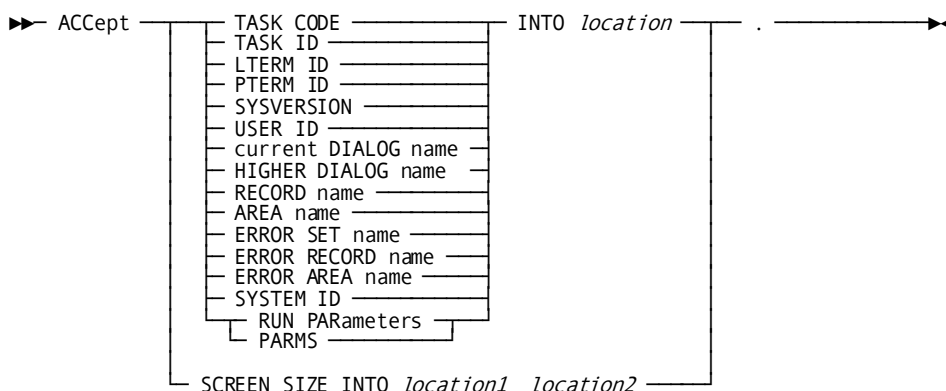

More information:

[CA ADS Runtime System](#) (see page 119)

ACCEPT

Purpose

Retrieves information about the current task and dialog. (In CA ADS Batch, ACCEPT is used to accept runtime parameters into a storage location.)

Syntax**Parameters****TASK CODE**

Retrieves the eight-character code used to invoke the current task.

TASK ID

Retrieves the DC/UCF-assigned task identification number. The task id is a unique sequence number stored in a binary fullword field. The task id is zero when DC/UCF is started and is incremented by one for each new task added to the system.

LTERM ID

Retrieves the eight-character identification of the logical terminal associated with the current task.

PTERM ID

Retrieves the eight-character identification of the physical terminal associated with the current task.

SYSVERSION

Retrieves the version number, in the range 0 to 32767, of the DC/UCF system currently in use. The version number is stored in a binary halfword field.

USER ID

Retrieves the user identification.

In CA ADS, the 32-character identification of the user signed on to the logical terminal associated with the current task is retrieved. If no user is signed on, a null user id (that is, 32 blanks) is returned.

In CA ADS Batch, the user identification specified in the USER (REQUESTOR) input parameter is retrieved.

current DIALOG name

Retrieves the name of the current dialog.

HIGHER DIALOG name

Retrieves the name of the dialog that is operative at the next higher level in the current application thread.

RECORD name

Retrieves the name of the record that is current of run unit for the issuing dialog.

AREA name

Retrieves the name of the area that is current of area for the issuing dialog.

ERROR SET name

Retrieves the name of the last set involved in an operation that resulted in an error condition.

ERROR RECORD name

Retrieves the name of the last record involved in an operation that resulted in an error condition.

ERROR AREA name

Retrieves the name of the last area involved in an operation that resulted in an error condition.

RUN PARAmeters

(CA ADS Batch only) Retrieves runtime parameters, which are specified in the JCL PARM parameter (z/OS, z/VSE Release 2.1, and z/VM) or in a JOB VARIABLE statement. If no runtime parameters are specified in the JCL, the storage location is blank filled.

PARMS can be used in place of RUN PAR.

SYSTEM ID

Retrieves the 8 character name(nodename) by which the DC/UCF system is known to other nodes in the DC/UCF communications network.

INTO *location*

Specifies the location to which the information is moved.

Location is the name of a variable data field in the dialog's record buffers. The specified field must have an appropriate picture and usage for the value being retrieved.

SCREEN SIZE INTO *location1 location2*

Retrieves the dimensions (that is, the number of rows and columns) of the physical terminal screen associated with the current task.

Location1 is the name of a numeric variable data field in the dialog's record buffers to which the number of rows moved.

Location2 is the name of a numeric variable data field in the dialog's record buffers to which the number of columns is moved.

INITIALIZE RECORDS

Purpose

Reinitializes one or more of a dialog's record buffers.

Syntax

```

▶▶— INITIALize records ———— ALL ————— . —————▶▶
      |
      | ( — record-name — )
  
```

Parameters**ALL**

Reinitializes the buffers for all subschema, map, and work records referenced by the issuing dialog, regardless of which dialog originally allocated the buffers.

record-name

Reinitializes the buffer for each record specified by *record-name*. The named records must be associated with the issuing dialog.

Usage

Considerations

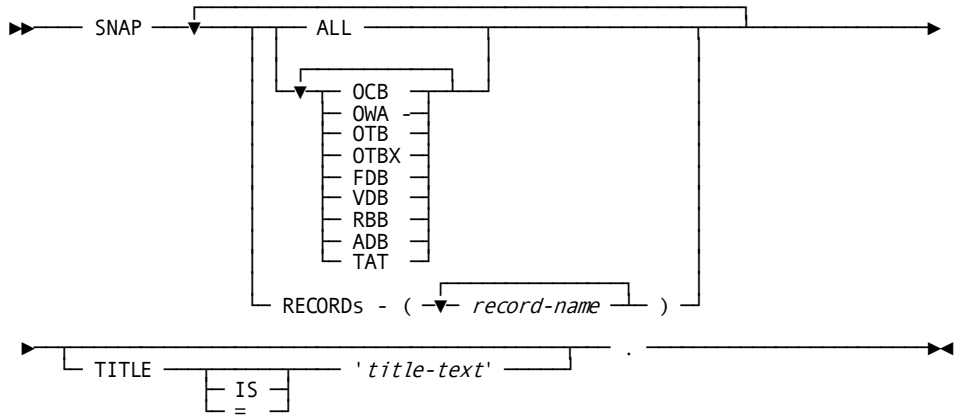
- After execution of an INITIALIZE RECORDS command, the record elements in the specified record buffers contain their original values, as follows:
 - If the record element is defined with a VALUE IS clause, the buffer is reinitialized with the defined value.
 - If the record element definition has no VALUE IS clause, the buffer is reinitialized either with spaces (EBCDIC and DBCS fields) or with zeros of the appropriate data type (numeric fields).

SNAP

Purpose

Request a snap dump of the contents of one or more areas maintained in memory for CA ADS. The dump produced by a SNAP command can be used to assess the use of system resources by an executing dialog.

Syntax



Parameters

ALL

Writes all areas of memory maintained for the issuing dialog to the SNAP dump.

OCB

Keyword which requests the CA ADS control block. The OCB contains CAADS system parameters specified in the system generation ADSO statement.

OWA

Keyword which specifies the CA ADS online work area. The OWA is maintained as a temporary storage buffer for application and dialog information used during CA ADS runtime processing. The OWA is not maintained across tasks.

OTB

Keyword which specifies the CA ADS terminal block. The OTB contains information about the current CA ADS session. The OTB is maintained across tasks.

OTBX

Keyword which specifies the CA ADS terminal block extension. The OTBX is an extension of the OTB and contains pointers to the TAT, the ADSO-APPLICATION-GLOBAL-RECORD record buffer, the RBB, and the ADB for the currently executing application. The OTBX exists only for applications defined using the application compiler (ADSA).

FDB

Keyword which specifies the fixed dialog block. The FDB is the dialog load module created by the dialog compiler (ADSC). Information in the FDB includes executable process code and parameters required to execute the dialog, and information on the maps and records associated with the dialog.

VDB

Keyword which specifies the variable dialog block. One VDB exists for each operative dialog. A VDB contains runtime variable information about a dialog, such as the status of map fields, information concerning flow of control, addresses of records used by the dialog, and the address of the executing command.

The VDB is created dynamically for the issuing dialog at runtime.

RBB

Keyword which specifies the record buffer block. The RBB contains header information and buffers for all records associated with the current application.

ADB

Keyword which specifies the application definition block. The ADB is the application load module created by the CA/ADS online application compiler. The ADB contains the application information supplied on the definition screens during an application compiler session. The ADB exists only if the application is defined using the application compiler.

TAT

Keyword which specifies the task application table. The TAT contains the names of task codes used to initiate applications and the names of the applications (ADBs) thus initiated. The TAT exists only if there are applications on the system that are defined using the application compiler.

RECORDS *record-name*

Includes information associated with the specified subschema, map, or work records in the SNAP dump. The information is taken from the RBB; it includes data, but no headers, from the buffers for the named records.

Record-name must be associated with the issuing dialog.

TITLE is '*title-text*'

Specifies a title for the SNAP dump.

Title-text is a 1- to 90-character string enclosed in single quotation marks. The specified title is printed on the hard-copy listing of the SNAP dump.

Usage

Snap dumps are written to the DC/UCF log and can be printed by using the PRINT LOG print log utility.

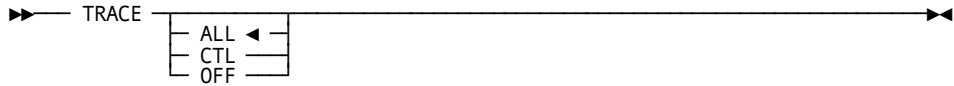
Note: For more information about PRINT LOG, see the *CA IDMS Utilities Guide*.

TRACE

Purpose

Activates the CA ADS trace facility; with the OFF parameter, deactivates the CA ADS trace facility.

Syntax



Parameters

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

OFF

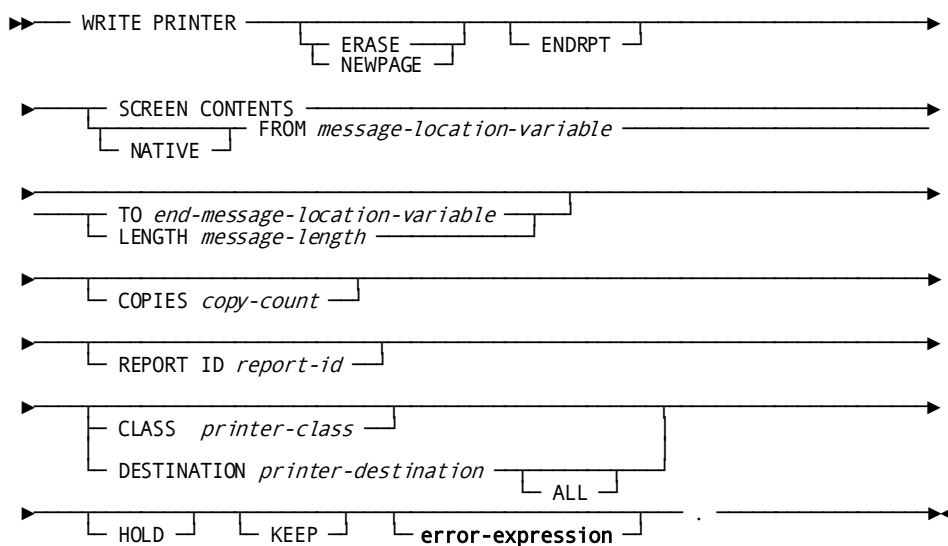
Deactivates the trace facility.

WRITE PRINTER

Purpose

Sends data from a dialog to a printer or to a file.

Syntax



Parameters

ERASE

Specifies that the data being transmitted is to be printed on a new page.

`NEWPAGE` may be used in place of `ERASE`.

ENDRPT

Specifies that the data being transmitted is the last record of the specified report. If `ENDRPT` is specified, the report is printed before the current task terminates.

SCREEN CONTENTS

Transmits the contents of the currently displayed screen to the print queue. This option is valid only for 3270-type terminals. If SCREEN CONTENTS is specified for another terminal type, an error condition results.

NATIVE

Specifies that the data stream being transmitted contains line and device control characters. If NATIVE is not specified, DC/UCF automatically inserts the necessary characters.

FROM *message-location-variable*

Specifies the location of the data to be transmitted to the print queue.

Message-location-variable is the name of a variable data field in the dialog's record buffers.

TO *end-message-location-variable*

Specifies the end of the buffer area that contains the data to be transmitted.

End-message-location-variable is the name of a dummy byte field or the name of a variable data field that contains a data item not associated with the data being transmitted.

The field specified by *end-message-location-variable* must immediately follow the last byte of the buffer area that contains the data to be transmitted.

LENGTH *message-length*

Specifies the length of the buffer area that contains the data to be transmitted.

Message-length is either the name of a numeric variable data field that contains the length in bytes or the length itself, in bytes, expressed as a numeric constant.

COPIES *copy-count*

Specifies the number of report copies to print.

Copy-count-number is either the name of a numeric variable data field that contains the copy count or the number of report copies itself, expressed as a numeric constant in the range 1 through 255.

If COPIES is not specified, the number of copies defaults to 1.

REPORT ID *report-id*

Specifies the report with which the transmitted data is associated. The report id must be an integer in the range 1 through 255.

Report-id is either the name of a numeric variable data field that contains the report id or the report id itself expressed as a numeric constant.

If REPORT ID is not specified, the report id defaults to 1.

CLASS *printer-class*

Specifies the print class, in the range 1 through 64, to which the report is assigned.

Printer-class is either The name of a numeric variable data field that contains the print class or the print class itself, expressed as a numeric constant.

If no print class is specified, the physical terminal default is used.

DESTINATION *printer-destination*

Specifies the printer to which the report is routed.

Printer-destination is either the name of a variable data field that contains the 1 to 8-character destination or the destination itself, enclosed in single quotation marks.

If no print destination is specified, the physical terminal default is used.

ALL

Specifies that the report is to be printed on all of the logical terminals at the specified print destination. If ALL is not specified, the report is printed on only one of the logical terminals.

HOLD

Specifies that DC/UCF is not to print the report until a system operator releases it with a DCMT VARY REPORT command.

Note: For more information about DCMT commands, see the *CA IDMS System Tasks and Operator Commands Guide*.

KEEP

Specifies that each time DC/UCF finishes printing the report, the report is to be kept instead of deleted. The report can be reprinted or deleted with a DCMT VARY REPORT command.

If KEEP is not specified, the report is deleted once it is printed.

error-expression

Specifies the status codes that are returned to the dialog.

Usage***Definition***

The WRITE PRINTER command is used to transmit data from the issuing dialog to a DC/UCF printer terminal and to initiate printing of the transmitted data. Data is passed first to a report queue maintained by DC/UCF and then to the printer.

Each line of data transmitted by a WRITE PRINTER request is considered a record. Each record is associated with a particular report in the report queue. A report consists of one or more records. The report queue can contain up to 256 active reports for any one task.

If autostatus is not in use, a dialog's error-status field indicates the outcome of a WRITE PRINTER command:

Status Code	Meaning
0000	The request was executed successfully
4807	An I/O error occurred in placing the record in the print queue
4818	The DC/UCF system has no logical terminals associated with a printer
4821	The specified printer destination is invalid
4838	The variable storage field that contains the record to be printed was not allocated
4845	The output terminal type is not correct for the WRITE PRINTER request
4846	A terminal I/O error occurred while attempting to print the contents of a screen.

Considerations

- A report is terminated when the current run unit is terminated or when a WRITE PRINTER ENDRPT command is issued. Note that a run unit can be extended across dialogs by using the LINK command.
- A process can contain multiple WRITE PRINTER requests, each for a different report. DC/UCF maintains the records associated with each report individually, ensuring that records associated with one report are not interspersed with records associated with other reports when the reports are printed.
- Printing is initiated either explicitly by a WRITE PRINTER request or implicitly by termination of the current task. If a task terminates abnormally, all data in the print queue is deleted, unless it was previously committed by a COMMIT TASK command.
- Each printer has one or more DC/UCF classes or destinations. The print class and destination for a report are assigned when the WRITE PRINTER command is issued for the first record in the report. The entire report is printed on the first available printer with the specified class.
- A default print class and print destination can be specified for applications defined using the application compiler. The defaults are specified on the General Options screen.

At runtime, the defaults are stored in the AGR-PRINT-CLASS and AGR-PRINT-DESTINATION record elements of the ADSO-APPLICATION-GLOBAL-RECORD. WRITE PRINTER commands can select these defaults by specifying these record elements in the CLASS and DESTINATION parameters.

More information:

[CA ADS Application Compiler \(ADSA\)](#) (see page 51)

[System Records](#) (see page 567)

[CA ADS Runtime System](#) (see page 119)

[Control Commands](#) (see page 325)

[Error Handling](#) (see page 277)

WRITE TO LOG/OPERATOR

Purpose

Sends messages to the log file or, in the batch environment, to the log file and the operator's console.

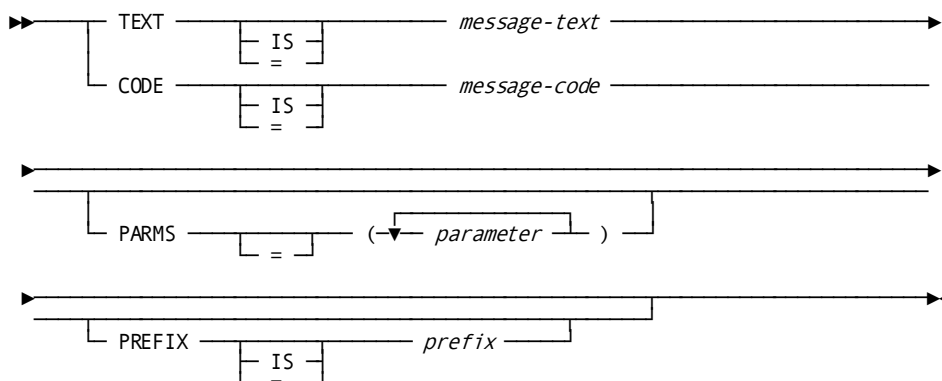
Syntax

```

▶▶ WRITE to [ LOG OPERator ] [ MESSAGE MSG ] message-options . ▶▶

```

Expansion of Message-Options



Parameters

LOG/OPERATOR

Sends a message to the system log or to the operator's console. OPERATOR can be specified only in the batch environment.

MESSAge *message-options*

Identifies message to be displayed.

MSG can be used in place of MESSAGE.

TEXT IS *message-text*

Specifies the text of a message to be sent to the system log or, if batch, sent to the console and batch log file.

Message-text specifies either the name of a variable data field containing the message text or the text string itself, enclosed in single quotation marks.

The text string can contain up to 240 displayable characters.

CODE IS *message-code*

Specifies the message dictionary code of a message to be displayed in an online map's message field or sent to the log file in a batch application.

In a batch application, the message is also sent to the operator, if directed by the destination specified in the dictionary.

Message-code specifies either the name of a variable data field that contains the message code or the 6-digit code itself, expressed as a numeric literal.

PARMS = *parameter*

Specifies a replacement parameter for each variable field in the stored message identified by *message-code*.

Up to nine replacement parameters can be specified for a message. Multiple parameters must be specified in the order in which they are numbered and separated by blanks or commas.

Parameter specifies either the name of an EBCDIC or unsigned zoned decimal variable data field that contains the parameter value or the actual parameter value, enclosed in single quotation marks.

The parameter value must contain displayable characters. At run time, each variable data field in a stored message expands or contracts to accommodate the size of its replacement parameter. A replacement parameter can be a maximum of 240 bytes.

PREFIX IS *prefix*

Overrides the default prefix of a dialog and a map.

Prefix must either specify an EBCDIC or unsigned zoned decimal variable data field that contains a 2-character prefix or the 2-character prefix itself, enclosed in single quotation marks

Usage

Considerations

- Up to nine replacement parameters can be specified for a message.
- Multiple parameters must be separated by blanks or commas.
- Multiple parameters must be specified in the order in which they occur in the stored message.

Chapter 21: Cooperative Processing Commands

This section contains the following topics:

[Using SEND/RECEIVE Commands](#) (see page 527)

[Sample Cooperative Application](#) (see page 528)

[SEND/RECEIVE Commands](#) (see page 534)

[ALLOCATE](#) (see page 535)

[CONFIRM](#) (see page 538)

[CONFIRMED](#) (see page 539)

[CONTROL SESSION](#) (see page 540)

[DEALLOCATE](#) (see page 541)

[PREPARE-TO-RECEIVE](#) (see page 543)

[RECEIVE-AND-WAIT](#) (see page 543)

[REQUEST-TO-SEND](#) (see page 545)

[SEND-DATA](#) (see page 545)

[SEND-ERROR](#) (see page 546)

[Design Guidelines](#) (see page 547)

[Understanding Conversation States](#) (see page 548)

[Testing APPC Status Codes and System Fields](#) (see page 552)

Using SEND/RECEIVE Commands

SEND/RECEIVE commands allow you to create applications that execute cooperatively on two systems. You can exchange information between an CA ADS application (on the mainframe) and:

- An CA ADS application (running under a different CA IDMS/DC system)
- An Assembler program (running under CA IDMS/DC)
- Any program using APPC/LU6.2, regardless of the program's platform

Applications that execute cooperatively on two systems are taking advantage of **cooperative processing**.

Client and Server

With cooperative processing, labor is divided so that one side of the application acts as a **client** and the other acts as a **server**. The client provides front-end processing for the user (like data input, validation, and display). The server provides back-end processing (like database access and the implementation of business rules and procedures).

How Cooperative Processing Works

The SEND/RECEIVE commands in CA ADS follow the standards set for Advanced Program to Program Communication (APPC).

APPC is an IBM standard that provides enhanced Systems Network Architecture (SNA) support for distributed processing. APPC enables 2 processors to work together: it describes the protocols the 2 processors' programs use to communicate as they execute a single distributed transaction.

APPC is composed of logical and physical definitions of the system network. The logical component is the LU 6.2 protocol, which defines the rules that govern the exchange of information between the 2 programs.

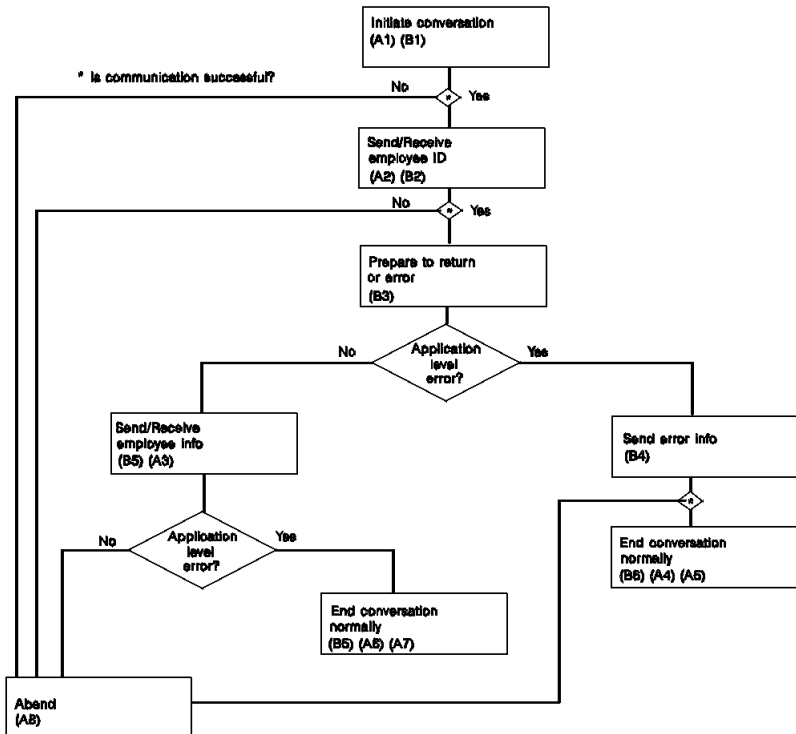
The LU 6.2 protocol structures a program-to-program conversation like a polite conversation: one side talks and the other side listens.

- One program (the **primary** program) starts the conversation by calling the other program (the **secondary** program).
- The 2 programs agree on the rules governing the conversation before the conversation can continue.
- The conversation goes back and forth, with the current speaker (in **send state**) always in control:
 1. When the listener (in **receive state**) wants to speak, the listener requests permission.
 2. If the speaker approves the request, the listener switches to send state and the speaker to receive state.

Sample Cooperative Application

This application retrieves employee information from the database. A user on the PC uses Program A (the client) to send an employee ID to the mainframe. Dialog B (the server) on the mainframe returns employee information to the PC.

The flowchart below describes the flow of information in the cooperative application. The communication commands in Program A and Dialog B are labeled in sequence (A1 through A8 and B1 through B6).



Program A: Client Listing (PC)

```
!*****
!
! This module sends up the employee ID and receives employee
! data back from mainframe server dialog A240D1.

LOCAL Emp_data_group DICT.

SHOW TEXT '&nAccessing. remote server ...'.

! Since we are going to use this data in some meaningful fashion we
! define a formatted conversation. Note the use of "FORMAT". This
! means that all data is automatically translated from PC data types
! to mainframe data types. IBM APPC LU 6.2 calls for the use of a return
! code to verify the state of a particular APPC verb. We use "Appccode".

(A1)    ALLOCATE CONNECT 'SYSTEM55' TPN 'A240D1' FORMAT.
        AFTER Comm-error CALL SR_Abend.

! Note the use of a local subroutine called SR_Abend.
!
! Let's send up the employee ID.
!

(A2)    SEND-DATA Emp_id.
        AFTER Comm-error CALL SR_Abend.

! "Turn the line around" and wait for server to send down
! the requested data. This flushes the communications
! buffer and passes control to the server dialog.

(A3)    RECEIVE-AND-WAIT Emp_data_group.
!
! If an employee was not found, then the server's SEND-ERROR
! shows PROG-ERROR at the client side of the conversation.
!
AFTER Prog-error
DO.
! One more receive to get the deallocate state...
(A4)    RECEIVE-AND-WAIT.
        AFTER Deallocate-normal
        DO.
(A5)    DEALLOCATE LOCAL.
        AFTER Comm-error CALL SR_Abend.
        INITIALIZE ( emp_first_name, emp_last_name,
                    office_code, emp_street, emp_city,
                    emp_state, emp_zip_first_five, status ).
        DISPLAY TEXT '&wEmployee. not on file. Try again.'
```

```

        END.
    END.

    IF Appcode LT 0 OR What-received NE 'DATA-COMPLETE'
        CALL SR_Abend.
    !
    ! We seem to have received the data ok.
    !
    ! The server is still in control of the conversation, so we
    ! do one more receive-and-wait to receive the fact that the
    ! server has deallocated. This puts the client in deallocate state,
    ! allowing the client to deallocate normally and regain control.
    !
(A6)    RECEIVE-AND-WAIT.
        AFTER Deallocate-normal
        DO.
(A7)    DEALLOCATE LOCAL.
        ! Now move data to form fields and display it.
        Emp_first_name    = Emp_data_group.emp_first_name.
        Emp_last_name     = Emp_data_group.emp_last_name.
        Emp_street        = Emp_data_group.emp_street.
        Emp_city          = Emp_data_group.emp_city.
        Emp_state         = Emp_data_group.emp_state.
        Emp_zip_first_five = Emp_data_group.emp_zip_first_five.
        Status            = Emp_data_group.status.
        Office_code       = Emp_data_group.office_code.

        DISPLAY FIELD Emp_id TEXT ' '.
    END.

    AFTER Comm-error CALL SR_Abend.

!***** Local subroutine *****

    DEFINE SR_Abend.

        SHOW TEXT '&cComm. Error. APPCCODE='&svb.&svb. Appcode
                &svb.&svb. ', APPCERC=' &svb.&svb. Appcerc.
(A8)    DEALLOCATE ABEND.
        DISPLAY.
    !

```

Dialog B: Server listing (Mainframe)

```
!*****
DIALOG A240D1 (CA ADS)

PROCESS NAME IS A240D1-PREMAP

! THE INTENT OF THIS MAPLESS DIALOG IS TO RECEIVE THE EMPLOYEE-ID
! FROM THE PC APPLICATION AND THEN OBTAIN THE APPROPRIATE RECORDS FROM
! THE CA IDMS/DB DATABASE, BASED ON THE CONTENTS OF THE EMPLOYEE ID.
! THEN CERTAIN ELEMENTS IN THE RECORDS ARE SENT BACK TO THE PC FOR FURTHER
! PROCESSING.
!
! IN ALL APPC CONVERSATIONS THERE ARE LOCAL AND REMOTE PROGRAM.
! ONLY ONE OF THE PROGRAMS CAN CONTROL THE CONVERSATION BUT CONTROL
! CAN BE PASSED BACK AND FORTH BETWEEN LOCAL AND REMOTE APPLICATIONS.
!
! IN THIS APPLICATION THE DEFAULT SENDER IS THE PC PROGRAM BECAUSE IT
! ISSUED THE ALLOCATE. ON THE RECEIVING END A "CONTROL SESSION" IS
! INITIATED IN RESPONSE TO THE ALLOCATE.
!
! NOTE THAT THE USE OF THE FORMAT/NOFORMAT PARAMETERS MUST BE THE SAME AT
! BOTH ENDS OF THE CONVERSATION.

READY.
(B2)CONTROL SESSION FORMAT.
  IF APPCCODE LT 0 THEN
    CALL SR-ABEND.

! LET'S GET THE EMPLOYEE ID FROM THE PC.
(B2)RECEIVE-AND-WAIT EMP-ID-WORK.
  IF APPCCODE LT 0 OR WHAT-RECEIVED NE 'DATA-COMPLETE' THEN
    CALL SR-ABEND.

! ALL APPC COMMUNICATIONS OCCUR IN HALF-DUPLEX MODE. THAT IS, ONLY ONE
! OF THE PROGRAMS CAN TALK WHILE THE OTHER LISTENS. IN ORDER TO
! "TURN THE LINE AROUND" THE RECEIVER MUST WAIT UNTIL THE SENDER SAYS IT'S
! OK TO SEND. THIS IS ACCOMPLISHED BY WAITING FOR 'SEND' TO BE RECEIVED
! IN THE WHAT-RECEIVED SYSTEM VARIABLE.

(B3)RECEIVE-AND-WAIT.
  IF APPCCODE LT 0 OR WHAT-RECEIVED NE 'SEND' THEN
    CALL SR-ABEND.

! EMP-ID-0415 IS DEFINED AS PIC 9(4) USAGE IS DISPLAY. APPC PRESENTATION
! SERVICES DO NOT SUPPORT ZONED DECIMAL DATA TYPE. IN ORDER TO RECEIVE THE
! EMPLOYEE ID FROM THE PC, A WORK RECORD IS CREATED WITH USAGE IS COMP.
```

```
! THEN GET THE RECORD USING AN OBTAIN CALC.

MOVE WK-EMP-ID TO EMP-ID-0415.
OBTAIN CALC EMPLOYEE.

IF DB-REC-NOT-FOUND THEN DO.
(B4)  SEND-ERROR.          ! NOTIFY CLIENT WE DIDN'T FIND EMPLOYEE
      IF APPCCODE LT 0 THEN CALL SR-ABEND.
(B5)  DEALLOCATE.
      LEAVE ADS.
      END.
! GET THE OFFICE RECORD, IF ONE EXISTS.

IF SET OFFICE-EMPLOYEE MEMBER THEN
    OBTAIN OWNER WITHIN OFFICE-EMPLOYEE.

MOVE EMP-FIRST-NAME-0415    TO WK-EMP-FIRST-NAME.
MOVE EMP-LAST-NAME-0415    TO WK-EMP-LAST-NAME.
MOVE EMP-STREET-0415       TO WK-EMP-STREET.
MOVE EMP-CITY-0415         TO WK-EMP-CITY .
MOVE EMP-STATE-0415        TO WK-EMP-STATE.
MOVE EMP-ZIP-FIRST-FIVE-0415 TO WK-EMP-ZIP-FIRST-FIVE.
MOVE STATUS-0415           TO WK-STATUS.
MOVE OFFICE-CODE-0450      TO WK-OFFICE-CODE.

(B6) SEND-DATA WK-EMP-REC2.
     IF APPCCODE LT 0 THEN CALL SR-ABEND.

! NOW THAT WE HAVE FINISHED, LET'S FLUSH THE COMMUNICATIONS
! BUFFER AND TERMINATE THE CONVERSATION.

(B6) DEALLOCATE.
     LEAVE ADS.

!*****
!
! SEND/RECEIVE ERROR HANDLING SUBROUTINE
!
     DEFINE SR-ABEND.

(B7)  DEALLOCATE ABEND.
      LEAVE ADS.
```

SEND/RECEIVE Commands

SEND/RECEIVE commands are listed below. Syntax and syntax rules for each command is presented in alphabetical order after the table.

Command	What it does	When it's issued
ALLOCATE	Begins a conversation with a server dialog	The first communication command issued by the client dialog when it's ready to communicate
CONFIRM	Sends a confirmation request to the remote program and waits for a reply	Issued by the dialog in send state
CONFIRMED	Sends a confirmation reply to the remote program	Issued by the dialog in confirm state
CONTROL SESSION	Acknowledges the conversation and agrees to the rules governing the conversation; a CA extension to APPC used in PC-to-mainframe conversations, but not required for mainframe-to-mainframe communication	Issued by the server dialog in response to the client dialog's ALLOCATE; parameters must match those on the ALLOCATE
DEALLOCATE	Ends the conversation	The last command in the conversation issued by either dialog
PREPARE-TO- RECEIVE	Changes the local side to receive state	Issued by the dialog in send state in response to REQUEST-TO-SEND
RECEIVE-AND-WAIT	Waits for a response from the remote program and receives the data or a value in the system field, WHAT-RECEIVED, upon arrival	Issued by a dialog that wants to receive data (the dialog can be in send or receive state); when used in send state, it indicates that the local dialog wants to receive data, allowing the remote dialog to send data

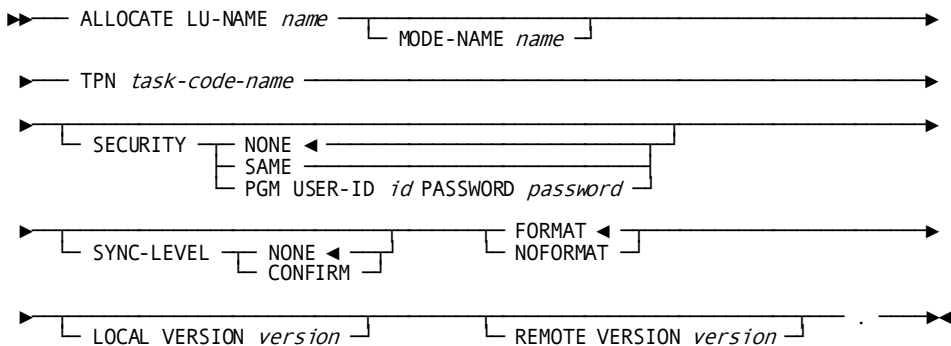
Command	What it does	When it's issued
REQUEST-TO-SEND	Notifies the remote program that the local program is ready to send data	Issued by the dialog in receive state
SEND-DATA	Sends data to the remote program	Issued by the dialog in send state
SEND-ERROR	Notifies the other dialog of an application-level error	Issued by either dialog

ALLOCATE

Purpose

Begins a conversation between an CA ADS dialog and a remote dialog or program. The FORMAT/NOFORMAT setting on the ALLOCATE command must match the FORMAT/NOFORMAT setting on the CONTROL SESSION command.

Syntax



Parameters

LU-NAME *name*

Specifies a field or string that identifies the 1- through 8-character name of the logical unit used by the remote dialog.

Name must match the logical unit name of an APPC line defined to the local CA IDMS/DC system.

MODE-NAME *name*

Specifies the name used by the remote logical unit to select the mode of transmission for the conversation.

Name is either a 1- through 8-character mode name or a variable containing the mode name.

If omitted, CA IDMS/DC uses the mode name defined to the APPC line.

TPN *task-code-name*

A variable or string that contains or specifies the name of the remote program to be initiated by the ALLOCATE command.

If trying to initiate a mainframe CA ADS task, the *task-code-name* must be a 1- through 8-character task code defined to the remote CA IDMS/DC system that invokes ADSORUN1.

SECURITY

Provides security information to the remote program.

NONE

Specifies that no security information is required for the conversation.

NONE is the default for SECURITY.

SAME

Specifies that the signon user ID is passed to the remote program. The following considerations apply:

- This signon does not work if a password is required to sign on to a *separate* CA IDMS/DC or CICS system
- This password does not allow you to sign on to the *same* CA IDMS/DC system (CA IDMS/DC will not support 2 LTEs signed on for the same user at the same time)

PGM USER-ID *id*

Specifies the user ID of the user who runs the application.

id is either a 1- through 32-character user ID or a field containing a user ID.

PASSWORD *password*

Introduces the password of the user who runs the application.

Password is either a 1- through 8-character password or a field containing a password.

This information is used to sign on to the remote logical unit.

SYNC-LEVEL

Introduces the level of synchronization to use for the conversation.

NONE

Specifies that no confirmation commands can be used.

CONFIRM

Specifies that confirmation commands can be used.

FORMAT

Specifies that data will be converted by APPC presentation services before the receiving program sees it:

- Text is converted between ASCII and EBCDIC.
- Numbers are converted between mainframe and PC format.

FORMAT is the default when neither FORMAT or NOFORMAT is specified.

NOFORMAT

Specifies that no data will be converted. If data conversion is required, you must code any data translation or conversion.

LOCAL VERSION *version*

Specifies either a 1- through 32-character local program version identifier or a field containing a version ID sent to the remote program.

REMOTE VERSION *version*

Specifies a variable of at least 32 characters to receive the version identifier sent by the remote program.

Example

In order to allocate a conversation with another CA ADS dialog on a different CA IDMS/DC system, code:

```
ALLOCATE LU-NAME 'S75LU1' TPN 'DLG1' SECURITY NONE  
        SYNC-LEVEL NONE NOFORMAT.
```

S75LU1 is the logical unit name of an APPC line defined to the local CA IDMS/DC system, and DLG1 is the task code that initiates an CA ADS dialog on the remote CA IDMS/DC system. Security and confirmation are not being used, and conversion is not needed between 2 mainframe applications.

CONFIRM

Purpose

Sends a confirmation request to a remote program and waits for a reply.

Syntax

▶▶—— CONFIRM — . ——▶▶

Usage

CONFIRM sends all data in the communications buffers to the remote program.

Considerations

- Before CONFIRM can be issued, the conversation must have a synchronization level of CONFIRM: SYNC-LEVEL CONFIRM on the ALLOCATE command and the issuing dialog must be in the send state.
- **When using CONFIRM to synchronize processing between 2 programs**
 - If the remote program received all data sent, it returns CONFIRMED.
 - If the remote program can't process the data due to an application-level error, it returns SEND-ERROR.
- **To check the status of a conversation**, test the system fields REQUEST-TO-SEND-RECEIVED and WHAT-RECEIVED.

Example

The local program can issue a confirmation or send an error message:

```
SEND-DATA EMPLOYEE-RECORD.  
IF APPCCODE EQ ZERO  
  THEN  
    CONFIRM.  
ELSE  
  DO.  
  DEALLOCATE ABEND.  
  ABORT MSG TEXT 'SEND-DATA ERROR'.  
END.
```

For the corresponding response from the remote program, see CONFIRMED.

CONFIRMED

Purpose

Sends a confirmation reply to the remote dialog.

Syntax

►—— CONFIRMED — . ——►

Usage

CONFIRMED sends all data in the communications buffers to the remote program.

Note: CONFIRMED is sent in response to CONFIRM only.

Considerations

- Before CONFIRMED can be issued, the conversation must have a synchronization level of CONFIRM: SYNC-LEVEL CONFIRM on the ALLOCATE command.
- **When using CONFIRM to synchronize processing between 2 programs**
 - If the local dialog received all data sent, it returns CONFIRMED.
 - If the local dialog can't process the data due to an application-level error, it returns SEND-ERROR.
- **To check the status of a conversation**

Test the system fields: APPCCODE and WHAT-RECEIVED.

Example

In response to the preceding CONFIRM example, the remote program would issue:

```
RECEIVE-AND-WAIT EMPLOYEE-RECORD.  
IF APPCCODE EQ ZERO  
  THEN  
  DO.  
  RECEIVE-AND-WAIT.  
  IF APPCCODE EQ ZERO AND WHAT-RECEIVED EQ 'CONFIRM'  
    THEN  
      CONFIRMED.  
  END.
```


Considerations

The FORMAT/NOFORMAT setting on the ALLOCATE command must match the FORMAT/NOFORMAT setting on the CONTROL SESSION command.

Example

In response to an ALLOCATE with the FORMAT setting, the local dialog sends LEVEL 1 in its local version to the remote dialog for testing. The remote dialog will receive it in the field, REMOTE-FIELD, for testing.

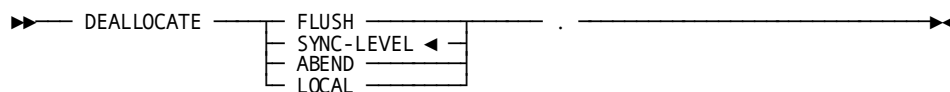
```
CONTROL SESSION FORMAT LOCAL VERSION 'LEVEL 1'
  REMOTE VERSION REMOTE-FIELD.
IF APPCCODE NE ZERO
  THEN
  DO.
  DEALLOCATE ABEND.
  ABORT MSG TEXT 'CONTROL SESSION ERROR'.
  END.
IF REMOTE-FIELD NE 'FIRST RELEASE'
  THEN
  DO.
  DEALLOCATE ABEND.
  ABORT MSG TEXT 'WRONG RELEASE OF PARTNER DIALOG'.
  END.
```

DEALLOCATE

Purpose

Ends the conversation.

Syntax



Parameters

FLUSH

Sends all data in the communications buffer to the remote program and then terminates the conversation normally.

SYNC-LEVEL

Terminates the conversation based on the synchronization level. If the current synchronization level is:

- **NONE**— All data in the communications buffer is sent to the remote dialog and the conversation terminates normally.
- **CONFIRM**— All data in the communications buffer is sent to the remote dialog along with a request for confirmation. Once confirmation arrives from the remote program, the conversation terminates normally. Otherwise, the return code determines the state of the conversation.

SYNC-LEVEL is the default for DEALLOCATE

ABEND

Terminates the conversation abnormally.

The following considerations apply:

- If the dialog is in **send** state, all data in the buffer is sent to the remote program
- If the dialog is in **receive** state, all data in the buffer is purged

LOCAL

Terminates the conversation after the remote dialog has deallocated the conversation.

Usage

DEALLOCATE is the last command in the conversation, but each dialog can continue independent processing. The command only ends the conversation.

Considerations

DEALLOCATE:

- Is not a control command
- Does not end the dialog

Example

```
IF APPCCODE NE ZERO
  THEN
  DO.
  DEALLOCATE ABEND.
  ABORT MSG TEXT 'PROBLEM IN PROCESSING'.
  END.
```

PREPARE-TO-RECEIVE

Purpose

Prepares to receive data from a remote program.

Syntax

```

▶▶ PREPARE-TO-RECEIVE [ SYNC-LEVEL
                        FLUSH ] .
  
```

Parameters

SYNC-LEVEL

Processes data based on the current synchronization level. If the synchronization level is:

- **NONE**— All data in the communications buffer is sent to the remote program before the local dialog is put in receive state.
- **CONFIRM**— All data in the communications buffer is sent to the remote program along with a request for confirmation. If the remote dialog issues CONFIRMED, the local dialog is put in receive state. Otherwise, the return code from the remote program determines the state of the conversation.

SYNC-LEVEL is the default for PREPARE-TO-RECEIVE

FLUSH

Sends all data in the communications buffer to the remote program.

Usage

PREPARE-TO-RECEIVE is issued by the dialog in send state in response to REQUEST-TO-SEND. PREPARE-TO-RECEIVE switches the local dialog to receive state.

Considerations

To check the status of a conversation test the system fields: REQUEST-TO-SEND and WHAT-RECEIVED.

RECEIVE-AND-WAIT

Purpose

Names the record or field to receive data.

Syntax

```

▶▶ RECEIVE-AND-WAIT [ variable-name ] .
  
```

Parameters

variable-name

Names the record or field to receive data from the remote program.

Variable-name can be up to 32 characters long.

If you omit the *variable-name*, data received from the remote program is lost. Only a value is received in the WHAT-RECEIVED field.

Usage

Considerations

RECEIVE-AND-WAIT is issued by the dialog in receive state:

- **If presentation services are being used**, then the variable's definition must match the description of the incoming data definition.
- **If unformatted data is being received**, the maximum length received is derived from the variable's definition.

Example

This server dialog receives an employee id from the client and then obtains the appropriate employee records from the database. The first RECEIVE-AND-WAIT receives the id. The second turns the line around prior to sending information to the client.

```
CONTROL SESSION FORMAT.  
IF APPCCODE LT ZERO  
    THEN CALL SR-ABEND.  
RECEIVE-AND-WAIT EMP-ID-WORK.  
IF APPCCODE LT 0 OR WHAT-RECEIVED NE 'DATA-COMPLETE'  
    THEN CALL SR-ABEND.  
RECEIVE-AND-WAIT.  
.  
.  
.
```


REQUEST-TO-SEND

Purpose

Notifies the remote dialog that the CA ADS application is ready to send data. The remote dialog can respond to this command in either of the following ways:

- Ignoring the request and continuing to send data
- Granting the request and starting to receive data

To determine whether permission to send has been granted, check the system field (WHAT-RECEIVED).

Syntax

▶▶ REQUEST-TO-SEND .

SEND-DATA

Purpose

Sends data to the remote dialog.

Syntax

▶▶ SEND-DATA *variable-name* .

Parameters

variable-name

Specifies the name of the record or element to be sent.

Variable-name can be up to 32 characters long.

Note: The definition of *variable-name* must be in the dictionary associated with the local dialog.

Usage

Data to be sent is stored in the CA ADS communications buffer. When the buffer is full or you issue a CONFIRM command or DEALLOCATE with the FLUSH option, the data in the buffer is sent to the remote dialog.

Considerations

CA ADS issues a compiler error:

- If the record or field transmitted is not a valid data type (these data types are not valid for SEND/RECEIVE):
 - Pointer
 - Multibit binary
 - Zoned decimal
 - Graphics (kanji)
- If the record or field transmitted is coded:
 - With an OCCURS DEPENDING ON clause
 - As USAGE COMP with decimal positions
- If the record is a logical record
- If *variable-name* is a level 88
- If a group contains more than 256 fields
- If a record definition makes an FD longer than 32,768 bytes

Any field that redefines another field is ignored, but a transmitted field can be a REDEFINES field. For example, FIELD-B redefines FIELD-A within RECORD1. If you send RECORD1, FIELD-B is ignored and the record is sent as if FIELD-A is the current definition of the field.

Because only a subset of IDD data types are supported, you should:

1. Create a work record (much like the map work record) of the proper data types.
2. Move the values from your database record to this work record.
3. Move the received values back to your database record.

Example

After retrieving data and building the work record, send the data to the client dialog.

```
SEND-DATA WK-EMP-REC2.  
IF APPCCODE LT 0 THEN CALL SR-ABEND.
```

SEND-ERROR

Purpose

Informs the remote program that CA ADS detected an application-level error (such as DB-REC-NOT-FOUND).

Syntax

► SEND-ERROR . ◄

Usage

When you issue SEND-ERROR, the remote program does not automatically send the data again. The remote program must detect the SEND-ERROR and respond appropriately.

Example

If the record is not found, notify the client.

```
IF DB-REC-NOT-FOUND THEN DO.
    SEND-ERROR.
    IF APPCCODE LT 0 THEN CALL SR-ABEND.
    DEALLOCATE.
    LEAVE ADS.
END.
```

Design Guidelines

- **Be careful not to end a conversation inadvertently.**

A conversation is a task-level resource in CA IDMS/DC. When the task ends, any ongoing conversation will be deallocated automatically. So do not use:

- DISPLAY
- LEAVE ADS NEXT TASK CODE
- LINK (to a program that will pseudoconverse)

Using any of these commands will deallocate your conversation.

- **Be aware of the location of the allocated dialog.**

The CA ADS allocated task runs as a nonterminal task. Because you can not point to a secondary load area or load library on the ALLOCATE command, the allocated dialog should reside in the CA IDMS/DC default load search sequence.

If you want to use a secondary load area or load library, you must override the search sequence by:

- Using the SECURITY parameter on the ALLOCATE command issued by the primary dialog.
- Dialogs which exist in secondary load areas can be accessed in the client task thread by using a signon profile associated with that user containing DCUF SET DICTNAME or LOADLIST to change the search sequence for the secondary dialog.

For server task threads a new system loadlist must be created and the secondary dictionary entry must be coded prior to the primary dictionary entry. A typical loadlist follows:

```
ADD LOADLIST NEWLOAD
    DICTNAME IS APPCDICT VERSION IS 1
    DICTNAME IS USER-DEFAULT VERSION IS USER-DEFAULT
    DICTNAME IS SYSTEM-DEFAULT VERSION IS SYSTEM-DEFAULT
    LOADLIB IS USER-DEFAULT
    DICTNAME IS USER-DEFAULT VERSION IS 1
    DICTNAME IS SYSTEM-DEFAULT VERSION IS 1
    LOADLIB IS SYSTEM-DEFAULT
MOD SYSTEM nnn
    LOADLIST = NEWLOAD
```

Understanding Conversation States

You must be aware of conversation states when you are developing a cooperative application. Whenever a dialog is involved in a conversation, it is in a specific conversation state. The state determines which communication operations can be performed at that time. For example, a dialog must be in send state to send data; and the partner dialog must be in receive state to receive data. You must keep both dialogs synchronized to allow information to be exchanged.

Valid Conversation States

State	What it means
Reset	No conversation exists
Send	The dialog can send data, request confirmation, or deallocate the conversation
Receive	The dialog can receive information from its partner dialog.
Confirm	The dialog can reply to a confirmation request (there are three types of confirm state, based on the state of the dialog after a communication command is issued)
Deallocate	The dialog can deallocate the conversation

Confirmation

The use of confirmation is optional. The primary program can request confirmation at the beginning of the session. If confirmation is used, the sending program can issue a CONFIRM command to which the recipient must respond CONFIRMED or SEND-ERROR. You can acknowledge the receipt of data programmatically if you prefer.

Conversation States

Statements and Conversation States

The following table summarizes, for each communication command, the states in which the command can be issued and the resulting state after the command is executed.

To issue this command	The dialog must be in this state	After this return code	The dialog is in this state	
ALLOCATE	Reset	OK	Send	
		Other	Reset	
CONFIRM	Send	OK	Send	
		PROG-ERROR	Receive	
		Other	Deallocate	
CONFIRMED	ConfirmR	OK	Receive	
		Other	Deallocate	
		ConfirmS	OK	Send
			Other	Deallocate
CONFIRMED	ConfirmD	Any	Deallocate	
CONTROL	Receive	OK	Receive	
SESSION		Other	Deallocate	
DEALLOCATE				
FLUSH	Send	Any	Reset	
CONFIRM	Send	OK	Reset	
		PROG-ERROR	Receive	
		Other	Reset	
ABEND	Any	Any	Reset	
LOCAL	Deallocate	Any	Reset	
PREPARE-TO-RECEIVE	Send	OK or PROG-ERROR	Receive	
		Other	Deallocate	
RECEIVE-AND-	Send or receive	OK, Data complete	Receive	
WAIT		OK, Send	Send	
		PROG-ERROR	Receive	

To issue this command	The dialog must be in this state	After this return code	The dialog is in this state
		Other	Deallocate
		OK, CONFIRM	ConfirmR
		OK, CONFIRM-SEND	ConfirmS
		OK, CONFIRM-	
		DEALLOCATE	ConfirmD
REQUEST-TO-SEND	Receive	OK	Receive
		Other	Deallocate
SEND-DATA	Send	OK	Send
		PROG-ERROR	Receive
		Other	Deallocate
SEND-ERROR	Send or Receive	OK	Send
		PROG-ERROR	Receive
		Other	Deallocate
	ConfirmR	OK	Send
	ConfirmS	PROG-ERROR	N/A
	ConfirmD	Other	Deallocate

Conversation States in a Successful Data Transfer

In the following diagram, for example, Dialog A on the PC establishes a conversation with Dialog B on the mainframe. Dialog A sends a request for employee information to Dialog B. Dialog B processes the request and returns a reply. Matching data is found and returned. This is the same application shown in the flowchart earlier in this chapter.

The state changes are noted under the communications commands. Refer back to the previous flowchart and the sample code if you wish. You can see that Dialog B uses a RECEIVE-AND-WAIT to switch the line (changing from receive to send state in preparation for returning data to the PC). Also note the state changes necessary to deallocate the conversation. (Here, WR represents the WHAT-RECEIVED system field.)

A Successful Transaction

Dialog A	Dialog B
(A1) ALLOCATE State (reset to send)	(B1) CONTROL SESSION State (receive)
(A2) SEND-DATA Emp-id State (send)	(B2) RECEIVE-AND-WAIT Emp-id State (receive) WR=DATA-COMPLETE
(A3) RECEIVE-AND-WAIT Emp-data State (send to receive)	(B3) RECEIVE-AND-WAIT State (receive to send) WR=SEND
(A6) RECEIVE-AND-WAIT. State (receive to deallocate)	(B5) SEND-DATA Emp-data State (send)
(A7) DEALLOCATE State (deallocate to reset)	(B6) DEALLOCATE State (send to reset)

In this transaction:

- (A1) initiates conversation with ALLOCATE.
- (B1) acknowledges conversation with CONTROL SESSION.
- (B2) prepares to receive a request with RECEIVE-AND-WAIT.
- (A2) issues SEND-DATA with the employee id as a parameter.
- (B3) prepares to send a reply with RECEIVE-AND-WAIT. This command switches the line. Dialog B changes state from receive to send.
- (A3) prepares to receive the employee data.
- (B5) returns employee data to Dialog A with SEND-DATA. Dialog A tests for the PROG-ERROR condition, but does not find it. Dialog A checks that the data is complete (APPCCODE=OK and WR=DATA-COMPLETE).
- (A6) once the data is successfully received, RECEIVE-AND-WAIT prepares to deallocate resources verifying that Dialog B (currently in control) is ready to end the conversation.
- (B6) flushes the communications buffer and terminates the conversation with DEALLOCATE.
- (A7) releases local resources with DEALLOCATE LOCAL.

Testing APPC Status Codes and System Fields

You can test the values of these codes and fields in your dialogs to determine how information will be processed. For example, you can refer back to the sample code to see the use of APPCCODE in the server dialog.

Status Codes

These codes report the status of each communications command performed:

- **APPCCODE** provides the category of the message returned from the communications services for the most recent command executed.
- **APPCERC** contains more detailed information about the message returned by APPCCODE.

System Fields

These system fields track information received from the remote program:

- **WHAT-RECEIVED** tells you what was received from the remote program.
- **REQUEST-TO-SEND-RECEIVED** tells you whether or not the remote program is requesting to send data.

When APPC Status Codes and System Field Values are Returned

These are the status codes and system fields returned by the communications commands.

Command	APPCCODE	APPCERC	RECEIVED	RECEIVED
ALLOCATE		●		
CONFIRM	●	●		●
CONFIRMED	●	●		
DEALLOCATE	●	●		
PREPARE-TO-RECEIVE	●	●		
RECEIVE-AND- WAIT	●	●	●	
REQUEST-TO- SEND	●	●		

Command	APPCCODE	APPCERC	RECEIVED	RECEIVED
SEND-DATA	●	●		●
SEND-ERROR	●	●		●

Keep in mind:

- Status codes are updated after each communications command executes.
- A condition can be reported when the communications command that caused the error executes or when a subsequent communications command executes.

APPCCODE and APPCERC

If an error description says *internal error*, request technical support from your site. If your technical support staff cannot remedy the problem, make sure they have the APPCCODE and APPCERC before they call CA Technical Support.

The following table displays an overview of the APPCCODEs (detailed descriptions of the APPCCODEs follow after this table):

APPCCODE	Status
0	OK
-1	Parameter check
-3	Allocation error
-4	Resource failure
-5	Deallocate condition
-6	Program error
-7	SVC error
-8	State error
-9	Unsuccessful
-10	Control session error
-11	Format descriptor error
-12	Send-data error
-13	Receive format error

0: OK

The communications command executed successfully. Control has returned to your CA ADS application. The current state of the conversation depends on the specific communications command you issued.

APPCERC	What it means
0	Data is available for the dialog to receive
1	Information other than data is available for the dialog to receive

-1: Parameter Check

There is a coding error in either the CA ADS application or the remote dialog that must be corrected. The syntax is correct, but there is a mismatch of parameters passed between the two dialogs or the parameters supplied are invalid.

APPCERC	What it means
0	Internal error
10	You issued a CONFIRM command when the conversation was allocated with a synchronization level of NONE
30	Internal error
31	Internal error
32	Internal error
33	Internal error
34	Internal error
35	Internal error
36	Internal error

-3: Allocation Error

The specified conversation cannot be allocated.

APPCERC	What it means
0	Internal error
1	The conversation cannot be allocated because of a condition that is not temporary (for example, a session protocol error). Do not retry the allocation request until the condition is corrected.

APPCERC	What it means
2	The conversation cannot be allocated because of a condition that can be temporary (for example, the secondary application is not available). If the condition is temporary, you can retry the allocation request.
3	The remote program rejected the allocation request because it did not understand the TPN. The TPN must be a task code associated with the dialog and defined to CA IDMS/DC if you are trying to allocate a CA ADS task.
4	The <i>task-code-name</i> specified on the ALLOCATE command exists but cannot be started. This is not a temporary condition and must be resolved by a systems programmer. Do not retry the ALLOCATE until the situation is corrected.
5	The <i>task-code-name</i> specified on the ALLOCATE command exists but cannot be started. This is a temporary condition. You can retry the allocation request.
6	The user specified on the SECURITY parameter of the ALLOCATE command is not known to the remote program.
7	Internal error.
8	Internal error.

-4: Resource Failure

A resource failure terminated the conversation prematurely.

APPCERC	What it means
1	The resource failure is not temporary (for example, a session protocol error). Do not retry the transaction until the condition is corrected.
2	The resource failure can be temporary (for example, a power outage, a line failure, or a problem with a modem). You can retry the transaction.

-5: Deallocate Condition

The remote program issued a DEALLOCATE command.

APPCERC	What it means
0	The deallocation was normal.
1	The remote program specified the ABEND option on the DEALLOCATE command or the remote program has abended. Any data remaining in the CA ADS communications buffer is purged.

-6: Program Error

The remote program issued the SEND-ERROR command. There is an error in the local application that must be corrected.

-7: SVC Error

The remote program issued the SEND-ERROR command. There is an error in the local application that must be corrected.

-8: State Error

There is a coding error in your CA ADS application. The CA ADS side of the conversation was not in the correct state to execute the communications command you specified: for example, you tried to issue SEND-DATA while in receive state.

In some cases, you can need to issue a DEALLOCATE ABEND to recover from this error.

-9: Unsuccessful

The conversation command was unsuccessful.

-10: Control Session Error

The CA ADS side of the conversation issued an ALLOCATE command correctly. But the remote program did one of the following:

- Omitted the CONTROL SESSION command
- Transmitted a CONTROL SESSION command after another communications command
- Transmitted a CONTROL SESSION command whose parameters do not agree with the ALLOCATE command

This code is returned by the ALLOCATE command. This code can indicate an internal error.

-11: Format Descriptor Error

The CA ADS side of the conversation received an internal error from presentation services about the format descriptors.

-12: Send-Data Error

The CA ADS side of the conversation detected an internal error or a conversion error. This code is reported by the SEND DATA command.

-13: Receive Format Error

CA ADS received an error in a formatted conversation. This indicates an internal error. This code is reported by the RECEIVE-AND-WAIT command.

System Fields

WHAT-RECEIVED

This variable tells you what was received from the remote program. It is updated after the RECEIVE-AND-WAIT command is executed.

Contents	Meaning
DATA-COMPLETE	Data was received successfully.
CONFIRM	The remote dialog issued a CONFIRM command and expects the local dialog to reply with the CONFIRMED command.
CONFIRM-SEND	The remote dialog issued a PREPARE-TO-RECEIVE command with the CONFIRM option. The local dialog can reply with either a CONFIRMED or a SEND-ERROR command.
CONFIRM-DEALLOCATE	The remote dialog issued a DEALLOCATE command with the CONFIRM option. The local dialog can reply with either a CONFIRMED or a SEND-ERROR command.
SEND	The remote program is in receive state and the local dialog is now in send state. The local dialog can now issue a SEND-DATA command.

REQUEST-TO-SEND- RECEIVED

This variable tells you whether or not the remote dialog issued the REQUEST-TO-SEND command. This variable is updated after the CONFIRM or the SEND-DATA command executes.

Contents	Meaning
0	The remote program has not requested to send data
1	The remote program is requesting to send data

If the local dialog receives REQUEST-TO-SEND then REQUEST-TO-SEND-RECEIVED is set to 1.

The local dialog resets REQUEST-TO-SEND-RECEIVED to 0 after every CONFIRM, SEND-DATA, and SEND-ERROR command.

Chapter 22: OSCaR Commands

This section contains the following topics:

[OSCaR Command Syntax](#) (see page 559)

[Sample OSCaR Application](#) (see page 563)

[OSCaR to APPC Mapping](#) (see page 565)

OSCaR Command Syntax

Purpose

OPEN, SEND, CLOSE, and RECEIVE (OSCaR) commands are an interface between mainframe CA ADS dialogs. OSCaR commands run as APPC commands; that is, as LU6.2 between mainframes. If a mainframe application is accessing a remote data base rather than a remote application, DDS should be more efficient.

Concept

OSCaR commands are much simpler than the CA IDMS SEND/RECEIVE verb set for APPC cooperative processing in that they provide only a subset of the complete APPC functionality and synchronization of conversation states is automatic. It is not necessary to understand the CA IDMS SEND/RECEIVE verb set, the IBM APPC verb set, or Conversation States before using OSCaR commands. However, it is necessary to understand basic cooperative processing concepts.

Coding Considerations

- Only four commands are defined: OPEN, SEND, CLOSE, and RECEIVE
- APPC and OSCaR commands are mutually exclusive within a single dialog:
 - APPC verbs such as ALLOCATE, CONTROL SESSION, and SEND-DATA are not allowed in dialogs containing OSCaR verbs
 - The four OSCaR verbs are not allowed in dialogs containing APPC verbs
- APPC data areas WHAT-RECEIVED and REQUEST-TO-SEND-RECEIVED cannot be referenced in a dialog that contains OSCaR commands
- OSCaR has no parameters equivalent to the FORMAT or SYNC-LEVEL parameters on the APPC ALLOCATE command

- OSCaR verbs always run as NOFORMAT
- Confirmation of user-validated data content must be sent via a user-defined control record rather than as a separate CONFIRM or SEND-ERROR command
- Commands to perform synchronization of conversation states, (RECEIVE-AND-WAIT, PREPARE-TO-RECEIVE, and REQUEST-TO-SEND) are done automatically by the runtime system when needed. These commands are needed only when a command is issued and the line is in the wrong state.

Error conditions can be detected with autostatus or by examining ERROR-STATUS for 6901 or APPCCODE for a negative value.

More information:

[Cooperative Processing Commands](#) (see page 527)

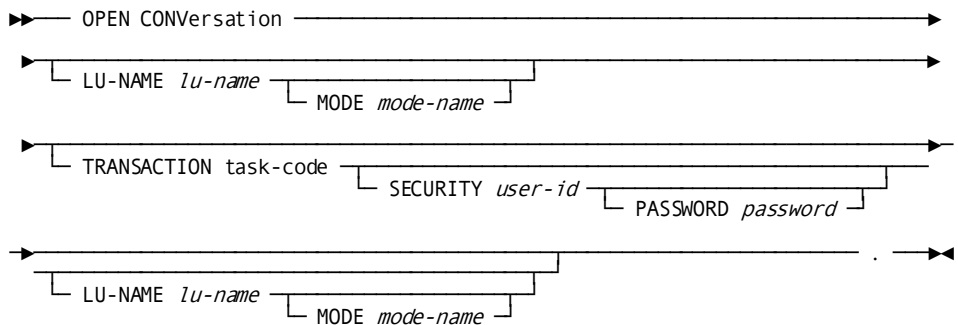
[Understanding Conversation States](#) (see page 548)

OPEN

Purpose

Establishes a conversation.

Syntax



Parameters

TRANSACTION *task-code*

Indicates that the dialog runs as a client; *task-code* names the task to be invoked on the remote logical unit.

If *task-code* names a mainframe CA ADS dialog, it must be defined on the remote logical unit as invoking program ADSORUN1.

SECURITY *user-id*

Specifies the identifier of user to be signed on to the remote logical unit.

If SECURITY is not specified, signon is performed with no user identifier.

PASSWORD *password*

Specifies the password associated with *user-id* during signon to the remote logical unit. If PASSWORD is not specified, signon is performed with no password.

LU-NAME *lu-name*

Specifies a field or string that identifies the 1- through 8-character name of the logical unit used by the remote dialog.

Lu-name must match the logical unit name of an APPC line defined to the local CA IDMS/DC system.

If LU-NAME is not specified, *lu-name* is the value of the ADSSLUNAM attribute for the user session.

MODE *mode-name*

Specifies the name used by the remote logical unit to select the mode of transmission for the conversation.

Mode-name is either a 1- through 8-character mode name or a variable containing the mode name.

If MODE is not specified, *mode-name* is the value of the ADSMODE attribute for the user session. If there is no ADSMODE attribute, the value is the default mode name defined by the DLOGMOD parameter within the VTAM definition. If there is no default mode in the VTAM definition, the dialog aborts.

Usage*Placement of OPEN CONVERSATION*

OPEN must be the first APPC command encountered in an OSCaR dialog.

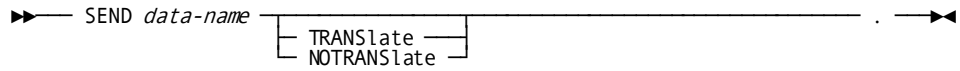
OPEN CONVERSATION with No Parameters

If no parameters are specified on OPEN CONVERSATION, the dialog runs as a server.

SEND**Purpose**

Sends a data name to a remote logical unit.

Syntax



Parameters

data-name

Identifies the record or element name associated with the dialog to send to the remote logical unit.

TRANSLate

Not meaningful in an CA ADS mainframe dialog.

NOTRANSLate

Not meaningful in an CA ADS mainframe dialog.

Usage

Restrictions on Data Name

In a SEND command, *data-name* must not be or correspond to the name of:

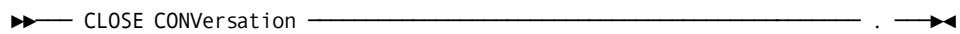
- A logical record
- A built-in function
- An element defined as pointer, multi-bit binary, or graphic
- An 88-level element
- A reserved word, such as SPACES, DATE, CURSOR-ROW
- A quoted literal

CLOSE

Purpose

Ends a conversation.

Syntax



Usage

Placement of CLOSE CONVERSATION

CLOSE CONVERSATION must be the last APPC command encountered in both client and server dialogs that use OSCaR commands.

RECEIVE

Purpose

Receives a data name from a remote logical unit.

Syntax

```

▶▶ RECEIVE data-name [ TRANSlate | NOTTRANSlate ] . ▶▶

```

Parameters

data-name

Identifies the record or element name associated with the dialog to receive from the remote logical unit.

TRANSlate

Not meaningful in an CA ADS mainframe dialog.

NOTTRANSlate

Not meaningful in an CA ADS mainframe dialog.

Usage

Restrictions on Data Name

In a RECEIVE command, *data-name* must not be or correspond to the name of:

- A logical record
- A built-in function
- An element defined as pointer, multi-bit binary, or graphic
- An 88-level element
- A reserved word, such as SPACES, DATE, CURSOR-ROW
- A quoted literal

Sample OSCaR Application

About this Example

This sample application program retrieves EMPLOYEE/OFFICE data from a remote demo data base. It performs the same function as the example in [Sample Cooperative Application](#) (see page 528). No intermediate records are needed because the OSCaR verbs support all data types found in the EMPLOYEE and OFFICE records.

Client Map

RETRIEVE EMPLOYEE DATA

EMPLOYEE ID: _____

Employee name:

Office street:

Office city :

Enter any employee ID.

The employee's name and office address will be returned.

Client ENTER Process

```
IF EMP-ID-0415 EQ ZEROES DO.                                !IF NO EMP-ID ENTERED
  INITIALIZE RECORDS (EMPLOYEE, OFFICE).                    ! CLEAR OLD DATA
  DISPLAY MESSAGE TEXT 'ENTER AN EMPLOYEE ID'.! REQUEST EMP-ID
END.

IF FIELD EMP-ID-0415 IS CHANGED DO.                          !IF EMP-ID WAS ENTERED
  OPEN CONVERSATION TRANSACTION 'EMPSERVE'.
  SEND EMP-ID-0415.                                          ! SEND EMP-ID TO EMPSERV
  RECEIVE EMPLOYEE.                                         ! RETRIEVE EMPLOYEE
  RECEIVE OFFICE.                                           ! RETRIEVE OFFICE
  CLOSE CONVERSATION.
END.

IF EMP-NAME-0415 EQ ALL '*'                                  !DISPLAY RESULTS
  DISPLAY MESSAGE TEXT 'EMPLOYEE DOES NOT EXIST'.
ELSE
  DISPLAY MESSAGE TEXT 'EMPLOYEE DISPLAY IS COMPLETE'.
```

Server PREMAP Process

```

!***** GET EMP-ID FROM DIALOG EMPCLIEN *****
OPEN CONVERSATION TRANSACTION 'EMPSERVE'.
RECEIVE EMP-ID-0415.

!***** GET EMPLOYEE/OFFICE DATA *****
OBTAIN CALC EMPLOYEE.
IF DB-STATUS-OK DO.
  IF SET OFFICE-EMPLOYEE MEMBER
    OBTAIN OWNER WITHIN OFFICE-EMPLOYEE.
  ELSE
    MOVE ALL '*' TO OFFICE-ADDRESS-0450.
  END.
ELSE DO.
  MOVE ALL '*' TO EMP-NAME-0415.
  !MIGHT INITIALIZE ALL EMPLOYEE FIELDS
  !EXCEPT EMP-NAME-0415 AND EMP-IF-0415.
  END.

!***** RETURN RECORDS TO CLIENT *****
SEND EMPLOYEE.
SEND OFFICE.
CLOSE CONVERSATION.
LEAVE ADS.

```

OSCaR to APPC Mapping

The following table outlines the conversions which can be used to map OSCaR commands to standard APPC commands. This will allow any CA IDMS/ADS dialog using the OSCaR verb set to communicate with any other program using the standard LU6.2/APPC verb set.

Command	Present State	APPC Result	ADS Equivalent	#TREQ Assembler Equivalent
OPEN *With LU-NAME or ADSLUNAM defined	Reset	ALLOCATE	Allocate...	#TREQ Alloc...
LU-NAME and ADSLUNAM not defined	Reset	GET_ATTRIBUTES	Control-Session no format	#TREQ UIOCB...

Command	Present State	APPC Result	ADS Equivalent	#TREQ Assembler Equivalent
SEND	Send	SEND_DATA	Send-Data	TREQ Put...
	Receive	REQUEST_TO_SEND SEND_DATA	Request-To-Send Send-Data	#TREQ Put optns= signal #Treq Put...
CLOSE	Deallocate	DEALLOCATE_LOCAL	Deallocate Local	#TREQ Get... (receive deallocate)
	Send	DEALLOCATE_SYNC_LEVEL	Deallocate Sync-Level	#TREQ Put optns= last
RECEIVE	Receive	RECEIVE_AND_WAIT	Receive- And-Wait	#TREQ Get...
	Send	PREPARE_TO_RECEIVE RECEIVE_AND_WAIT	Prepare-To-Receive Receive- And-Wait	#TREQ Put optns= invite #TREQ Get...

*ADSLUNAM is a user-defined User-Profile attribute. This can be used to define a default LU name for OPEN commands. Additionally, ADSMODE may be used to define a default MODE name for the OSCaR OPEN command. For more information about user profiles, see the *CA IDMS Security Administration Guide*.

Appendix A: System Records

This section contains the following topics:

[Overview](#) (see page 567)

[ADSO-APPLICATION-GLOBAL-RECORD](#) (see page 568)

[ADSO-APPLICATION-MENU-RECORD](#) (see page 579)

Overview

CA ADS provides the three system records, listed in the table below. This appendix describes ADSO-APPLICATION-GLOBAL-RECORD and ADSO-APPLICATION-MENU-RECORD.

CA ADS System Records

System record	Purpose
ADSO-APPLICATION-GLOBAL-RECORD	Passes information between dialogs in an application
ADSO-APPLICATION-MENU-RECORD	Holds information for the runtime system to use in building menus.
ADSO-STAT-DEF-REC	Associates level-88 condition names with status codes during dialog compilation

ADSO-APPLICATION-GLOBAL-RECORD and ADSO-APPLICATION-MENU-RECORD have fully addressable fields. At runtime, information supplied during application definition is moved to the applicable fields in the system records.

When the fields of a system record are referenced by a dialog, the record must be associated with the dialog as a work or map record. In applications not defined using the application compiler, system records are treated like any other work or map records.

The CA ADS system records ADSO-APPLICATION-GLOBAL-RECORD and ADSO-APPLICATION-MENU-RECORD are discussed separately below.

More information:

[Error Handling](#) (see page 277)

ADSO-APPLICATION-GLOBAL-RECORD

ADSO-APPLICATION-GLOBAL-RECORD is automatically associated with an application as a global record, unless the record is explicitly deselected on the Global Records screen while defining the application with the application compiler (ADSA). If ADSO-APPLICATION-GLOBAL-RECORD is deselected, on the Global Records screen, the runtime system does not supply runtime information to the application's dialogs, and dialogs cannot modify runtime flow of control by changing AGR-CURRENT-RESPONSE.

The CA ADS runtime system uses ADSO-APPLICATION- GLOBAL-RECORD in the following ways:

- To pass information about the current application to dialogs in the application
- To allow dialogs to modify the application flow of control (by modifying AGR-CURRENT-RESPONSE)
- To provide an additional means of passing information between dialogs (by assigning values to the AGR-PASSED-DATA and AGR-MESSAGE fields)

Field Descriptions

AGR-APPLICATION-NAME

Contains the name of the current application, as specified on the Main Menu during application definition.

This field is updated once at the beginning of the application.

AGR-CURRENT-FUNCTION

Contains the name of the current function.

This field is updated at the beginning of each function.

AGR-NEXT-FUNCTION

Contains the name of the next function to be executed. The next function is the function initiated by the response contained in the AGR-CURRENT-RESPONSE field.

This field is updated on mapin from the terminal.

If a process command modifies AGR-CURRENT-RESPONSE to change the flow of control, AGR-NEXT-FUNCTION does not have to be changed. On an EXECUTE NEXT FUNCTION command, the runtime system transfers control to the function associated with the response.

AGR-CURRENT-RESPONSE

Contains the name of the next response to be executed, as specified by the user.

This field is updated on each mapin from the terminal.

The runtime system executes the response in AGR-CURRENT- RESPONSE when it encounters an EXECUTE NEXT FUNCTION command. The value in

AGR-CURRENT-RESPONSE can be overwritten by the premap or response process of a dialog function.

Note that if AGR-CURRENT-RESPONSE is modified by a process command, the runtime system does not perform security checking.

AGR-DEFAULT-RESPONSE

Contains the name of the default response (if any) for the current function, as specified on any Function Definition screen with ADSA during application definition.

This field is updated at the beginning of each function. If the function has no default response, AGR-DEFAULT-RESPONSE contains blanks.

AGR-TASK-CODE

Contains the task code entered by the user to initiate the application.

This field is updated once at the beginning of the application.

AGR-EXIT-DIALOG

Contains the name of the user exit dialog (if any) associated with the current function, as specified on the Function Definition screen during application definition.

This field is updated at the beginning of each function, and is blank if the function has no user exit dialog.

AGR-PRINT-DESTINATION

Contains the default print destination for the application, as specified on the Main Menu during application definition.

AGR-PRINT-DESTINATION can be specified in a WRITE PRINTER command to specify a print destination.

This field is updated once at the beginning of the application and is blank if the application has no default print destination.

AGR-DATE

Contains the current date in the format specified by the application developer on the Main Menu during application definition.

This field is updated at the beginning of each premap and response process, and each menu and menu/dialog function.

AGR-USER-ID

Contains the user id passed to ADSO-APPLICATION-GLOBAL- RECORD from the AMR-USER-ID field of ADSO- APPLICATION-MENU-RECORD.

This field is updated after a successful user signon to the application and is blank if signon is unsuccessful or is not performed.

AGR-SECURITY-CODE

Contains the security class associated with the user id, as returned by CA IDMS/DC or DC/UCF following successful execution of a system SIGNON function.

AGR-SECURITY-CODE is treated as a 256-bit field, with each bit representing a security class, from 0 to 255. A bit is set to 1 if the user is authorized at that security class, and is set to 0 if the user is not.

AGR-INSTALLATION-CODE

Contains the installation-defined security code associated with the user id, as returned by CA IDMS/DC or DC/UCF following successful execution of a system SIGNON function.

AGR-PASSED-DATA

A group field that consists of the following elements:

AGR-PASSED-ONE

Contains data passed to ADSO-APPLICATION-GLOBAL-RECORD from the AMR-PASSING field of ADSO-APPLICATION-MENU- RECORD on mapin from the terminal.

Note that if the user does not enter data in AMR- PASSING, AGR-PASSED-ONE is not updated.

AGR-PASSED-TWO

A 32-byte field that the application developer can use as applicable.

AGR-PASSED-THREE

A 32-byte field that the application developer can use as applicable.

AGR-PASSED-FOUR

A 32-byte field that the application developer can use as applicable.

The runtime system never updates fields AGR-PASSED-TWO, AGR-PASSED-THREE, and AGR-PASSED-FOUR.

AGR-APPLICATION-VERSION

Contains the version number of the current application, as specified by the application developer on the Main Menu during application definition.

This field is updated once at the beginning of the application.

AGR-APPL-SECURITY-CLASS

Contains the security class associated with the current application, as specified during application definition.

This field is updated once at the beginning of the application.

AGR-RESP-SECURITY-CLASS

Contains the security class associated with the response contained in the AGR-CURRENT-RESPONSE field, as specified on the Response Definition screen during application definition.

This field is updated on mapin from the terminal.

Note that if AGR-CURRENT-RESPONSE is modified by a process command, the runtime system does not perform security checking.

AGR-PRINT-CLASS

Contains the default print class for the application, as specified on the General Options screen during application definition.

AGR-PRINT-CLASS can be specified in a WRITE PRINTER command to specify a print class.

This field is updated once at the beginning of the application and is blank if the application has no default print class.

AGR-MODE

Contains the value passed to ADSO-APPLICATION-GLOBAL-RECORD from the AMR-MODE field of ADSO-APPLICATION-MENU-RECORD. The following level-88 condition names are defined for AGR-MODE:

AGR-STEP-MODE

AGR-MODE contains the value STEP.

AGR-FAST-MODE

AGR-MODE contains the value FAST.

This field is updated at the beginning of the application with the default mode specified on the Main Menu during application definition.

AGR-DATE-FORMAT

Contains a value indicating the date format specified by the application developer on the Main Menu during application definition. The following level-88 condition names are defined for AGR-DATE-FORMAT:

AGR-MMDDYY

AGR-DATE-FORMAT contains the value C.

AGR-DDMMYY

AGR-DATE-FORMAT contains the value E.

AGR-YYMMDD

AGR-DATE-FORMAT contains the value G.

AGR-YYDDD

AGR-DATE-FORMAT contains the value J.

This field is updated once at the beginning of the application.

AGR-AID-BYTE

Contains the AID byte that represents the control key pressed by the user.

A level-88 condition name is defined for each possible value.

AGR-CURRENT-FUNC-TYPE

Contains a value indicating the type of function named in the AGR-CURRENT-FUNCTION field. The following level-88 condition names are defined for AGR-CURRENT-FUNC-TYPE:

AGR-FUNCTION-DIALOG

AGR-CURRENT-FUNC-TYPE contains the value D.

AGR-FUNCTION-MENU

AGR-CURRENT-FUNC-TYPE contains the value M.

AGR-FUNCTION-SIGNON

AGR-CURRENT-FUNC-TYPE contains the value S.

This field is updated at the beginning of each function.

AGR-NEXT-FUNC-TYPE

Contains a value indicating the type of the function named in the AGR-NEXT-FUNCTION field. The following level-88 condition names are defined for AGR-NEXT-FUNC-TYPE:

AGR-NEXT-DIALOG

AGR-NEXT-FUNC-TYPE contains the value D.

AGR-NEXT-MENU

AGR-NEXT-FUNC-TYPE contains the value G.

AGR-NEXT-MENU-DIALOG

AGR-NEXT-FUNC-TYPE contains the value M.

AGR-NEXT-SIGNON

AGR-NEXT-FUNC-TYPE contains the value N.

AGR-NEXT-SIGNON-DIALOG

AGR-NEXT-FUNC-TYPE contains the value S.

AGR-NEXT-PROGRAM

AGR-NEXT-FUNC-TYPE contains the value P.

AGR-NEXT-SYSTEM-FUNC

AGR-NEXT-FUNC-TYPE contains the value F, B, O, U, R, T, or Q.

AGR-NEXT-FORWARD

AGR-NEXT-FUNC-TYPE contains the value F.

AGR-NEXT-BACKWARD

AGR-NEXT-FUNC-TYPE contains the value B.

AGR-NEXT-POP

AGR-NEXT-FUNC-TYPE contains the value O.

AGR-NEXT-POPTOP

AGR-NEXT-FUNC-TYPE contains the value U.

AGR-NEXT-RETURN

AGR-NEXT-FUNC-TYPE contains the value R.

AGR-NEXT-TOP

AGR-NEXT-FUNC-TYPE contains the value T.

AGR-NEXT-QUIT

AGR-NEXT-FUNC-TYPE contains the value Q.

This field is updated on mapin from the terminal.

AGR-CTRL-COMMAND

Contains a value indicating the control command associated with the response named in the AGR-CURRENT-RESPONSE field, as specified on the Response Definition screen during application definition. The following level-88 condition names are defined for AGR-CTRL-COMMAND:

AGR-TRANSFER

AGR-CTRL-COMMAND contains the value T.

AGR-INVOKE

AGR-CTRL-COMMAND contains the value I.

AGR-LINK

AGR-CTRL-COMMAND contains the value L.

AGR-RETURN

AGR-CTRL-COMMAND contains the value R.

This field is updated on mapin from the terminal. If a process command modifies AGR-CURRENT-RESPONSE to change the flow of control, AGR-CTRL-COMMAND does not have to be changed. On an EXECUTE NEXT FUNCTION command, the runtime system uses the control command associated with the response.

AGR-SIGNON-SWITCH

Contains a value indicating whether a system SIGNON function was performed for the current application. The following level-88 condition names are defined for AGR-SIGNON-SWITCH:

AGR-SIGNON-NOT-DONE

AGR-SIGNON-SWITCH contains the value N.

AGR-SIGNON-OK

AGR-SIGNON-SWITCH contains the value Y.

AGR-DIALOG-NAME

Contains the name of the dialog or user program (if any) associated with the function named in the AGR-CURRENT-FUNCTION field.

This field is updated at the beginning of a dialog, menu/dialog, or user program function, and is blank if the function is not associated with a dialog or user program.

AGR-FUNC-DESCRIPTION

Contains the description of the function named in the AGR-CURRENT-FUNCTION field, as specified on the Function Definition screen during application definition.

This field is updated at the beginning of each function and is blank if the function does not contain a description.

AGR-MESSAGE

AGR-MESSAGE is a 240-byte field that the application developer can use, as necessary. The runtime system never updates this field. (This field can be used for pass data.)

AGR-SIGNON-REQMTS

Contains a value indicating the signon requirements for the current application, as specified during application definition. The following level-88 condition names are defined for AGR-SIGNON-REQMTS:

AGR-SIGNON-REQUIRED

AGR-SIGNON-REQMTS contains the value R.

AGR-SIGNON-OPTIONAL

AGR-SIGNON-REQMTS contains the value O.

AGR-SIGNON-NOT-ALLOWED

AGR-SIGNON-REQMTS contains the value N.

This field is updated once at the beginning of the application.

AGR-MAP-RESPONSE

Contains a response name entered by the user in a field that maps to AGR-MAP-RESPONSE. AGR-MAP-RESPONSE performs the same function as a \$RESPONSE map field. The application developer can initialize the AGR-MAP-RESPONSE field with a default response name.

If both AGR-MAP-RESPONSE and \$RESPONSE are defined for a map, a value in the AGR-MAP-RESPONSE field has precedence over a value entered in the \$RESPONSE field. Once the AGR-MAP-RESPONSE field is initialized with a value, that value remains in the AGR-MAP-RESPONSE field until it is reinitialized with a new value either by process code or by the user.

Note: The \$RESPONSE map field can also be initialized by process code, through the \$RESPONSE system-supplied data field. For more information about the \$RESPONSE system-supplied data field, see the *CA IDMS Mapping Facility Guide*.

Usage

The following example illustrates the role of ADSO-APPLICATION-GLOBAL-RECORD during runtime execution of an application.

During execution of a nonmenu dialog function, the user selects a response that initiates either the FORWARD or BACKWARD system function. The following values are established in ADSO-APPLICATION-GLOBAL-RECORD:

- AGR-NEXT-FUNCTION contains FORWARD or BACKWARD, as applicable.
- AGR-CURRENT-RESPONSE contains the name of the response that initiates the FORWARD or BACKWARD system function.
- AGR-AID-BYTE contains the AID byte representing the control key pressed by the user.
- AGR-NEXT-FUNC-TYPE contains F or B, as applicable.
- AGR-CTRL-COMMAND contains a blank (X'40').

All other fields in the record remain unchanged.

The dialog can now access these fields to do its own paging, provided that ADSO-APPLICATION-GLOBAL-RECORD is defined to the dialog as a work record. (Automatic paging by the runtime system is performed only for menu functions.)

Note: For more information about the use of ADSO-APPLICATION-GLOBAL-RECORD, see the *CA ADS Application Design Guide*.

More information:

[CA ADS Application Compiler \(ADSA\)](#) (see page 51)

[CA ADS Runtime System](#) (see page 119)

[Variable Data Fields](#) (see page 285)

ADSO-APPLICATION-GLOBAL-RECORD

One copy of the record is automatically associated with all applications during application definition.

ADSO-APPLICATION-GLOBAL-RECORD is defined in the data dictionary as follows:

```

01  ADSO-APPLICATION-GLOBAL-RECORD.
    03  AGR-APPLICATION-NAME      PICTURE IS X(8)  USAGE IS DISPLAY.
    03  AGR-CURRENT-FUNCTION      PICTURE IS X(8)  USAGE IS DISPLAY.
    03  AGR-NEXT-FUNCTION         PICTURE IS X(8)  USAGE IS DISPLAY.
    03  AGR-CURRENT-RESPONSE     PICTURE IS X(8)  USAGE IS DISPLAY.
    03  AGR-DEFAULT-RESPONSE     PICTURE IS X(8)  USAGE IS DISPLAY.
    03  AGR-TASK-CODE            PICTURE IS X(8)  USAGE IS DISPLAY.
    03  AGR-EXIT-DIALOG          PICTURE IS X(8)  USAGE IS DISPLAY.
    03  AGR-PRINT-DESTINATION    PICTURE IS X(8)  USAGE IS DISPLAY.
    03  AGR-DATE                 PICTURE IS X(8)  USAGE IS DISPLAY.
    03  AGR-USER-ID              PICTURE IS X(32)  USAGE IS DISPLAY.
    03  AGR-SECURITY-CODE        PICTURE IS X(32)  USAGE IS DISPLAY.
    03  AGR-INSTALLATION-CODE   PICTURE IS X(32)  USAGE IS DISPLAY.
    03  AGR-PASSED-DATA          PICTURE IS X(32)  USAGE IS DISPLAY.
        05  AGR-PASSED-ONE        PICTURE IS X(32)  USAGE IS DISPLAY.
        05  AGR-PASSED-TWO        PICTURE IS X(32)  USAGE IS DISPLAY.
        05  AGR-PASSED-THREE     PICTURE IS X(32)  USAGE IS DISPLAY.
        05  AGR-PASSED-FOUR     PICTURE IS X(32)  USAGE IS DISPLAY.
    03  AGR-APPLICATION-VERSION  PICTURE IS S9(4)  USAGE IS COMP.
    03  AGR-APPL-SECURITY-CLASS  PICTURE IS S999  USAGE IS COMP.
    03  AGR-RESP-SECURITY-CLASS PICTURE IS S999  USAGE IS COMP.
    03  AGR-PRINT-CLASS          PICTURE IS S999  USAGE IS COMP.
    03  AGR-MODE                 PICTURE IS X(4)  USAGE IS DISPLAY.
        88  AGR-STEP-MODE        USAGE IS CONDITION-NAME VALUE IS 'STEP'.
        88  AGR-FAST-MODE        USAGE IS CONDITION-NAME VALUE IS 'FAST'.
    03  AGR-DATE-FORMAT          PICTURE IS X          USAGE IS DISPLAY.
        88  AGR-MMDDYY           USAGE IS CONDITION-NAME VALUE IS 'C'.
        88  AGR-DDMMYY           USAGE IS CONDITION-NAME VALUE IS 'E'.
        88  AGR-YYMMDD           USAGE IS CONDITION-NAME VALUE IS 'G'.
        88  AGR-YYDDD            USAGE IS CONDITION-NAME VALUE IS 'J'.

```



```
03 AGR-AID-BYTE          PICTURE IS X      USAGE IS DISPLAY.
88 AGR-ENTER            USAGE IS CONDITION-NAME VALUE IS QUOTE.
88 AGR-PF1              USAGE IS CONDITION-NAME VALUE IS '1'.
88 AGR-PF2              USAGE IS CONDITION-NAME VALUE IS '2'.
88 AGR-PF3              USAGE IS CONDITION-NAME VALUE IS '3'.
88 AGR-PF4              USAGE IS CONDITION-NAME VALUE IS '4'.
88 AGR-PF5              USAGE IS CONDITION-NAME VALUE IS '5'.
88 AGR-PF6              USAGE IS CONDITION-NAME VALUE IS '6'.
88 AGR-PF7              USAGE IS CONDITION-NAME VALUE IS '7'.
88 AGR-PF8              USAGE IS CONDITION-NAME VALUE IS '8'.
88 AGR-PF9              USAGE IS CONDITION-NAME VALUE IS '9'.
88 AGR-PF10             USAGE IS CONDITION-NAME VALUE IS ':'.
88 AGR-PF11             USAGE IS CONDITION-NAME VALUE IS '#'.
88 AGR-PF12             USAGE IS CONDITION-NAME VALUE IS '@'.
88 AGR-PF13             USAGE IS CONDITION-NAME VALUE IS 'A'.
88 AGR-PF14             USAGE IS CONDITION-NAME VALUE IS 'B'.
88 AGR-PF15             USAGE IS CONDITION-NAME VALUE IS 'C'.
88 AGR-PF16             USAGE IS CONDITION-NAME VALUE IS 'D'.
88 AGR-PF17             USAGE IS CONDITION-NAME VALUE IS 'E'.
88 AGR-PF18             USAGE IS CONDITION-NAME VALUE IS 'F'.
88 AGR-PF19             USAGE IS CONDITION-NAME VALUE IS 'G'.
88 AGR-PF20             USAGE IS CONDITION-NAME VALUE IS 'H'.
88 AGR-PF21             USAGE IS CONDITION-NAME VALUE IS 'I'.
88 AGR-PF22             USAGE IS CONDITION-NAME VALUE IS '&cent.'.
88 AGR-PF23             USAGE IS CONDITION-NAME VALUE IS '.'.
88 AGR-PF24             USAGE IS CONDITION-NAME VALUE IS '<'.
88 AGR-PA1              USAGE IS CONDITION-NAME VALUE IS '%'.
88 AGR-PA2              USAGE IS CONDITION-NAME VALUE IS '>'.
88 AGR-PA3              USAGE IS CONDITION-NAME VALUE IS ','.
88 AGR-CLEAR            USAGE IS CONDITION-NAME VALUE IS '_'.
88 AGR-LPEN             USAGE IS CONDITION-NAME VALUE IS '='.
88 AGR-EOF              USAGE IS CONDITION-NAME VALUE IS 'N'.
88 AGR-IOERR            USAGE IS CONDITION-NAME VALUE IS 'O'.
88 AGR-SERR             USAGE IS CONDITION-NAME VALUE IS 'P'.
88 AGR-IERR             USAGE IS CONDITION-NAME VALUE IS 'R'.
88 AGR-OERR             USAGE IS CONDITION-NAME VALUE IS 'S'.
```

```

03 AGR-CURRENT-FUNC-TYPE PICTURE IS X      USAGE IS DISPLAY.
   88 AGR-FUNCTION-DIALOG  USAGE IS CONDITION-NAME
                                   VALUE IS 'D'.
   88 AGR-FUNCTION-MENU    USAGE IS CONDITION-NAME
                                   VALUE IS 'M'.
   88 AGR-FUNCTION-SIGNON  USAGE IS CONDITION-NAME
                                   VALUE IS 'S'.
03 AGR-NEXT-FUNC-TYPE     PICTURE IS X      USAGE IS DISPLAY.
   88 AGR-NEXT-DIALOG     USAGE IS CONDITION-NAME VALUE IS 'D'.
   88 AGR-NEXT-MENU       USAGE IS CONDITION-NAME VALUE IS 'G'.
   88 AGR-NEXT-MENU-DIALOG USAGE IS CONDITION-NAME
                                   VALUE IS 'M'.
   88 AGR-NEXT-SIGNON     USAGE IS CONDITION-NAME VALUE IS 'N'.
   88 AGR-NEXT-SIGNON-DIALOG USAGE IS CONDITION-NAME
                                   VALUE IS 'S'.
   88 AGR-NEXT-PROGRAM    USAGE IS CONDITION-NAME VALUE IS 'P'.
   88 AGR-NEXT-SYSTEM-FUNC USAGE IS CONDITION-NAME
                                   VALUE IS 'F'
                                   VALUE IS 'B'
                                   VALUE IS 'O'
                                   VALUE IS 'U'
                                   VALUE IS 'R'
                                   VALUE IS 'T'
                                   VALUE IS 'Q'.
   88 AGR-NEXT-FORWARD    USAGE IS CONDITION-NAME VALUE IS 'F'.
   88 AGR-NEXT-BACKWARD   USAGE IS CONDITION-NAME VALUE IS 'B'.
   88 AGR-NEXT-POP        USAGE IS CONDITION-NAME VALUE IS 'O'.
   88 AGR-NEXT-POPTOP     USAGE IS CONDITION-NAME VALUE IS 'U'.
   88 AGR-NEXT-RETURN     USAGE IS CONDITION-NAME VALUE IS 'R'.
   88 AGR-NEXT-TOP        USAGE IS CONDITION-NAME VALUE IS 'T'.
   88 AGR-NEXT-QUIT       USAGE IS CONDITION-NAME VALUE IS 'Q'.
03 AGR-CTRL-COMMAND      PICTURE IS X      USAGE IS DISPLAY.
   88 AGR-TRANSFER        USAGE IS CONDITION-NAME VALUE IS 'T'.
   88 AGR-INVOKE          USAGE IS CONDITION-NAME VALUE IS 'I'.
   88 AGR-LINK            USAGE IS CONDITION-NAME VALUE IS 'L'.
   88 AGR-RETURN          USAGE IS CONDITION-NAME VALUE IS 'R'.
03 AGR-SIGNON-SWITCH     PICTURE IS X      USAGE IS DISPLAY
                                   VALUE IS 'N'.
   88 AGR-SIGNON-NOT-DONE  USAGE IS CONDITION-NAME
                                   VALUE IS 'N'.
   88 AGR-SIGNON-OK        USAGE IS CONDITION-NAME VALUE IS 'Y'.
03 AGR-DIALOG-NAME       PICTURE IS X(8)    USAGE IS DISPLAY.
03 AGR-FUNC-DESCRIPTION  PICTURE IS X(28)   USAGE IS DISPLAY.
03 AGR-MESSAGE           PICTURE IS X(240)  USAGE IS DISPLAY.
03 AGR-SIGNON-REQMTS     PICTURE IS X      USAGE IS DISPLAY.
   88 AGR-SIGNON-REQUIRED  USAGE IS CONDITION-NAME
                                   VALUE IS 'R'.
   88 AGR-SIGNON-OPTIONAL  USAGE IS CONDITION-NAME
                                   VALUE IS 'O'.

```

```
      88  AGR-SIGNON-NOT-ALLOWED USAGE IS CONDITION-NAME
                                         VALUE IS 'N'.
03  AGR-MAP-RESPONSE      PICTURE IS X(8)  USAGE IS DISPLAY.
03  FILLER                 PICTURE IS X(54)  USAGE IS DISPLAY.
```

ADSO-APPLICATION-MENU-RECORD

The CA ADS runtime system builds menus by storing information in ADSO-APPLICATION-MENU-RECORD. This record is associated with maps used by menu and menu/dialog functions.

The menu map can be system-defined or user-defined. If a menu map is user-defined, ADSO-APPLICATION-MENU-RECORD must be explicitly associated with the map when it is defined.

Usage

In a menu/dialog function, ADSO-APPLICATION-MENU-RECORD is initialized at the beginning of the dialog, and its fields are primed by the runtime system when the map is displayed.

Thus, for example, at the beginning of a menu/dialog, AMR-PASSING does not contain any value passed by the previous menu function (AGR-PASSED-ONE of ADSO-APPLICATION-GLOBAL-RECORD does); and any value moved to a field in ADSO-APPLICATION-MENU-RECORD will be overwritten when the menu map is displayed.

Field Descriptions

AMR-PAGE

Maps to the PAGE field.

AMR-PAGE contains the number of the currently displayed page of the menu screen.

AMR-TOTAL-PAGES

Maps to the OF field.

AMR-TOTAL-PAGES contains the total number of pages for the current menu.

AMR-NEXT-PAGE

Maps to the NEXT PAGE field.

AMR-NEXT-PAGE contains the number of the next page to be displayed, as entered by the user.

AMR-HEADING

(or AMR-HDG) Maps to the heading text area.

AMR-HEADING or AMR-HDG contains the heading text specified by the application developer on the Function Definition (Menu) screen during application definition.

AMR-DATE

Maps to the DATE field.

AMR-DATE contains the current date in the format specified by the application developer on the General Options screen during application definition.

AMR-DIALOG

Maps to the DIALOG field.

If the current menu is associated with a dialog, AMR-DIALOG contains the name of the menu/dialog, as specified on the Response/Function List screen during application definition.

AMR-RESPONSE-FIELD

Maps to the RESPONSE field.

AMR-RESPONSE-FIELD contains the name of the next response to be executed, as entered by the user.

AMR-RESPONSE-FIELD is initialized with the default response for the current function (if any), as specified on the Function Definition screen during application definition. If the user does not specify a response, the default response remains in AMR-RESPONSE-FIELD.

AMR-MODE

Maps to the MODE field.

MODE contains the execution mode for the function, as specified on the Main Menu during application definition. At runtime, the user can change the specification in the MODE field on a menu map, thereby modifying the value in AMR-MODE. The value in AMR-MODE is passed to the AGR-MODE field of ADSO-APPLICATION-GLOBAL-RECORD each time the menu is mapped in.

AMR-PASSING

Maps to the SEND DATA field.

AMR-PASSING contains data to be passed to the next function, as entered by the user. CA ADS transfers the contents of AMR-PASSING to the AGR-PASSED-ONE field of ADSO-APPLICATION-GLOBAL-RECORD. The runtime system reinitializes the AMR-PASSING field each time the menu is mapped out.

AMR-USER-ID

Maps to the ENTER USER ID field of a signon menu.

AMR-USER-ID contains the user id entered by the user.

CA ADS transfers the contents of AMR-USER-ID to the AGR- USER-ID field of ADSO-APPLICATION-GLOBAL-RECORD. If a system SIGNON function is initiated, the runtime system passes the value in AMR-USER-ID to CA IDMS/DC or DC/UCF (DC/UCF) for security verification.

AMR-PASSWORD

Maps to the PASSWORD field of a signon menu.

AMR-PASSWORD contains the password entered by the user.

If a system SIGNON function is initiated, the runtime system passes the value in AMR-PASSWORD to DC/UCF for security verification. After passing the value, CA ADS overwrites the AMR-PASSWORD field with blanks.

AMR-SELECT-SECTION

Maps to the responselisting area.

AMR-SELECT-SECTION is a group field that occurs 50 times.

Each valid response associated with the current function is moved to an occurrence of AMR-SELECT-SECTION. The occurrences are mapped out to the menu screen one page at a time.

Each occurrence of AMR-SELECT-SECTION consists of the following elements:

AMR-SELECT

Contains the character entered by the user in the one-byte field provided to select the response.

AMR-RESPONSE

Contains the name of the response, as specified on the Response/Function List screen during application definition.

AMR-KEY

Contains the AID byte representing the control key that initiates the response, as specified on the Response/Function List screen during application definition.

AMR-KEY is translated by a code table to a five-byte map field in order to display the associated control key.

AMR-DESCRIPTION

Contains the description of the response, as specified on the Response Definition screen during application definition.

More information:

[CA ADS Application Compiler \(ADSA\)](#) (see page 51)

[CA ADS Runtime System](#) (see page 119)

ADSO-APPLICATION-MENU-RECORD

ADSO-APPLICATION-MENU-RECORD is defined in the data dictionary as follows:

```

01 ADSO-APPLICATION-MENU-RECORD.
   03 AMR-PAGE                PICTURE IS S99      USAGE IS COMP.
   03 AMR-TOTAL-PAGES        PICTURE IS S99      USAGE IS COMP.
   03 AMR-NEXT-PAGE         PICTURE IS S99      USAGE IS COMP.
   03 AMR-HEADING           PICTURE IS X(237)    USAGE IS DISPLAY.
   03 AMR-HDG REDEFINES AMR-HEADING  USAGE IS DISPLAY.
       05 AMR-HL1            PICTURE IS X(79)    USAGE IS DISPLAY.
       05 AMR-HL2            PICTURE IS X(79)    USAGE IS DISPLAY.
       05 AMR-HL3            PICTURE IS X(79)    USAGE IS DISPLAY.
   03 AMR-DATE               PICTURE IS X(8)      USAGE IS DISPLAY.
   03 AMR-DIALOG             PICTURE IS X(8)      USAGE IS DISPLAY.
   03 AMR-RESPONSE-FIELD    PICTURE IS X(8)      USAGE IS DISPLAY.
   03 AMR-MODE               PICTURE IS X(4)      USAGE IS DISPLAY.
   03 AMR-PASSING            PICTURE IS X(32)     USAGE IS DISPLAY.
   03 AMR-USER-ID           PICTURE IS X(32)     USAGE IS DISPLAY.
   03 AMR-PASSWORD          PICTURE IS X(8)      USAGE IS DISPLAY.
   03 AMR-SELECT-SECTION    USAGE IS DISPLAY    OCCURS 50 TIMES.
       05 AMR-SELECT         PICTURE IS X        USAGE IS DISPLAY.
       05 AMR-RESPONSE      PICTURE IS X(8)     USAGE IS DISPLAY.
       05 AMR-KEY           PICTURE IS X        USAGE IS DISPLAY.
       05 AMR-DESCRIPTION   PICTURE IS X(28)    USAGE IS DISPLAY.

```

Appendix B: CA ADS Dialog and Application Reporter

This section contains the following topics:

[Overview](#) (see page 583)

[Dialog Reports](#) (see page 584)

[Application Reports](#) (see page 595)

[Control Statements](#) (see page 596)

[SYSIDMS Parameter File](#) (see page 604)

[JCL and Commands To Run Reports](#) (see page 605)

Overview

The CA ADS dialog and application reporter (ADSORPTS) is used to request batch reports about dialogs and applications. Reports can be summary or detailed. One dialog and/or application can be reported on, or several. Dialogs and applications to be included can be specified as a list of names, name ranges, and mask values.

Additional reports (AREPORTs) also provide information about dialogs and their components that are stored in the data dictionary. The information provided by each report is shown below.

Note: For more information about these reports, see the *CA IDMS Reports Guide*.

This appendix provides the following information about ADSORPTS:

- A description of the dialog reporting capabilities
- A description of the application reporting capabilities
- Syntax rules for control statements
- JCL and commands for running reports

AREPORTs Documenting CA ADS Dialogs

Report	Description
1	Lists detail information about dialogs and their components
2	Lists information about specified dialogs
3	Lists all dialogs associated with specified processes

Report	Description
4	Lists all dialogs associated with specified records.
5	Lists all dialogs associated with specified subschemas
6	Lists all dialogs associated with specified maps

Dialog Reports

The dialog reporting capabilities of ADSORPTS enable the application developer to request any or all of the following reports for one or several dialogs:

Summary Reports

These list the following information about the object dialog:

- The name and version number of the dialog, and the date and time at which the dialog was compiled
- The name and version number of the map associated with the dialog, and the date and time at which the map was compiled
- The name and version number of the schema associated with the dialog
- The name of the subschema associated with the dialog
- The dialog's autostatus specification
- The dialog's FDB size

Processes Reports

These list the module source statements for the premap and response processes associated with the object dialog. Source statements from included process modules are listed separately.

For each module listed, the following information is provided:

- The name and version number of the module
- The date on which the module was created, the date on which the module was last modified, and the ids of the users who created and modified the module

Note: The cross-reference report options are available with processes.

Records Reports

These list the following information:

- The dictionary definitions for all records associated with the dialog
- The decimal position and hexadecimal offset of the fields in the listed records
- The lengths of the fields in the listed records

FDBLIST Reports

These list the contents of the Fixed Dialog Block (FDB) for the dialog. The FDBLIST report includes the following information:

- General information, such as the dialog's name and compilation date, its map name and compile date, and its subschema description.
- Record descriptions contained in the Record Description Elements (RDEs).
- Premap process information contained in the Premap Process Element (PME).
- Response process information contained in the Response Process Elements (RSEs).
- Object code contained in the Process Object Code Table. One Process Object Code Table is associated with each PME and RSE of an FDB that contains object code.
- Command information contained in the Command Element (CME). One or more CMEs can be associated with a PME or RSE.
- Executable code and vector call information contained in the Executable Code/Vector Call Offset Table. One Executable Code/Vector Call Offset Table is associated with each PME and RSE of an FDB that contains object code.
- Included process module information in the Included Module Table (MDTA). One MDTA is displayed for each process in a dialog that has included modules.

Note that the representation of premap and response process information in a dialog's FDB and, consequently, the representation of this information in the report, depend on how the dialog was compiled, as follows:

- If the dialog was compiled with the symbol table option enabled, premap and response process commands are converted in the FDB to a series of command elements (CMEs).
- If the dialog was compiled without the symbol table option, premap and response process commands are converted in the FDB to object code. The object code for a command can be either an item of executable code or a vector call that references a CME.

Note: Dialogs being reported on by ADSORPTS should be compiled with the diagnostic tables option enabled. ADSORPTS uses diagnostic tables to format premap and response process information. If a dialog's FDB does not contain diagnostic tables, ADSORPTS produces an unformatted report wherever formatting is not possible.

Fixed Dialog Block Field Descriptions

The following table lists the fields displayed in the FDBLIST report.

Group	Field	Description
FDB	ID	Fixed dialog block identifier
	NAME	Dialog name
	DATE	Date dialog compiled
	TIME	Time dialog compiled
	MPNM	Map name
	MPDT	Date map compiled
	MPTM	Time map compiled
	SCHNM	Schema name
	SSNM	Subschema name
	RDEA	Offset— start of record table
	PMEA	Offset— start of premap element
	RSEA	Offset— start of response table
	LITA	Offset— start of literal pool
	SSANA	Offset— subschema area name table
	NSSAN	Number of subschema area names
	SVER	Schema version
	MPVER	Map version
	DVER	Dialog version
	NRECS	Number of map records
	NFLDS	Number of map fields
	NDREC	Number of dialog records
	RSPMI	MRE index of map response field
	MSGMI	MRE index of map message field
	SEGVW	MRB— subschema segmented view
	FLAG	Fixed dialog block flag byte
	LREA	Offset— first logical record RDE
	ASRA	Offset— status definition record ASR

Group	Field	Description
	RLSE	CA ADS release
	FLAG2	FDB flag byte 2
	MAPPG	Map paging type
	HEXTA	Offset— FDB header extension area
	MDBO	Offset— map descriptor block (MDB)
	FLAG3	FDB flag byte 3
	PREFX	Message prefix
	DRSPO	Offset— default response process
	FDEO	Offset— format description headers (FDH) and elements (FDE)
FHE (FDB header extension)	NODE	Alternate DB name
	DICT	Alternate dictionary name
	SDDN	Suspense file DD name
	DCLA	Offset— SQL declaration process
	SQLAM	SQL Access module name
	SQLTM	SQL time format
	SQLDT	SQL date format
	SQLFL	SQL compliance flag
MDB (map descriptor block)	MPNAM	Map name (batch)
	NEXT	Offset— next MDB
	DATE	Date map compiled
	TIME	Time map compiled
	VER	Version
	NRECS	Number of records
	NFLDS	Number of fields
	RSPMI	MRE index of map response field
	FLG1	Flag byte 1

Group	Field	Description
	DDNAM	File/ddname
	CRECL	Compressed length for output map external record
	RECL	Real external record length
	CRECO	Offset— compressed external record
SSAN		Subschema area names
ASR	NAME	Status definition record name
	VER	Status definition record version
RDE	NAME	Record name
	NRDEA	Offset— next RDE
	RECL	Record length (except logical records)
	NLRE	Number of logical record elements
	VER	Record version
	INDX	Relative variable record element index entry
	MINDX	Map record index
	FLG1	Flag byte 1
	FLG2	Flag byte 2
	CRECL	Compressed INIT record size
	INTOF	Offset— RDEINITV within RDE to the compressed initialized record
	NLRA	Offset— next logical record RDE (logical records only)
	FLG3	Flag byte 3
	IMNDX	Input map record index
	OMNDX	Output map record index
	SCHML	Length of schema name when created from an SQL table
	SCHMO	Offset into RDE of schema name when created from an SQL table
	INITV	Initial value (in compressed format)
FDH (format description header)	LEN	Length of format description

Group	Field	Description
	ID	Format identifier
	FDES	Start of format descriptor element
FDE (format description element)	TYPE	Element type
	FLAGS	Flag byte
	PEND	Type dependent section
DCL (declaration module)	NAME	Declaration module name
	VER	Declaration module version
	DATLU	Date module last updated
	DATCR	Date module last created
PME	NAME	Premap process name
	LASTB	Offset of last byte in PME
	RATA	Offset to ready area table
	FCMEA	Offset to first CME ¹
	PVER	Process version
	NCMES	Number of CMEs in response ²
	NEWF	Initialized to X'FF' if new format
	FLAG1	Flag byte
	NMDTE	Number of module table entries
	LNTA	Offset of line number table
	DATLU	Date module last updated
	DATCR	Date module created
	MDTA	Offset of included module table
	OFTBL	Offset to executable code/vector call offset table
RSE	NAME	Response process name
	NXTA	Offset of next RSE in FDB
	LASTB	Offset of last byte in response process
	RATA	Offset to ready area table

Group	Field	Description
	FCMEA	Offset to first CME ¹
	PVER	Process version
	NCMES	Number of CMEs in response ²
	PFKEY	PF key for response
	FUNLN	Length of response field
	OFUNC	Start old-format function code
	FLAG1	Flag byte
	FUNOF	Offset within RSE to function code
	NMDTE	Number of module table entries
	LNTA	Offset of line number table
	DATLU	Date module last updated
	DATCR	Date module created
	MDTA	Offset of included module table
	OFTBL	Offset to executable code/vector call offset table
	FUNC	Response field value
PROCESS OBJECT CODE TABLE	ICMD# GENERATED CODE	Internal command number Executable code and vector calls
CME	CLASS	Command element major class
	FUNC	Command element function
	NXTA	Offset to next CME from first CME within PME or RSE
	NEXT	Offset of next CME with FDB
	INCLUDED MODULE	Name of included module from which CME was generated
	VERS	Included module version
	SEQ#	IDD sequence number
	FLAG1	First flag byte
	FLAG2	Second flag byte
	FLAG3	Third flag byte
	FLAG4	Fourth flag byte

Group	Field	Description
	ICMD#	Internal command number
	BODY	Parameter for use by run-time system
EXEC CODE/ VECTOR CALL OFFSET TABLE	ICMD#	Internal command number
	VECTOR #	Vector code
	CODE/CME LEN	Length of object code, if executable code; length of CME, if a vector call
	CODE OFF	Offset from first item of object code in process
	VECTOR CALL	Identifies an ICMD as a vector call
RAT TABLE		Ready Area Table
INCLUDED MODULE TABLE	PROCESS	Included module name
VER	Included module version	
	DATLU	Date included module last updated
	DATCR	Date included module created
LIT	POOL	Literal pool

1 If the FDB contains object code, the FCMEA indicates the offset to the first item of object code in the Process Object Code Table.

2 If the FDB contains object code, the NCMES contains the original number of CMEs before their conversion to executable code and vector calls.

Vector Call Codes

The following table lists vector call codes and their associated process commands.

Vector code	Process command
■ 0000	■ Database command
■ 0001	■ Database command— logical record
■ 0002	■ ABORT
■ 0003	■ INVOKE
■ 0004	■ TRANSFER
■ 0005	■ RETURN
■ 0006	■ DISPLAY
■ 0007	■ --
■ 0008	■ LEAVE
■ 0009	■ LINKT
■ 000A	■ Assignment command
■ 000B	■ Conditional command
■ 000C	■ WHILE REPEAT
■ 000D	■ Internal branch
■ 000E	■ Subroutine call
■ 000F	■ --

Vector code	Process command
■ 0010	■ ON
■ 0011	■ ADD
■ 0012	■ SUBTRACT
■ 0013	■ MULTIPLY
■ 0014	■ DIVIDE
■ 0015	■ MOVE
■ 0016	■ COMPUTE
■ 0017	■ MODIFY MAP
■ 0018	■ (DC) ACCEPT
■ 0019	■ PUT/GET/DELETE SCRATCH
■ 001A	■ PUT/GET/DELETE QUEUE
■ 001B	■ WRITE PRINTER
■ 001C	■ INITIALIZE RECORDS
■ 001D	■ KEEP LONGTERM
■ 001E	■ SNAP
■ 001F	■ COMMIT

Vector code	Process command
■ 0020	■ ROLLBACK TASK
■ 0021	■ EXECUTE NEXT FUNCTION
■ 0022	■ --
■ 0023	■ --
■ 0024	■ Last vector call; end of process
■ 0025	■ PUT DETAIL
■ 0026	■ GET DETAIL
■ 0027	■ WRITE TRANSACTION
■ 0028	■ READ TRANSACTION
■ 0029	■ CONTINUE
■ 002A	■ WRITE TO LOG
■ 002B	■ CLOSE FILE MAPS
■ 003D	■ ALLOCATE
■ 003E	■ CONTROL SESSION
■ 003F	■ SEND-DATA
	■ CONFIRM
■ 0040	■ CONFIRM
■ 0041	■ CONFIRMED
■ 0042	■ REQUEST-TO-SEND
■ 0043	■ SEND-ERROR
■ 0044	■ RECEIVE-AND-WAIT
■ 0045	■ PREPARE-TO-RECEIVE
■ 0046	■ DEALLOCATE
■ 0047	■ SQL call
■ 0048	■ TRACE
■ 0049	■ OPEN
■ 004A	■ CLOSE
■ 004B	■ SEND
■ 004C	■ RECEIVE

Debugging Information

The information provided in the dialog reports generated by ADSORPTS can be used for debugging. For example, when the CA ADS runtime system causes a dialog to abnormally terminate, it sends messages to the system log. The messages provide the following information:

- The reason for the abnormal termination.
- The name of the aborted dialog.
- The name of the process that was executing at the time of the termination.
- The hexadecimal offset within the Fixed Dialog Block (FDB) of the command that was executing at the time of the termination. If the FDB does not contain object code, the offset is to the CME representing the command. If the FDB contains object code, the offset is to the object code that references the CME representing the command.
- The IDD sequence number (SEQ#) of the source line containing the command that caused theabend.
- The internal command number (ICMD#) of the source line containing the command that caused theabend.

Note: The information above is also displayed on the Dialog Abort Information screen, if enabled.

The information in the system messages can be used in conjunction with the FDBLIST report to determine the command that caused theabend. The internal command number and the hexadecimal offset of the problem command can both be used to locate the command as it is represented in the Process Object Code Table, the list of CMEs, and the Executable Code/Vector Call Offset Table. A CME displays the process command that it represents; an item in the Process Object Code Table and the Executable Code/Vector Call Offset Table displays the vector code of the command, as described in the vector call codes table earlier in this appendix.

More information:

[CA ADS Runtime System](#) (see page 119)

[CA ADS Dialog Compiler \(ADSC\)](#) (see page 91)

Application Reports

The application reporting capabilities of ADSORPTS enable the application developer to request any or all of the following reports for one or several applications:

- **Summary reports** list information about task codes, global records, functions, and responses.
- **Records reports** list information about global records.

- **Functions/responses detail reports** list information about functions, responses, and the relationships between functions and responses.
- **Functions/responses summary reports** list the relationships between functions and responses.

Additionally, all of the application reports list basic information about the application, such as security requirements and application-wide defaults.

Control Statements

ADSORPTS is driven by five control statements, as shown below.

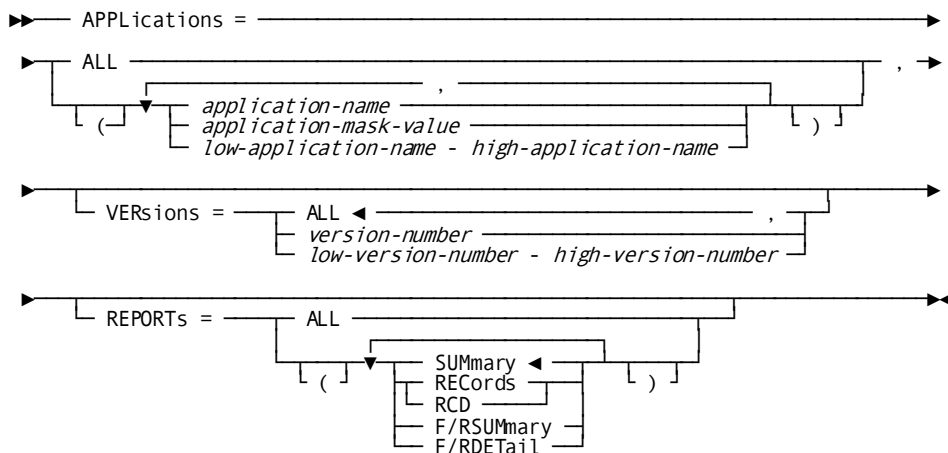
Summary of the ADSORPTS Control Statements

Control statement	Purpose
APPLICATIONS	Specifies the applications for which reports are being requested and the reports desired for the applications
DIALOGS	Specifies the dialogs for which reports are being requested and the reports desired for the dialogs
LIST	Controls the online or printed format of the ADSORPTS output
SEARCH	Specifies whether ADSORPTS searches in the load (core-image) library or the load area for the dialogs and applications specified by the DIALOGS and APPLICATIONS control statements

APPLICATIONS

Purpose

Generates reports for specified applications.

Syntax**Parameters****ALL**

Specifies all applications in the load area.

application-name

Specifies the 1- to 8-character name of a single application.

If the name includes a hyphen (-) as a character, replace with a mask character. The mask character is the asterisk (*).

application-mask-value

Specifies any application with a name that matches the mask criteria.

The mask character is the asterisk (*); it matches any character. For example, APPLICATIONS=ORE***** generates the requested reports for all applications beginning with ORE.

low-application-name - high-application-name

Specifies all applications within the application-name range (inclusive).

The hyphen (-) is required and cannot have surrounding blanks.

Application names and masks that have fewer than eight characters are padded on the right with blanks.

VERSIONS =

Introduces the version numbers of the applications for which reports are requested.

ALL

Specifies all versions of the named applications.

ALL is the default when no other version is specified.

version-number

Specifies a single version number for the named applications.

low-version-number - high-version-number

Specifies all versions of the named applications within the version-number range (inclusive).

The hyphen (-) is required and cannot have surrounding blanks.

REPORTS=

Introduces the reports requested for the named applications.

ALL

Requests the summary, records, and functions/responses detail reports for the named applications.

SUMmary

Requests summary reports for the named applications.

SUMMARY is the default when no other report is specified.

RECords

Requests records reports for the named applications.

F/RSUMmary

Requests functions/responses summary reports for the named applications.

F/RDETail

Requests functions/responses detail reports for the named applications.

Usage

Considerations

If both dialog and application reports are requested in a single ADSORPTS run, dialogs are reported first, followed by applications.

Example 1: Requesting summary reports

The following statement requests summary reports for all versions of applications with names in the range A through C (inclusive):

```
APPLICATIONS=(A-C) ,REPORTS=SUMMARY
```

Example 2: Requesting all reports

The following statement requests all reports for all applications whose names begin with ABC and whose version number is 20:

```
APPLICATIONS=ABC*****,VERSION=20,REPORT=ALL
```

Example 3: Requesting summary and records reports for all versions

The following statement requests summary and records reports for all versions of applications with names that contain the characters S and T in the third and fourth positions and blanks in the last two positions:

```
APPLICATIONS=**ST** ,REPORTS=(SUMMARY,RECORDS)
```

Example 4: Requesting functions/responses summary reports

The next statement requests functions/responses summary reports for all versions of the following applications:

- Applications whose names begin with the characters ABC
- Applications whose names contain the characters S and T in the third and fourth positions and blanks in the last two positions
- Applications whose names are in the range A through C (inclusive)

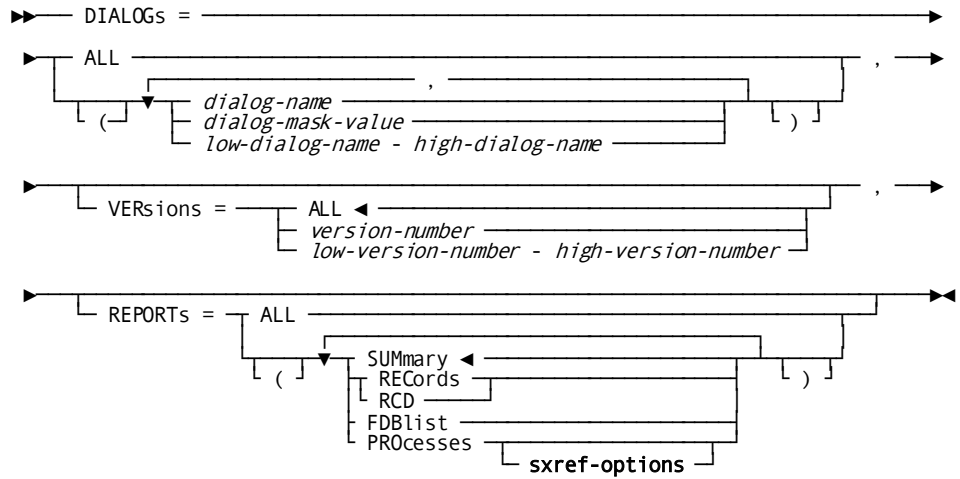
```
APPLICATIONS=(ABC*****,**ST** ,A-C) ,REPORTS=F/RSUMMARY
```

DIALOGS

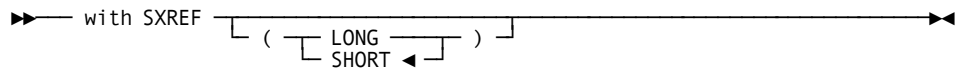
Purpose

Generates reports for specified dialogs.

Syntax



Expansion of sxref-options



Parameters

ALL

Generates reports for all dialogs in the load area.

dialog-name

Specifies the 1- to 8-character name of a single dialog.

If the name includes a hyphen (-) as a character, replace with a mask character. The mask character is the asterisk (*).

dialog-mask-value

Specifies any dialog with a name that matches the mask criteria. The mask character is the asterisk (*); it matches any character. For example, DIALOGS=DCB***** generates the requested reports for all dialogs beginning with DCB.

low-dialog-name - high-dialog-name

Specifies all dialogs within the dialog-name range (inclusive).

The hyphen (-) is required and cannot have surrounding blanks.

Dialog names and masks that have fewer than eight characters are padded on the right with blanks.

VERSIONS =

Introduces the version numbers of the dialogs for which reports are requested.

ALL

Specifies all versions of the named dialogs.

ALL is the default when no other version is specified.

version-number

Specifies a single version number for the named dialogs.

low-version-number - high-version-number

Specifies all versions of the named dialogs within the version-number range (inclusive). The hyphen (-) is required and cannot have surrounding blanks.

REPORTS =

Introduces the reports requested for the named dialogs.

ALL

Requests all reports (that is, the summary, processes, records, and FDBLIST reports) for the named dialogs.

SUMmary

Requests summary reports for the named dialogs.

SUMMARY is the default when no other report is specified.

RECORDs

Requests records reports for the named dialogs.

FDBlist

Requests FDBLIST reports for the named dialogs.

PROcesses

Requests processes reports for the named dialogs.

sxref-options

Specifies sorted cross-reference report options.

with SXREF

Requests a sorted cross-reference for process reports. The usage of all data names and subroutine calls is cross-referenced.

LONG

Specifies that all elements be included in the report.

SHORT

Specifies that only elements that are referenced be included in the report.

SHORT is the default.

Note: When specifying the cross-reference option, the Master Function Table (RHDCEVBF) must reside in either the load area or the load library.

Usage

Considerations

A maximum of 100 dialog report requests can be specified in a single ADSORPTS run. If both dialog and application reports are requested in a single ADSORPTS run, dialogs are reported first, followed by applications, regardless of their order in the control statements.

Example 1: Requesting summary reports

The following statement requests summary reports for all versions of dialogs with names in the range A-C (inclusive):

```
DIALOGS=(A-C),REPORTS=SUMMARY
```

Example 2: Requesting summary and records reports

The following statement requests all reports for all dialogs with names that contain the characters S and T in the third and fourth positions and blanks in the last two positions:

```
DIALOGS=**ST**,REPORTS=(SUMMARY,RECORDS)
```

Example 3: Requesting summary reports for all versions

The next statement requests summary reports for all versions of these dialogs:

- Dialogs whose names begin with the characters ABC
- Dialogs whose names contain the characters S and T in the third and fourth positions and blanks in the last two positions
- Dialogs whose names are in the range A through C (inclusive)

```
DIALOGS=(ABC****, **ST**, A-C)
```

Example 4: Requesting all reports for the dialog named TBXSUMD

The following statement requests all reports for the named dialog:

```
DIALOGS=TBXSUMD,REPORTS=ALL
```


loadLIB

Specifies that ADSORPTS searches for the dialogs and applications in the load (core-image) library.

Usage

Considerations

- If more than one SEARCH statement is submitted for a single run of ADSORPTS, the specification in the last SEARCH statement applies for the entire run.
- The DIALOGS and APPLICATIONS statements cannot specify a range of dialog or application names (such as *low-dialog-name - high-dialog-name*) or a dialog or application mask (such as *dialog-mask-value*).
- The load (core-image) libraries in which the dialogs and applications are located must be specified in the JCL or commands that run the reports, as follows:
 - **z/OS JCL**
In the CDMSLIB statement or, if a CDMSLIB statement is not specified, in the STEPLIB statement
 - **z/VSE JCL**
In the ASSGN/EXTNT statement for the private core-image library or in the LIBDEF equivalent
 - **z/VM commands**
In the GLOBAL LOADLIB command, added to the list of libraries

SYSIDMS Parameter File

For more information about SYSIDMS parameters, see the *CA IDMS Database Administration*.

JCL and Commands To Run Reports

Sample z/OS and PS/390 JCL for Central Version

ADSORPTS (central version) (z/OS and PS/390)

```
//ADSORPTS EXEC PGM=ADSORPTS,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
// DD DSN=idms.loadlib,DISP=SHR
//sysctl DD DSN=idms.sysctl,DISP=SHR
//dcmmsg DD DSN=idms.sysmsg.ddldcmmsg,DISP=SHR
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
DMCL=dmcl-name
DICTNAME=apldict
Put other SYSIDMS parameters, as appropriate, here
/*
//SYSIPT DD *
Put ADSORPTS parameters, as appropriate, here
/*
/*
```

Sample z/OS and PS/390 JCL for Local Mode ADSORPTS (local mode) (z/OS and PS/390)

```
//ADSORPTS EXEC PGM=ADSORPTS,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
// DD DSN=idms.loadlib,DISP=SHR
//dictdb DD DSN=idms.apldict.ddldml,DISP=SHR
//dloddb DD DSN=idms.apldict.ddldclod,DISP=SHR
//dcmmsg DD DSN=idms.sysmsg.ddldcmmsg,DISP=SHR
//sysjrn1 DD DUMMY
//SYSPCH DD sypch-def
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
DMCL=dmcl-name
DICTNAME=apldict
Put other SYSIDMS parameters, as appropriate, here
/*
//SYSIPT DD *
Put ADSORPTS parameters, as appropriate, here
/*
/*
```

<i>idms.dba.loadlib</i>	Data set name of the load library containing the DMCL and database name table load modules
-------------------------	--

<i>idms.loadlib</i>	Data set name of the load library containing the CA IDMS executable modules
<i>sysctl</i>	DDname of the SYSCTL file
<i>idms.sysctl</i>	Data set name of the SYSCTL file
<i>dcmsg</i>	DDname of the system message (DDLDM/ESAG) area
<i>idms.sysmsg.ddldcmsg</i>	Data set name of the system message (DDLDM/ESAG) area
<i>dmcl-name</i>	Name of the DMCL load module
<i>apldict</i>	Name of the application dictionary
<i>dictdb</i>	DDname of the application dictionary definition (DDLDM) area
<i>idms.apldict.ddldml</i>	Data set name of the application dictionary definition (DDLDM) area
<i>dloddb</i>	DDname of the application dictionary definition load (DDLDM) area
<i>idms.apldict.ddldclod</i>	Data set name of the application dictionary definition load (DDLDCLOD) area
<i>sysjml</i>	DDname of the tape journal file

Sample z/VSE JCL for Central Version

ADSORPTS (z/VSE)

```
// UPSI      b                if specified in the IDMSOPTI module
// DLBL      userlib
// EXTENT    ,nnnnnn
// LIBDEF    *,SEARCH=(userlib.cdmslib)
// EXEC      ADSORPTS
SYSIDMS parameters
control statements
```

<i>b</i>	appropriate 1- through 8-character UPSI bit switch, as specified in the IDMSOPTI module
<i>nnnnnn</i>	volume serial number of the library
<i>userlib</i>	filename of the CA IDMS/DB library
<i>userlib.cdmslib</i>	file-id of the CA IDMS/DB sublibrary
<i>SYSPCH definition</i>	See "ADSA migration syntax considerations" below
<i>SYSIDMS parameters</i>	A list of SYSIDMS parameters for this job

Sample z/VSE JCL for Local Mode

To execute ADSORPTS in local mode, perform the following steps:

1. Remove the UPSI specification.
2. Add the following statements before the EXEC statement:

```
// DLBL      dictdb,'idms.appldict.ddldml',,DA
// EXTENT   sys015,nnnnnn
// ASSGN    sys015,DISK,VOL=nnnnnn,SHR
// DLBL      dloddb,'idms.appldict.ddldclod',,DA
// EXTENT   sys017,nnnnnn
// ASSGN    sys017,DISK,VOL=nnnnnn,SHR
// DLBL      dmsgdb,'idms.system.ddldcmsg',,DA
// EXTENT   sys016,nnnnnn
// ASSGN    sys016,DISK,VOL=nnnnnn,SHR
// TLBL     sys009,'idms.tapejrnl',,nnnnnn,,f
// ASSGN    sys009,TAPE,VOL=nnnnnn
```

<i>idms.appldict.ddldml</i>	= file-id of the data dictionary DDLML area
<i>idms.appldict.ddldclod</i>	= file-id of the data dictionary load area
<i>idms.system.ddldcmsg</i>	= file-id of the data dictionary message area
<i>idms.tapejrnl</i>	= file-id of the tape journal file
<i>dictdb</i>	= filename of the data dictionary DDLML area
<i>dloddb</i>	= filename of the data dictionary load area (DDLDCLOD)
<i>dmsgdb</i>	= filename of the data dictionary message area (DDLDM/ESAG)
<i>f</i>	= file number of the tape journal file
<i>nnnnnn</i>	= volume serial number
<i>sys009</i>	= logical unit assignment for the tape journal file
<i>sys015</i>	= logical unit assignment for the data dictionary DDLML area
<i>sys016</i>	= logical unit assignment for the data dictionary message area
<i>sys017</i>	= logical unit assignment for the data dictionary load area

Sample z/VM Commands for Central Version

ADSORPTS (z/VM)

```
FILEDEF SYSCTL DISK sysctl frame a
FILEDEF SYSLST PRINTER
FILEDEF SYSIDMS DISK sysidms input a
FILEDEF SYSIPT DISK rpts input a
GLOBAL LOADLIB idmslib
OSRUN ADSORPTS
```

<i>sysctl frame a</i>	filename, filetype, and filemode of the SYSCTL file for the CV to run against
<i>sysidms input a</i>	filename, filetype, and filemode of the file containing the SYSIDMS input parameters
<i>rpts input a</i>	file identifier of the file containing ADSORPTS source statements
<i>idmslib</i>	filename of the CA IDMS/DB LOADLIB library

Sample z/VM Commands for Local Mode

To execute ADSORPTS in local mode, add the following commands before the OSRUN command:

```
FILEDEF sysjml TAP1 SL VOLID nnnnn (RECFM VB LRECL lll BLKSIZE bbb
FILEDEF dictdb DISK dictdb addr
FILEDEF dloddb DISK dloddb addr
FILEDEF dmsgdb DISK dmsgdb addr
```

<i>bbb</i>	= block size of the tape journal file
<i>dictdb</i>	= ddname of the data dictionary DDLML area
<i>dictdb addr</i>	= disk address of the data dictionary DDLML area; for example, 500
<i>dloddb</i>	= ddname of the data dictionary load area (DDLDCLOAD)
<i>dloddb addr</i>	= disk address of the data dictionary load area; for example, 500
<i>dmsgdb</i>	= ddname of the data dictionary message area (DDLDM/ESAG)
<i>dmsgdb addr</i>	= disk address of the data dictionary message area
<i>lll</i>	= record length of the tape journal file

Specifying Central Version or Local Mode

To specify whether ADSORPTS executes under central version or in local mode, take one of the following actions:

- Specify either `CVMACH=dc/ucf-machine-name` (for central version) or `*LOCAL*` (for local mode) as the first statement to be submitted to ADSORPTS.
Dc/ucf-machine-name is the 1- through 8-character user identifier of the z/VM virtual machine in which the CA IDMS/DC or DC/UCF (DC/UCF) system is executing.
- Link edit ADSORPTS with an IDMSOPTI module that specifies either `CVMACH=dc/ucf-machine-name` (for central version) or `CENTRAL=NO` (for local mode).
Note: For more information about the instructions to create an IDMSOPTI module, see the *CA IDMS System Operations Guide*.
- Code `PARM='CVMACH=dc/ucf-machine-name'` or `PARM='*LOCAL*'` on the OSRUN command used to invoke the compiler. This option is not allowed if the OSRUN command is issued from a z/VM EXEC program; however, it is allowed if the OSRUN command is issued from a System Product interpreter (REXX) or EXEC 2 program.

Note: For more information about central version and local mode operations in the z/VM environment, see the *Installation and Maintenance Guide—z/VM*.

Appendix C: Dialog Statistics

This section contains the following topics:

[Overview](#) (see page 611)

[Collecting Selected Statistics](#) (see page 611)

[Enabling Dialog Statistics](#) (see page 615)

[Selecting Dialogs](#) (see page 616)

[Setting a Checkpoint Interval](#) (see page 617)

[Collecting and Writing Statistics](#) (see page 617)

[Statistics Reporting](#) (see page 618)

Overview

The CA ADS dialog statistics feature allows collection of runtime statistics about dialog and overhead activity. (Overhead activity is not directly attributable to any dialog. Overhead activity occurs once at the beginning and once at the end of an application.) Statistics are collected for each logical terminal through which the application is executed.

The following aspects of dialog statistics are discussed in this appendix:

- Collecting selected statistics
- Enabling dialog statistics
- Selecting dialogs for collection of individual statistics
- Setting a checkpoint interval, after which accumulated statistics are written to the log file at runtime
- Collecting and writing statistics at runtime
- Dialog statistics reporting

Note: For more information about statistics-related DCMT commands described later in this appendix, see the *CA IDMS System Tasks and Operator Commands Guide*.

Collecting Selected Statistics

Individual sets of dialog statistics can be collected for selected dialogs or for every dialog that executes during an application. If statistics are collected for selected dialogs, one additional set of statistics is collected for all the nonselected dialogs.

Transaction Statistics Block Fields

The following table lists the sets of CA IDMS/DB and CA IDMS/DC transaction statistics that can be collected for each dialog and for overhead activity.

Note: For more information about transaction statistics, see the *CA IDMS System Tasks and Operator Commands Guide*.

Type of information	Fields
IDENTIFICATION INFORMATION	Transaction Statistics Block identifier DC user identifier DC logical terminal identifier Dialog identifier Date that BIND command was issued Time that BIND command was issued
CA IDMS/DC STATISTICS	Number of programs called Number of programs loaded Number of terminal reads Number of terminal writes Number of terminal errors Number of storage acquisitions Number of scratch gets Number of scratch puts Number of scratch deletes Number of queue gets Number of queue puts Number of queue deletes Number of get time requests Number of set time requests Number of database calls Max words used in stack User mode time System mode time Wait time Task storage high-water mark Total number of free storage requests

Type of information	Fields
IDMS-DB STATISTICS	Number of pages read Number of pages written Number of pages requested Number of CALC records stored with no overflow Number of CALC records stored with overflow Number of VIA records stored with no overflow Number of VIA records stored with overflow Number of records requested Number of records current of run unit Number of fragments stored Number of records relocated Total number of locks Number of select locks Number of update locks

CA ADS Statistics Block Fields

The following table lists the sets of CA ADS statistics that can be collected for each dialog.

Type of information	Fields
IDENTIFICATION INFORMATION	CA ADS Statistics Block identifier DC user identifier DC logical terminal identifier Dialog identifier Date that Transaction Statistics Block BIND command was issued Time that Transaction Statistics Block BIND command was issued Dialog version number

Type of information	Fields
STATISTICS FOR EXPLICITLY CODED CONTROL COMMANDS	Number of DISPLAY commands Number of DISPLAY CONTINUE commands Number of INVOKE commands Number of LINK TO DIALOG commands Number of LINK TO PROGRAM commands Number of RETURN commands Number of RETURN CONTINUE commands Number of TRANSFER commands Number of LEAVE ADS commands Number of LEAVE APPLICATION commands Number of ABORT commands
STATISTICS FOR IMPLICITLY GENERATED CONTROL COMMANDS	Number of DISPLAY commands Number of INVOKE commands Number of LINK TO DIALOG commands Number of LINK TO PROGRAM commands Number of RETURN commands Number of RETURN CONTINUE commands Number of TRANSFER commands Number of LEAVE ADS commands Number of LEAVE APPLICATION commands Number of ABORT commands

Type of information	Fields
DIALOG EXECUTION STATISTICS	Number of premap process executions Number of response process executions Number of statistics accumulation calls Number of explicit scratch gets Number of explicit scratch puts Number of explicit scratch deletes Number of WRITE PRINTER commands Number of PUT NEW DETAIL commands Number of PUT CURRENT DETAIL commands Number of GET DETAIL commands Size of Fixed Dialog Block (FDB) Size of Variable Dialog Block (VDB) Highest link level at which a dialog was executed Lowest link level at which a dialog was executed
STATISTICS FOR RECORD BUFFER BLOCK (RBB) USAGE	Number of times RBBs put to scratch Most RBB storage used (all dialogs) RBB free space when most storage used Least RBB storage used (all dialogs) RBB free space when least storage used Most RBB space acquired for a dialog Least RBB space acquired for a dialog Highest number of RBBs used Lowest number of RBBs used

Enabling Dialog Statistics

Dialog statistics can be collected only if task and transaction statistics collection is enabled. Task statistics are enabled at system generation time. Transaction statistics can be enabled at either system generation or runtime in the following manner:

- To enable transaction statistics at **system generation time**, use the STATISTICS parameter in the SYSTEM statement, specifying TASK, WRITE, and TRANSACTION.
- To enable transaction statistics at **runtime**, use the DCMT VARY STATISTICS TRANSACTION command.

The DCMT VARY ADSO STATISTICS command is used to enable or disable dialog statistics and to specify whether individual statistics are collected for selected dialogs or for all dialogs, as follows:

- **DCMT VARY ADSO STATISTICS ON ALL DIALOGS** enables dialog statistics and specifies that sets of statistics are to be collected for overhead activity and for each dialog that is executed during an CA ADS application.
- **DCMT VARY ADSO STATISTICS ON SELECTED DIALOGS** enables dialog statistics and specifies that sets of statistics are to be collected for overhead activity and for each selected dialog that is executed during an CA ADS application. One additional set of statistics is collected to accumulate statistics for all nonselected dialogs.
- **DCMT VARY ADSO STATISTICS OFF** disables the dialog statistics feature.

If no DCMT VARY ADSO STATISTICS command is issued prior to the execution of an application, the runtime system uses the default specification established at system generation.

Selecting Dialogs

The DCMT VARY PROGRAM command is used to select or deselect dialogs for individual statistics collection, as follows:

- **DCMT VARY PROGRAM dialog-name ADSO STATISTICS ON** selects the named dialog for individual statistics collection. At runtime, if dialog statistics are enabled, individual statistics are collected for the dialog when it executes.
- **DCMT VARY PROGRAM dialog-name ADSO STATISTICS OFF** deselects a dialog from individual statistics collection. At runtime, if dialog statistics are enabled in the SELECTED DIALOGS mode, individual statistics for the dialog are not collected; however, one set of statistics is collected for all nonselected dialogs. If dialog statistics are enabled in the ALL DIALOGS mode, individual statistics are collected for the dialog when it executes, even if it is not selected.

If no DCMT VARY PROGRAM command with the STATISTICS parameter is issued for a dialog prior to the execution of an application, the runtime system uses the specification established at system generation.

The DCMT VARY ADSO STATISTICS and the DCMT VARY PROGRAM commands can be issued in any order.

Setting a Checkpoint Interval

The DCMT VARY ADSO STATISTICS command is used to set a checkpoint interval, which determines when the collected statistics are written to the system log, as follows:

- **DCMT VARY ADSO STATISTICS CHECKPOINT INTERVAL *checkpoint-interval-number*** specifies that statistics for all dialogs are written to the log once at every *checkpoint-interval-number* statistics accumulations. Additionally, statistics are written to the log when the application terminates. Note that CHECKPOINT INTERVAL 0 is equivalent to CHECKPOINT INTERVAL OFF.
- **DCMT VARY ADSO STATISTICS CHECKPOINT INTERVAL OFF** specifies that statistics are written to the log only when the application terminates.

If no DCMT VARY ADSO STATISTICS command with the CHECKPOINT INTERVAL parameter is issued prior to the execution of an application, the runtime system uses the specification established at system generation.

Collecting and Writing Statistics

At runtime, if dialog statistics are enabled, statistics for overhead activity are collected and written to the CA IDMS/DC or DC/UCF (DC/UCF) system log whenever overhead activity is performed, once at the beginning of the application and once at the end. The transaction statistics block identifier for overhead activity is either the application name or, for applications not defined using the application generator, \$ADS@@OH.

Dialog statistics are collected each time a dialog issues a control command. These statistics are not written immediately to the system log, but are accumulated in transaction and CA ADS statistics blocks (TSBs and ASBs).

The runtime system allocates TSBs and ASBs as follows:

- If dialog statistics are enabled in the ALL DIALOGS mode, one TSB and one ASB are allocated for each dialog the first time the dialog becomes operative in the application thread. The statistics block identifier for the TSB and ASB is the dialog name.
- If dialog statistics are enabled in the SELECTED DIALOGS mode, one TSB and one ASB are allocated for each selected dialog the first time the dialog becomes operative in the application thread. Additionally, one TSB and one ASB are allocated to accumulate statistics for all nonselected dialogs; the statistics block identifier for the additional TSB and ASB is \$ADS@@AO.

Dialog statistics are written to the system log each time the number of statistics accumulations equals the predefined checkpoint interval. Additionally, statistics are written to the log when the application terminates.

When dialog statistics are written to the system log, only TSBs and ASBs that contain accumulated statistics are written to the system log. The TSBs and ASBs are then initialized, and the statistics accumulations count is reset to zero.

TSBs and ASBs are freed only when the application terminates. Note, however, that during a pseudo-converse they may be written to scratch along with record buffer blocks, as directed at system generation with the `FAST MODE THRESHOLD` and `RESOURCES` parameters of the `ADSO` statement.

Statistics Reporting

DC/UCF statistics reports (SREPORTs) allow the application developer to produce reports on dialog statistics.

Statistics collected in the CA ADS statistics block can be reported on by using any of the following SREPORTs, identified by report number:

- **018**— CA ADS statistics by user id
- **019**— CA ADS statistics by dialog and version number
- **020**— CA ADS statistics by logical terminal id

Statistics collected in the transaction statistics block can be reported on by using any of the following SREPORTs:

- **011**— CA IDMS/DC transaction statistics by logical terminal id
- **021**— IDMS-DC transaction statistics by dialog and

SREPORTS are similar in format.

Sample SREPORT for Dialog Statistics

The following shows sample output from SREPORT number 019:

REPORT NO. 019	ADS STATISTICS BY DIALOG AND VERSION NUMBER - R15.0	09/19/99	PAGE 8
DIALOG NAME : ADSOAFNC	VERSION NUMBER: 1		
DATE :	91043 TIME :	09:49 USER ID :	SMT
DATE BIND :	91043 TIME BIND :	09:46 LTERM ID :	LT12011
DISPLAY COMMAND:	21 DISPLAY CONTINU:	21 INVOKES :	3 LINK TO DIALOGS: 18
LNKS TO PROGRAM:	18 RETURNS :	0 RETURN CONTINUE:	0 TRANSFERS : 18
LEAVE ADS :	0 LEAVE APPLICATN:	0 ABORTS :	0 IMPL DISPLAYS : 0
IMPL INVOKE :	0 IMPL LINK DLGS :	0 IMPL LINK PGMS :	0 IMPL RETURNS : 0
IMPL RET CONT :	0 IMPL TRANSFERS :	0 IMPL LEAVE ADS :	0 IMPL LEAVE PGMS: 0
IMPL ABORTS :	0 PREMAP PROCESS :	42 RESPONSE PROCES:	21 STAT ACCUM CALL: 99
EXPL GET SCRS :	0 EXPL PUT SCRS :	0 EXPL DEL SCRS :	0 WRTE PRINT REQS: 0
PUT NEW DETAILS:	0 PUT CUR DETAILS:	0 GET DETAILS :	0 SIZE OF FDB : 23,080
SIZE OF VDB :	836 HIGHEST LNK LEV:	1 LOWEST LNK LEVL:	1 RBB PUT TO SCR : 0
RBB STG HI MARK:	3,176 RBB FREE HI :	908 RBB STG LOW MK :	3,176 RBB FREE LOW : 908
MOST RBB ACQ :	304 LEAST RBB ACQ :	304 HICOUNT RBB USE:	1 LOCOUNT RBB USE: 1
**** DIALOG TOTAL ****			
DISPLAY COMMAND:	21 DISPLAY CONTINU:	21 INVOKES :	3 LINK TO DIALOGS: 18
LNKS TO PROGRAM:	18 RETURNS :	0 RETURN CONTINUE:	0 TRANSFERS : 18
LEAVE ADS :	0 LEAVE APPLICATN:	0 ABORTS :	0 IMPL DISPLAYS : 0
IMPL INVOKE :	0 IMPL LINK DLGS :	0 IMPL LINK PGMS :	0 IMPL RETURNS : 0
IMPL RET CONT :	0 IMPL TRANSFERS :	0 IMPL LEAVE ADS :	0 IMPL LEAVE PGMS: 0
IMPL ABORTS :	0 PREMAP PROCESS :	42 RESPONSE PROC :	21 STAT ACCUM CALL: 99
EXPL GET SCRS :	0 EXPL PUT SCRS :	0 EXPL DEL SCRS :	0 WRTE PRINT REQS: 0
PUT NEW DETAILS:	0 PUT CUR DETAILS:	0 GET DETAILS :	0 RECORD COUNT : 1

Appendix D: Application and Dialog Utilities

This section contains the following topics:

- [Overview](#) (see page 621)
- [ADSOBCOM](#) (see page 621)
- [ADSOBSYS](#) (see page 654)
- [ADSOBTAT](#) (see page 662)
- [ADSOTATU](#) (see page 671)

Overview

CA ADS provides utilities that allow the application developer to maintain applications and dialogs. The utilities are summarized in the table below and discussed in further detail throughout this appendix.

Summary of CA ADS Utilities

Utility	Purpose
ADSOBCOM	Creates, modifies, deletes, and recompiles dialogs in batch mode
ADSOBSYS	Sets up system generation parameters required by ADSOBCOM
ADSOBTAT	Modifies the task application table (TAT) in batch mode when an application is migrated from one dictionary to another
ADSOTATU	Modifies the task application table (TAT) online when an application is migrated from one dictionary to another

ADSOBCOM

ADSOBCOM, the batch dialog compiler, allows the application developer to add, modify, delete, and recompile dialogs. Batch dialog recompilation is useful when modifications are made to maps, processes, subschemas, or records that are associated with several dialogs. There is no limit to the number of dialogs that can be processed in a single run of ADSOBCOM.

Special Control Statements

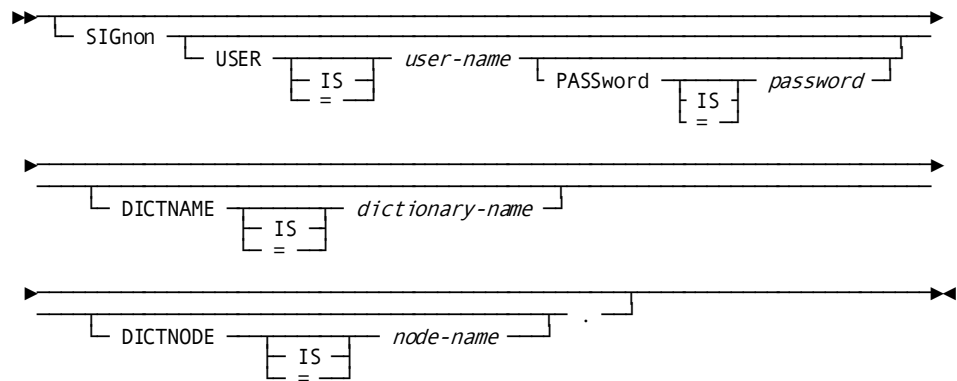
ADSOBCOM is driven by the control statements SIGNON, COMPILE, and DECOMPILE.

SIGNON

Purpose

Specifies the name and any necessary password of the DC/UCF user, as well as the dictionary in which the dialogs to be recompiled are stored.

Syntax



Parameters

USER is *user-name*

Specifies the signon user.

The equals sign (=) can be used in place of IS.

Note: USER must be the first parameter specified on the SIGNON statement.

PASSword is *password*

Specifies, when necessary, the user's DC/UCF password.

The user name and password must be supplied in order to use ADSOBCOM when dialog compiler level security is in effect. Additionally, security at the dialog level may also require that the user name and password be supplied.

DICTNAME is *dictionary-name*

Specifies the 1- to 8-character name of the data dictionary from which the dialog load module, process source code, record, map, and subschema definitions are retrieved. This is the same dictionary into which the compiled dialog load module is placed.

If no dictionary name is specified, ADSOBCOM uses the name of the primary dictionary.

DICTNODE is *node-name*

(for DDS only) Specifies the 1- to 8-character name of the DDS node that controls the data dictionary specified by DBNAME.

More information:

[Security Features](#) (see page 717)

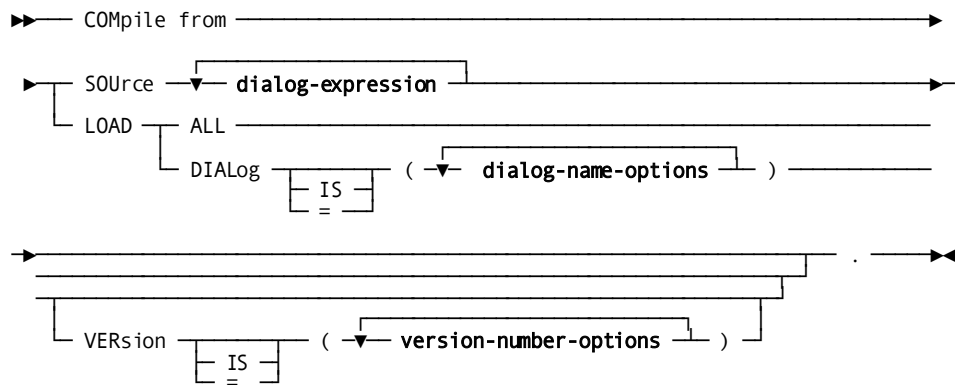
COMPILE

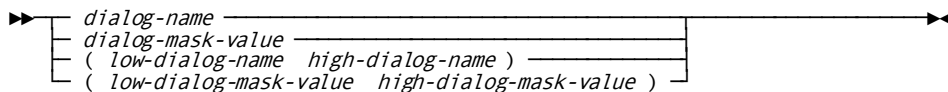
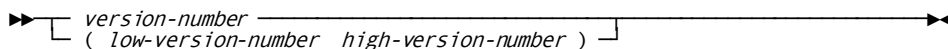
Purpose

Either specifies the dialogs to be recompiled based on information in the load module, or specifies the dialogs to be added, modified, or deleted, based on information in the dialog statements that accompany the COMPILE statement.

There is no limit to the number of COMPILE statements that can be submitted to each run of ADSOBCOM.

Syntax



Expansion of dialog-name-options**Expansion of version-number-options****Parameters****SOURCE dialog-expression**

Specifies that dialogs to be added, modified, or deleted based on information in the dialog expression.

Repeated dialog expressions can be used to process several dialogs. Each expression must end with a period.

See the explanation of *dialog-expression* on the following pages.

LOAD

Specifies that the dialogs are to be recompiled based on the information in the dialog load modules.

ALL

Specifies that all dialogs in the dictionary load area are to be recompiled.

DIALOG IS dialog-name-options

Specifies the dialogs in the dictionary load area to be recompiled. See expansion of *dialog-name-options* below.

VERSION IS version-number-options

Specifies the version numbers of the dialogs to be recompiled. See expansion of *version-number-options* below.

dialog-name

Specifies the 1 to 8-character name of a single dialog.

dialog-mask-value

Specifies any dialog with a name that matches the mask criteria.

The mask character is the asterisk (*); it matches any character. For example, DIALOG IS (DCB*****) causes all dialogs beginning with DCB to be recompiled.

If the mask contains fewer than eight characters, the remaining character positions are treated as blanks.

(low-dialog-name high-dialog-name)

Specifies all dialogs within the dialog-name range (inclusive).

Note: Parentheses are needed when using a range of values.

(low-dialog-mask-value high-dialog-mask-value)

Specifies all dialogs within the dialog-mask range (inclusive).

Note: Parentheses are needed when using a range of values.

version-number

Specifies a single version number for the selected dialogs.

(low-version-number high-version-number)

Specifies all versions of the selected dialogs within the version-number range (inclusive).

Note: Parentheses are needed when using a range of values.

The default version number is 1.

Usage*Considerations*

ADSOBCOM does not update a dialog's program definition element (PDE) to indicate that a new copy of the dialog exists in the load area. If a dialog is recompiled by ADSOBCOM and then executed during a single DC/UCF run, the application developer should update the PDE by issuing the following command:

```
DCMT VARY PROGRAM dialog-name NEW COPY
```

Note: For more information about the DCMT VARY PROGRAM command, see the *CA IDMS System Tasks and Operator Commands Guide*.

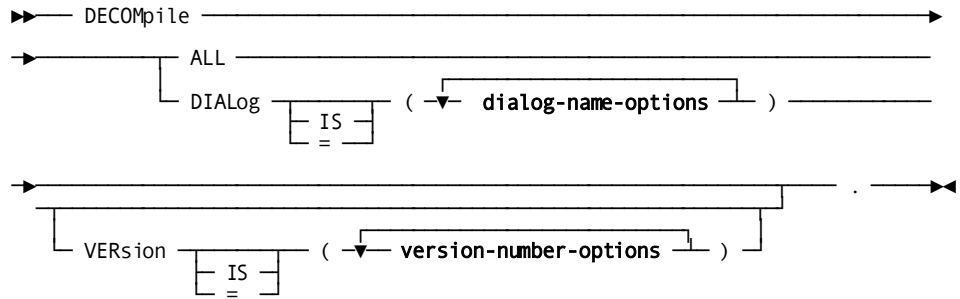
DECOMPILE

Purpose

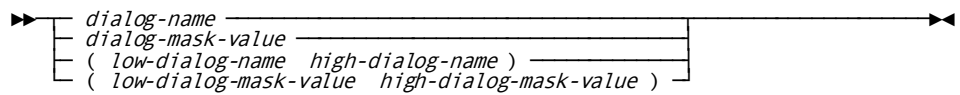
Specifies the dialogs to be decompiled based on information in the load module.

There is no limit to the number of DECOMPILE statements that can be submitted to each run of ADSOBCOM.

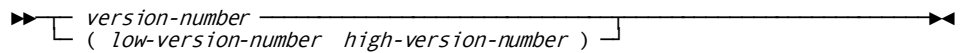
Syntax



Expansion of dialog-name-options



Expansion of version-number-options



Parameters

SOURCE dialog-expression

Specifies that dialogs to be added, modified, or deleted based on information in the dialog expression.

Repeated dialog expressions can be used to process several dialogs. Each expression must end with a period.

See the explanation of *dialog-expression* on the following pages.

LOAD

Specifies that the dialogs are to be recompiled based on the information in the dialog load modules.

ALL

Specifies that all dialogs in the dictionary load area are to be recompiled.

DIALOG is dialog-name-options

Specifies the dialogs in the dictionary load area to be recompiled. See expansion of *dialog-name-options* below.

VERSION is version-number-options

Specifies the version numbers of the dialogs to be recompiled. See expansion of *version-number-options* below.

dialog-name

Specifies the 1- to 8-character name of a single dialog.

dialog-mask-value

Specifies any dialog with a name that matches the mask criteria.

The mask character is the asterisk (*); it matches any character. For example, DIALOG IS (DCB*****) causes all dialogs beginning with DCB to be recompiled.

If the mask contains fewer than eight characters, the remaining character positions are treated as blanks.

(low-dialog-name high-dialog-name)

Specifies all dialogs within the dialog-name range (inclusive).

Note: Parentheses are needed when using a range of values.

(low-dialog-mask-value high-dialog-mask-value)

Specifies all dialogs within the dialog-mask range (inclusive).

Note: Parentheses are needed when using a range of values.

version-number

Specifies a single version number for the selected dialogs.

(low-version-number high-version-number)

Specifies all versions of the selected dialogs within the version-number range (inclusive).

Note: Parentheses are needed when using a range of values.

The default version number is 1.

Usage***Considerations***

ADSOBCOM does not update a dialog's program definition element (PDE) to indicate that a new copy of the dialog exists in the load area. If a dialog is recompiled by ADSOBCOM and then executed during a single DC/UCF run, the application developer should update the PDE by issuing the following command:

```
DCMT VARY PROGRAM dialog-name NEW COPY
```

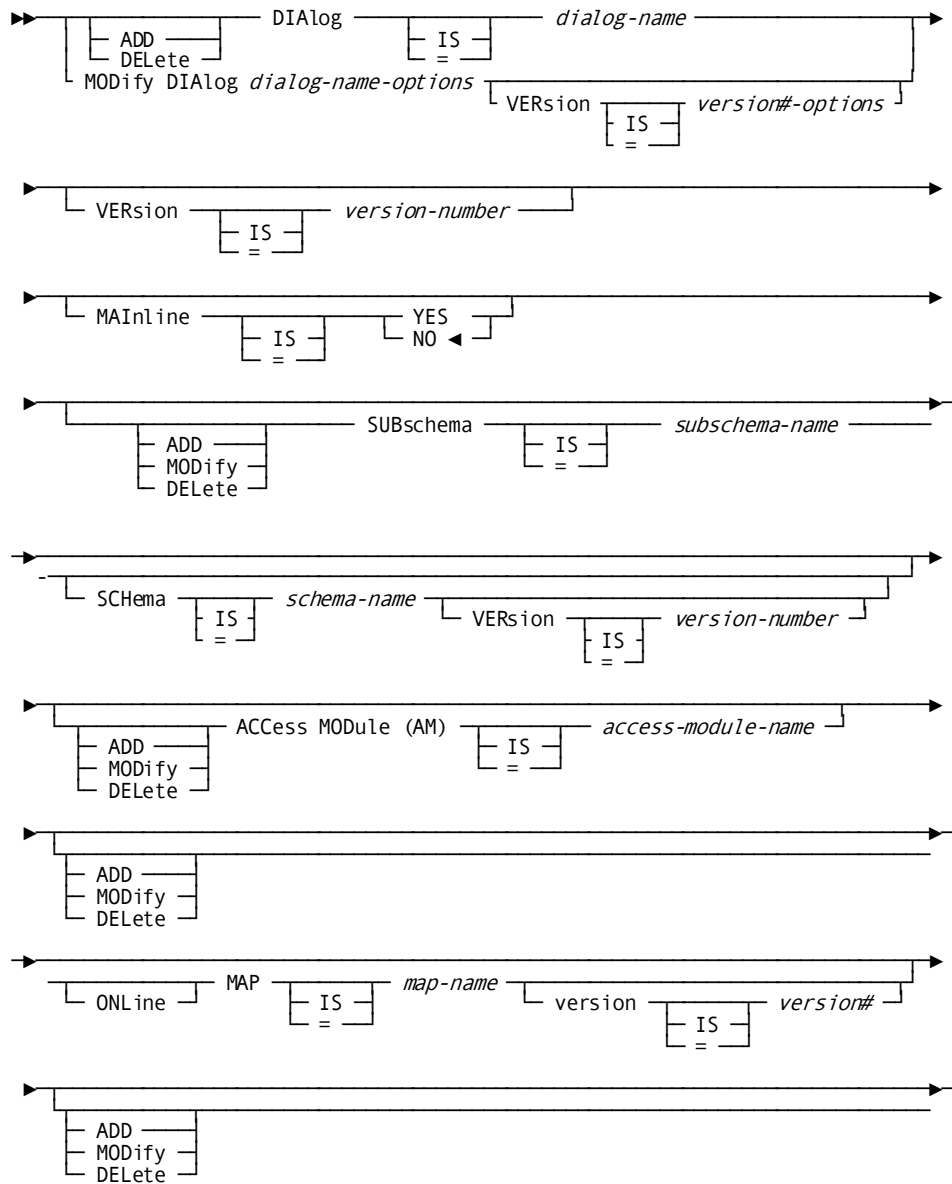
Note: For more information about the DCMT VARY PROGRAM command, see the *CA IDMS System Tasks and Operator Commands Guide*.

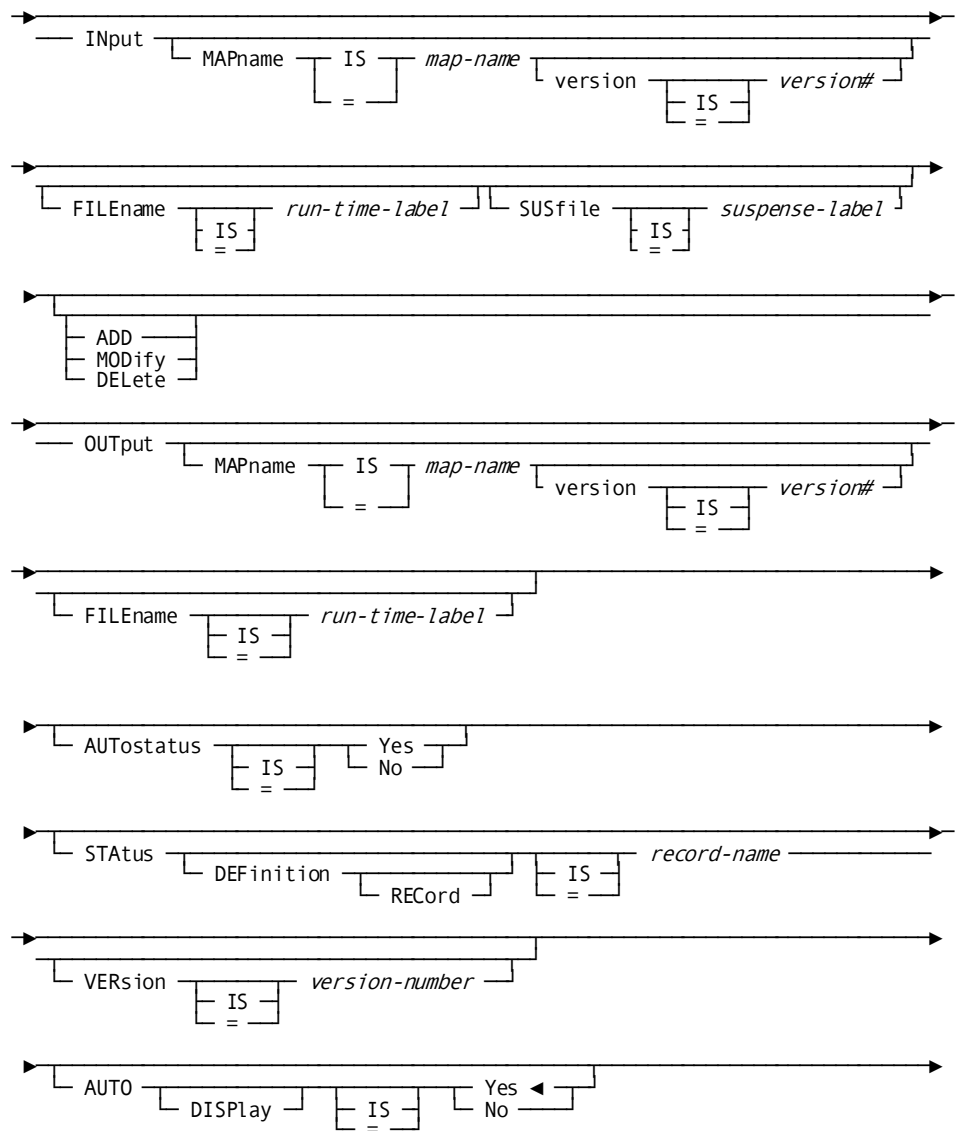
Dialog-expression

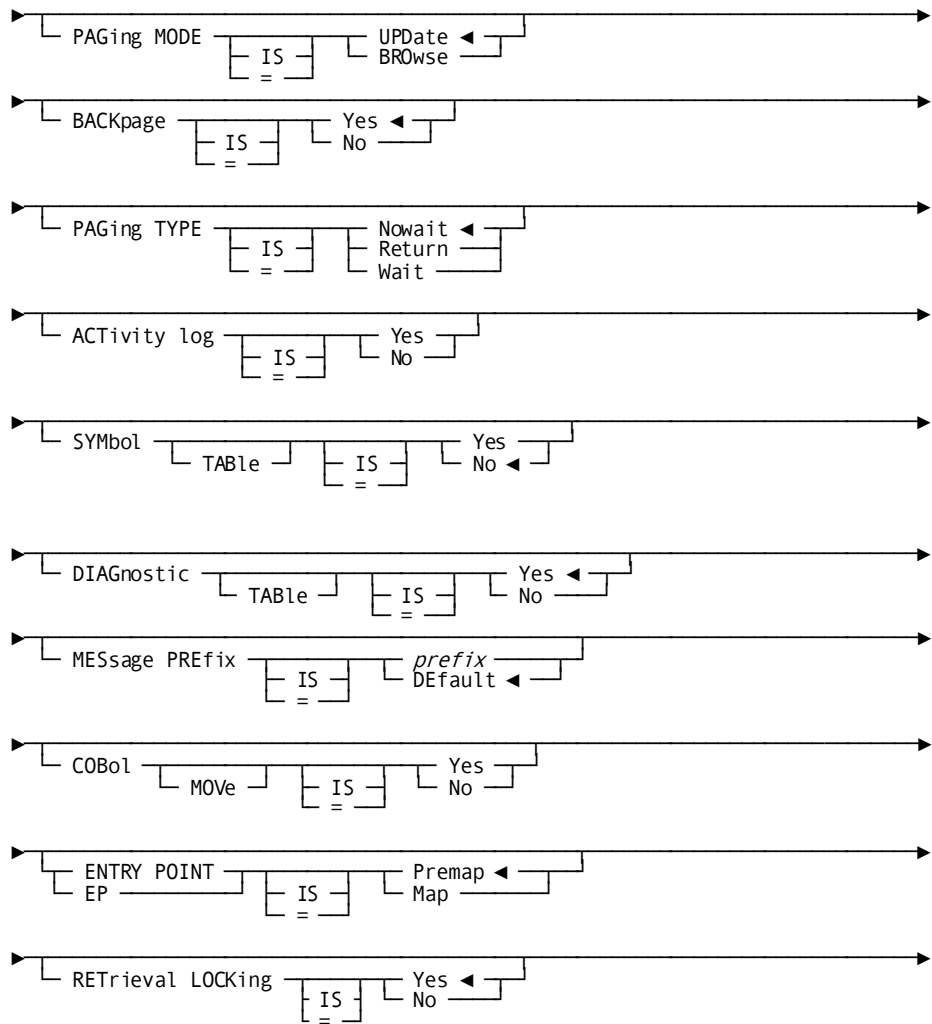
Purpose

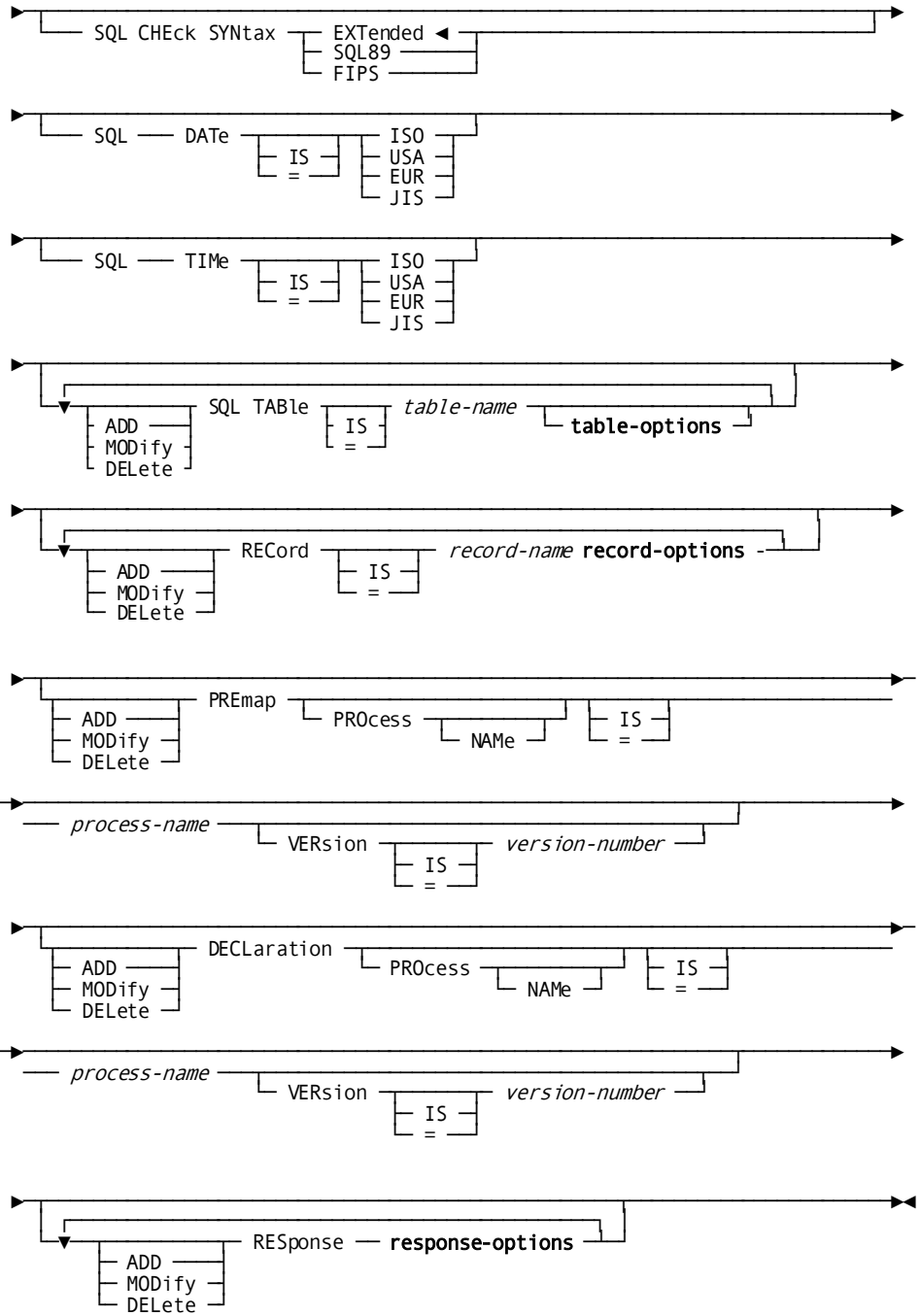
Dialog-expression is used to specify those dialogs which are to be added, modified, and deleted.

Syntax

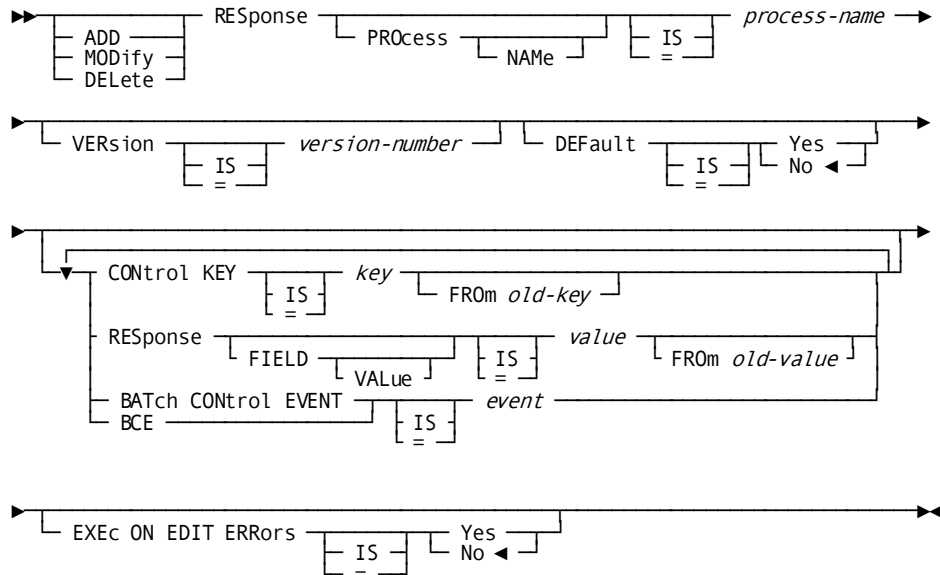




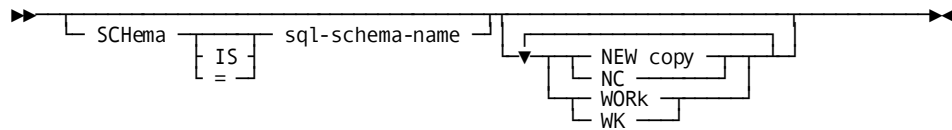




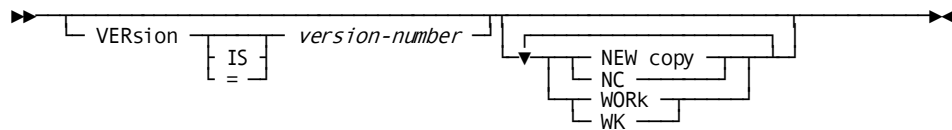
Expansion of response-options



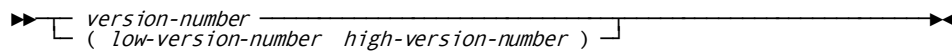
Expansion of table-options



Expansion of record-options



Expansion of version#-options



Parameters

ADD

Specifies that a dialog is to be added to the data dictionary.

ADD is the default if the named dialog does not exist in the data dictionary.

MODify

Specifies that an existing dialog is to be modified.

MODIFY is the default if the named dialog exists in the data dictionary.

DElete

Specifies that an existing dialog is to be deleted.

When the action is DELETE, only the dialog name and version number can be specified in the dialog expression.

DIAlog is *dialog-name*

Specifies the 1- to 8-character name of the dialog being added, modified, or deleted.

The dialog name must begin with an alphabetic or national (@, #, and \$) character and cannot contain embedded blanks.

The equals sign (=) can be used in place of IS.

VERsion is *version-number*

Specifies the version number (in the range 1 through 9999) of the dialog being added, modified, or deleted.

The default version number is 1.

The equals sign (=) can be used in place of IS.

MAInline is Yes/No

Specifies whether the 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.

No is the default when neither Yes or No is specified.

ADD

Specifies that the subschema specification is to be added.

ADD is the default if no subschema is associated with the dialog.

MODify

Specifies that the existing subschema specification is to be replaced by a new subschema specification.

MODIFY is the default if a subschema is associated with the dialog.

DElete

Specifies that the subschema specification is to be deleted.

If the action is DELETE, the SCHEMA clause cannot be specified.

SUBSchema is *subschema-name*

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

The equals sign (=) can be used in place of IS.

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

SCHEMA is *schema-name*

Specifies the 1- to 8-character name of the schema.

A schema name must be specified if the named subschema is associated with more than one schema or version of a schema. If the named subschema is associated with only one schema and version, SCHEMA defaults to the name of that schema.

The equals sign (=) can be used in place of IS.

VERSION is *version-number*

Specifies the version number (in the range 1 through 9999) of the named schema.

The equals sign (=) can be used in place of IS.

If no version number is specified, VERSION defaults to the version of the named schema that was defined most recently.

ADD

Specifies that the access module specification is to be added.

MODify

Specifies that the existing access module specification is to be replaced by a new access module specification.

DElete

Specifies that the access module specification is to be cleared to spaces.

ACCESS MODULE

Sets the access module name which is used at runtime to satisfy the IDMS/DB request of the dialog.

(AM) is *access-module-name*

Specifies the 1- to 8-character name of the access module associated with the current dialog.

The dialog can override this specification at runtime by issuing a SET ACCESS MODULE statement.

When the access module name is not specified, the name defaults to the dialog name.

ADD

Specifies that a map specification is to be added to the dialog.

ADD is the default if no map of the type specified (online, input, or output) is associated with the dialog.

MODify

Specifies that the existing map specification is to be replaced by a new map specification.

MODIFY is the default if a map of the type specified (online, input, or output) is already associated with the dialog.

DElete

Specifies that the map definition is to be dissociated from the dialog.

If DELETE is specified, the version number of the MAPNAME clause cannot be specified.

ONLine/INput/OUTput

Specifies the type of map.

ONLINE is the default when no other map type is specified.

A dialog associated with an online map cannot be associated with an input or output file map. A dialog can be associated with both an input and an output file map by coding multiple ADD ... MAPNAME clauses. A dialog not associated with a map is called a mapless dialog and can be executed in both batch and online environments.

MAPname is *map-name*

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

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

VERsion is *version-number*

Specifies the version number (in the range 1 through 9999) of the named map.

The equals sign (=) can be used in place of IS.

If no version number is specified and the map is being added to a dialog, or the dialog is being associated with a different map, the version defaults to 1. Otherwise, the version number defaults to the version of the map currently associated with the dialog.

FILEname is *runtime-label*

(Batch only) Specifies the z/OS ddname (z/VSE filename, z/VM ddname) of the input or output file added or modified.

The equals sign (=) can be used in place of IS.

The runtime label must be specified either during dialog definition or at runtime.

The runtime control statement overrides the default specified during dialog definition.

SUSfile is *suspense-label*

(Batch only) Specifies the z/OS ddname (z/VSE filename, z/VM ddname) of the suspense file for input file maps only.

The equals sign (=) can be used in place of IS.

If a suspense file is maintained for the dialog at runtime, the label must be specified either during dialog definition or at runtime. The runtime control statement overrides the default specified during dialog definition.

AUTostatus is *Yes/No*

Specifies whether the autostatus facility is used when the current dialog executes.

The default 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, the initial setting cannot be changed.

STATus DEFinition RECOrd is *record-name*

Specifies the 1 to 32-character name of the status definition record.

The specified record must be defined in the data dictionary. If no record name is specified, STATUS DEFINITION RECORD defaults to the name of the status definition record defined at DC/UCF system generation.

VERsion is *version-number*

Specifies the version number (in the range 1 through 9999) of the named status definition record.

If a version number is not specified, VERSION defaults to the system default version number, as specified in the OOK record at system generation.

If no system default version is specified in the OOK record, VERSION defaults to 1.

The equals sign (=) can be used in place of IS.

AUTO DISPLAY is *Yes/No*

Specifies whether the first page of pageable map is displayed automatically.

A DISPLAY statement must be coded in the dialog's premap process to display the first page.

YES is the default when neither YES or NO is specified.

PAGing MODE is UPDate/BROwse

Specifies whether the user can modify data fields on a map during a map paging session.

UPDATE is the default setting for the paging mode option.

BACKpage is Yes/No

Specifies whether the user can page backward in a map paging session.

YES is the default setting for the backpage option.

PAGing TYPE is Nowait/Return/Wait

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

NOWAIT is the default setting for the paging type option.

These three paging session dialog options can be specified only if the dialog is associated with a pageable map.

The following combination of paging session dialog options cannot be specified: PAGING MODE IS UPDATE, BACKPAGE IS NO, and PAGING TYPE IS NOWAIT.

ACTivity log is Yes/No

Specifies whether 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.

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

SYMBOL TABLE is Yes/No

Specifies whether 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.

Note: For more information about the online debugger, see the *CA IDMS Online Debugger Guide*.

NO is the default setting for the symbol table option.

DIAGnostic TABLE is Yes/No

Specifies whether the dialog load module contains diagnostic tables (line number tables and offset tables).

Diagnostic tables facilitate the testing and debugging of a dialog. If a process 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.

YES is the default setting for the diagnostic table option.

The setting must be YES if the symbol table setting is YES. Also, during the testing of a dialog, the diagnostic table setting should be YES.

Once a dialog has been tested thoroughly, the diagnostic table setting should be NO 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.

MESSAge PREFIX is

Clause introducing a message prefix for a dialog.

The equals sign (=) can be used in place of IS.

prefix

Specifies a user-supplied 2-character alphanumeric message prefix for the dialog.

DEfault

Specifies that the dialog uses the default message prefix.

DEFAULT is the default setting when the message prefix is not specified.

COBoL MOVE is Yes/No

Specifies whether the rules of COBOL or CA ADS are used in the conversion between data types and in the rounding or truncation of the results of arithmetic and assignment commands.

The default setting for the COBOL MOVE option is defined at DC/UCF system generation. The system generation default is NO.

ENTRY POINT is

Clause introducing the entry point into the dialog when the dialog begins execution at runtime.

EP can be used in place of ENTRY POINT; the equals sign (=) can be used in place of IS.

Premap

Specifies that the dialog begins with its premap process.

PREMAP is the default when no other entry point is specified.

Map

Specifies that the dialog begins with its first mapping operation (mapout for online dialogs, mapin for batch dialogs).

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

RETRetrieval LOCKing is Yes/No

Specifies whether or not the dialog will cause record locks to be held for database records.

YES, the default, specifies that database record retrieval locks will be held on behalf of run units started by the dialog.

SQL CHEck SYNTax

Specifies the SQL standard you are enforcing. The default is CA IDMS extended ANSI-standard SQL. CA ADS supports the following SQL standards:

- EXTended
- SQL89
- FIPS

Note: For more information about SQL standards, see the *CA IDMS SQL Reference Guide*.

SQL DATe is

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

SQL TIME is

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

Note: For more information on date/time representations, see the *CA IDMS SQL Reference Guide*.

ADD

Specifies that the SQL table specification is to be added.

MODify

Specifies that the existing SQL table specification is to be replaced by a new SQL table specification.

DElete

Specifies that the SQL table specification is to be cleared to spaces.

SQL TABLE is *table-name*

Specifies the name of the SQL table assigned the new copy attribute and/or the work record attribute.

table-options

See expansion of *table-options* below.

ADD

Specifies that a new copy/work record specification is to be added to the dialog.

ADD is the default if the named new copy/work record is not associated with the dialog.

MODify

Specifies that a new copy/work record specification of a dialog is to be modified.

MODIFY is the default if the named new copy/work record is already associated with the dialog.

DElete

Specifies that a new copy/work record specification of a dialog is to be deleted.

If the action is DELETE, the VERSION specification is optional, and the NEW COPY and WORK specifications cannot be included.

RECORD is *record-name record-options*

Specifies the name of the record assigned the new copy attribute and/or the work record attribute.

See expansion of *record-options* below.

ADD

Specifies that a premap process is to be added to the dialog.

ADD is the default if no premap process is associated with the dialog.

MODify

Specifies that a new premap process is to replace the existing premap process.

MODIFY is the default if a premap process is already associated with the dialog.

DElete

Specifies that the premap process is to be deleted from the dialog.

If the action is DELETE, the version number cannot be specified.

PREmap PROcess NAME is *process-name*

Specifies the 1- to 32-character name of the process source module associated with the dialog as a premap process.

PROCESS and NAME are optional keywords; the equals sign may be used in place of IS.

Note: The specified process source module must exist in the data dictionary.

VERsion is *version-number*

Specifies the version number (in the range 1 through 9999) of the named process source module.

The equals sign (=) may be used in place of IS.

The default version number is the system default version number, as specified in the OOK record at system generation. If no system default version number is specified in the OOK record, the default version number is 1.

ADD

Specifies that a premap process is to be added to the dialog.

ADD is the default if no premap process is associated with the dialog.

MODify

Specifies that a new premap process is to replace the existing premap process.

MODIFY is the default if a premap process is already associated with the dialog.

DElete

Specifies that the premap process is to be deleted from the dialog.

If the action is DELETE, the version number cannot be specified.

DECLARATION PROcess NAME is *process-name*

Specifies the 1- to 32-character name of the process source module associated with the dialog as a premap process.

PROCESS and NAME are optional keywords; the equals sign may be used in place of IS.

Note: The specified process source module must exist in the data dictionary.

VERsion is *version-number*

Specifies the version number (in the range 1 through 9999) of the named process source module.

The equals sign (=) may be used in place of IS.

The default version number is the system default version number, as specified in the OOK record at system generation. If no system default version number is specified in the OOK record, the default version number is 1.

ADD

Specifies that a response process is to be added to the dialog.

ADD is the default if the named response process is not already associated with the dialog.

ADD can be used to define duplicate response processes, in which the same response process is associated with several control keys and/or response field values. In the example shown below, response process RP1 is associated with control keys PF1, PF2, and PF3, and with response field values ADD and MOD:

```
ADD RESPONSE PROCESS RP1 CONTROL KEY PF1 RES VALUE ADD
ADD RESPONSE PROCESS RP1 CONTROL KEY PF2 RES VALUE MOD
ADD RESPONSE PROCESS RP1 CONTROL KEY PF3
```

MODify

Specifies that a response process of the dialog is to be modified.

MODIFY is the default if the named response process is already associated with the dialog.

In the example shown below, the control key specification for nonduplicate response process RP1 is modified to PF2:

```
MODIFY RESPONSE PROCESS RP1 CONTROL KEY PF2
```

To modify a duplicate response process, the application developer must specify which occurrence of the duplicate response process is being modified.

To modify the control key associated with the response process, the application developer specifies the FROM parameter of the CONTROL KEY specification. In the example shown below, the control key ENTER is changed to PA1 for duplicate response process RP1:

```
MODIFY RESPONSE PROCESS RP1 CONTROL KEY PA1 FROM ENTER
```

To modify the response field value associated with the response process, the application developer specifies the FROM parameter of the RESPONSE FIELD VALUE specification. In the example shown below, the response field value MOD is changed to ADD for duplicate response process RP1:

```
MODIFY RESPONSE PROCESS RP1 RES VALUE ADD FROM MOD
```

To modify the EXECUTE ON EDIT ERRORS specification associated with the response process, the application developer specifies either the CONTROL KEY or RESPONSE FIELD VALUE parameter. In the example shown below, the EXECUTE ON EDIT ERRORS specification is set to YES for the occurrence of duplicate response process RP1 that is associated with the ENTER key:

```
MODIFY RESPONSE PROCESS RP1 CONTROL KEY ENTER
EXECUTE ON EDIT ERRORS YES
```

DElete

Specifies that a response process of the dialog is to be deleted.

If the action is DELETE, the version number is optional and the EXEC ON EDIT ERRORS specifications cannot be included.

An occurrence of a duplicate response process is deleted by specifying the CONTROL KEY or RESPONSE FIELD VALUE parameter. The example shown below deletes the occurrence of duplicate response process RP1 that is associated with the response field value ADD:

```
DELETE RESPONSE PROCESS RP1 RES VALUE ADD
```

RESponse *response-options*

See expansion of *response-options* below.

Expansion of response-options**RESponse PROcess NAME**

Specifies the 1- to 32-character name of the process source module associated with the dialog as a response process.

Note: The specified source module must exist in the data dictionary.

VERsion is *version-number*

Specifies the version number (in the range 1 through 9999) of the named process source module.

The default version number is the system default version number, as specified in the OOAK record at system generation. If no system default version number is specified in the OOAK record, the default version number is 1.

The equals sign (=) may be used in place of IS.

DEFault is Yes/No

Specifies whether the response process defined 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 values.

NO is the default specification.

If DEFAULT is NO, a control key, a response field value, a response field value, or a batch control event for the response process must be specified. If DEFAULT is YES, these specifications are optional.

CONtrol KEY is *key*

Specifies a user-defined control key that initiates the response process at runtime.

The equals sign (=) can be used in place of IS.

Key can also be specified to identify an occurrence of a duplicate response process, as described under ADD/MODIFY/DELETE RESPONSE PROCESS above.

Valid control key specifications are ENTER, CLEAR, PA1 through PA3, PF1 through PF24, FWD, BWD, and HDR. 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 for paging forward and backward, respectively. If FWD and 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.

FRom *old-key*

Identifies the occurrence of a duplicate response process whose associated control key specification is being modified, as described under ADD/MODIFY/DELETE RESPONSE PROCESS above.

RESponse FIEld VALue is *value*

Specifies a response name associated with the response process.

Value can also be specified to identify an occurrence of a duplicate response process, as described under ADD/MODIFY/DELETE RESPONSE PROCESS above.

The equals sign (=) may be used in place of IS.

When a control key value or a response field value of a response process needs to be dissociated from the response, a blank value(' ') can be used, as in the following example:

```
MOD RES PRO response-name VER 1 RES VALUE ' '
```

FRom *old-value*

Identifies the occurrence of a duplicate response process whose associated response field value specification is being modified, as described under ADD/MODIFY/DELETE RESPONSE PROCESS above.

BATCh CONTROL EVENT is event

Specifies a batch control event that initiates the response process at runtime.

BCE can be used in place of BATCH CONTROL EVENT; the equals sign (=) can be used in place of IS.

Valid Batch Control Events

- **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 physical input-error condition. In CA ADS Batch, output errors cause the runtime system to terminate the application.

Batch control events can be specified only for batch dialogs. Control keys can be specified only for online dialogs.

EXEc ON EDIt ERRors is

Introduces whether processing continues if a automatic editing encounters map input errors.

The equals sign (=) can be used in place of IS.

Yes

Specifies that the response process executes even if the map contains input errors.

No

Specifies that the response process is not executed if the map contains input errors. The user must correct all map fields that are in error before processing continues.

NO is the default when neither YES or NO is specified.

Expansion of table-options**SCHema is sql-schema-name**

Specifies the schema containing the SQL table.

The equals sign (=) can be used in place of IS.

NEW copy

Specifies that the table is assigned the new copy attribute.

Records with the new copy attribute are allocated new table buffers when the dialog executes at runtime.

NC can be used in place of NEW COPY.

WORK

Specifies that the table is assigned the work attribute.

Records with the work table attribute are available to the dialog as working storage at runtime.

WK can be used in place of WORK.

If no attribute is specified for the named table, WORK is assigned as the default. If NEW COPY is specified for the table, WORK is not automatically assigned; the application developer must explicitly specify the work table attribute.

Expansion of record-options**VERsion is *version-number***

Specifies the version number (in the range 1 through 9999) of the named record.

If a version number is not specified, VERSION defaults to the system default version number, as specified in the OOAK record at system generation.

If no system default version number is specified in the OOAK record, VERSION defaults to 1.

The equals sign (=) can be used in place of IS.

NEW copy

Specifies that the record is assigned the new copy attribute.

Records with the new copy attribute are allocated new record buffers when the dialog executes at runtime.

NC can be used in place of NEW COPY.

WORK

Specifies that the record is assigned the work attribute.

Records with the work record attribute are available to the dialog as working storage at runtime.

WK can be used in place of WORK.

If no attribute is specified for the named record, WORK is assigned as the default. If NEW COPY is specified for the record, WORK is not automatically assigned; the application developer must explicitly specify the work record attribute.

Expansion of version#-options***version-number***

Specifies a single version number for the selected dialogs.

low-version-number high-version-number

Specifies all versions of the selected dialogs within the version-number range (inclusive).

The default version number is 1.

Usage*Considerations*

- The DIALOG clause must be the first clause of a dialog expression.
- The MAINLINE, SUBSCHEMA, MAPNAME, AUTOSTATUS, STATUS DEFINITION RECORD, ACTIVITY LOG, SYMBOL TABLE, DIAGNOSTIC TABLE, MESSAGE PREFIX, COBOL MOVE, ENTRY POINT, and RETRIEVAL LOCKING clauses can appear in any order, but must precede the first RECORD clause.
- The PAGING MODE, BACKPAGE, and PAGING TYPE clauses can be specified only if the dialog's map is pageable. These clauses can appear in any order, but must follow the MAPNAME clause and precede the first RECORD clause.
- All RECORD clauses must precede the first PROCESS clause.
- PREMAP PROCESS and RESPONSE PROCESS clauses can appear in any order, provided that the above requirements are met.

Example 1: Recompiling all dialogs

All dialogs in the load area are recompiled:

```
COMPILE FROM LOAD ALL.
```

Example 2: Recompiling dialogs by version number

All dialogs with version number 2 and version numbers 5 through 8 are recompiled:

```
COMPILE FROM LOAD ALL VERSION ( 2 ( 5 8 ) ).
```

Example 3: Recompiling dialogs by name

All dialogs with names that begin with C and that have the letters D and R in the fourth and fifth positions are recompiled:

```
COMPILE FROM LOAD DIALOG (C**DR**).
```

Example 4: Recompiling dialogs within a specified range

Dialogs QWERT001 through ZZZZZZZZ are recompiled:

```
COMPILE FROM LOAD DIALOG ( ( QWERT001 ZZZZZZZZ ) ).
```


Example 5: Recompiling an added dialog

The dialog SXADIAL is added and compiled:

```
COMPILE FROM SOURCE
  ADD DIALOG SXADIAL VER IS 1 MAINLINE YES
  ADD SUBSCHEMA DEMOSS01 SCHEMA DEMOSCHM VER 1
  ADD MAPNAME SXA1 VER IS 1
  ADD REC CUSTOMER VER 2 NC
  ADD REC SXAREC1 VER 1 NC WK
  ADD PREMAP SXAPREMAP VER 1
  ADD RESPONSE PROCESS NAME SXARESP5 VER 2 CONTROL KEY
    PF5 EXEC NO
  ADD RESPONSE PROCESS NAME SXARESP3 VER 1 CONTROL KEY ENTER
    RESPONSE FIELD SXARESP4 EXEC NO.
```

More information:

[CA ADS Runtime System](#) (see page 119)
[Control Commands](#) (see page 325)
[CA ADS Dialog Compiler \(ADSC\)](#) (see page 91)
[Error Handling](#) (see page 277)
[Debugging an CA ADS Dialog](#) (see page 723)
[Activity Logging for an CA ADS Dialog](#) (see page 675)
[Database Specifications Screen](#) (see page 109)
[Introduction to Process Language](#) (see page 155)
[Map Commands](#) (see page 449)
[Database Access Commands](#) (see page 363)

JCL and Commands

JCL and commands for running ADSOBCOM are shown below for z/OS, z/VSE, and z/VM systems.

z/OS JCL

Sample z/OS JCL for Central Version

ADSOBCOM (z/OS)

```
//          EXEC PGM=ADSOBCOM,REGION=500K
//STEPLIB DD DSN=idms.loadlib,DISP=SHR
//sysctl  DD DSN=idms.sysctl,DISP=SHR
//dclscr  DD DSN=cdms.dclscr,DISP=SHR
//SYSLST DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIDMS DD *
SYSIDMS parameters
//SYSIPT DD *
control statements
/*
```

<i>idms.loadlib</i>	data set name of the CA IDMS load library
<i>idms.sysctl</i>	data set name of the SYSCTL file
<i>dclscr</i>	ddname of the local scratch file (if one is specified in the DMCL; otherwise not required)
<i>cdms.dclscr</i>	data set name of the local scratch file (if one is specified; otherwise not required)
<i>sysctl</i>	ddname of the SYSCTL file
<i>SYSIDMS parameters</i>	a list of the SYSIDMS parameters that pertain to this job

Sample z/OS JCL for Local Mode

To execute ADSOBCOM in **local mode**, perform the following steps:

1. Remove the sysctl DD statement.
2. Add the following statements after the CDMSLIB DD statement:

```
//sysjrn1 DD DSN=idms.tapejrn1,DISP=(NEW,KEEP),UNIT=tape
//dictdb  DD DSN=idms.appldict.ddldml,DISP=SHR
//dloddb  DD DSN=idms.appldict.ddldclod,DISP=SHR
//dmsgdb  DD DSN=idms.sysmsg.ddldcmsg,DISP=SHR
```

<i>idms.appldict.ddldml</i>	data set name of the data dictionary DDLML area
<i>idms.appldict.ddldclod</i>	data set name of the data dictionary load area
<i>idms.sysmsg.ddldcmsg</i>	data set name of the data dictionary message area
<i>idms.tapejrn1</i>	data set name of the tape journal file

<i>dictdb</i>	ddname of the data dictionary DDLML area
<i>dloddb</i>	ddname of the data dictionary load area (DDLDCLOD)
<i>dmsgdb</i>	ddname of the data dictionary message area (DDLDM/ESAG)
<i>sysjnl</i>	ddname of the tape journal file
<i>tape</i>	symbolic device name of the tape journal file

z/VSE JCL

// LIBDEF *Sample z/VSE JCL for Central Version

ADSOBCOM (z/VSE)

```
// UPSI b           if specified in ADSOOPTI module
// DLBL    userlib
// EXTENT    ,nnnnnn
// LIBDEF    *.SEARCH=idmslib.sublib
// DLBL    dclscr,, 'cdms.dclscr',,DA
// EXTENT    sys014,nnnnnn
// ASSGN    sys014,DISK,VOL=nnnnnn,SHR
// EXEC     ADSOBCOM
control statements
SYSIDMS parameters
```

<i>b</i>	appropriate 1- to 8-character UPSI bit switch, as specified in the IDMSOPTI module
<i>cdms.dclscr</i>	file-id of the local scratch area
<i>dclscr</i>	filename of the local scratch area (if one is specified in the DMCL, otherwise not required)
<i>sys014</i>	logical unit assignment for the local scratch area (if one is specified in the DMCL, otherwise not required)
<i>nnnnnn</i>	volume serial number of the library
<i>userlib</i>	filename of the CA IDMS library
<i>idmslib.sublib</i>	file-id of the CA IDMS sublibrary
<i>SYSIDMS parameters</i>	A list of SYSIDMS parameters for this job

Note: For more information about SYSIDMS parameters, see the *CA IDMS Database Administration*.

Sample z/VSE JCL for Local Mode

To execute ADSOBCOM in **local mode**, perform the following steps:

1. Remove the UPSI specification.
2. Add the following statements before the EXEC statement:

```
// DLBL    dictdb, 'idms.appldict.ddldml',,DA
// EXTENT  sys015,nnnnnn
// ASSGN   sys015,DISK,VOL=nnnnnn,SHR
// DLBL    dloddb, 'idms.appldict.ddldclod',,DA
// EXTENT  sys017,nnnnnn
// ASSGN   sys017,DISK,VOL=nnnnnn,SHR
// DLBL    dmsgdb, 'idms.sysmsg.ddldcmsg',,DA
// EXTENT  sys016,nnnnnn
// ASSGN   sys016,DISK,VOL=nnnnnn,SHR
// TLBL    sys009, 'idms.tapejrn1',,nnnnnn,,f
// ASSGN   sys009,TAPE,VOL=nnnnnn
```

<i>idms.appldict.ddldml</i>	file-id of the data dictionary DDLML area
<i>idms.appldict.ddldclod</i>	file-id of the data dictionary load area
<i>idms.sysmsg.ddldcmsg</i>	file-id of the data dictionary message area
<i>idms.tapejrn1</i>	file-id of the tape journal file
<i>dictdb</i>	filename of the data dictionary DDLML area
<i>dloddb</i>	filename of the data dictionary load area (DDLDCLOD)
<i>dmsgdb</i>	filename of the data dictionary message area (DDLDM/ESAG)
<i>f</i>	file number of the tape journal file
<i>nnnnnn</i>	volume serial number
<i>sys009</i>	logical unit assignment for the tape journal file
<i>sys015</i>	logical unit assignment for the data dictionary DDLML area
<i>sys016</i>	logical unit assignment for the data dictionary message area
<i>sys017</i>	logical unit assignment for the data dictionary load area

z/VM commands

Sample z/VM commands for Central Version

ADSOBCOM (z/VM)

```
FILEDEF SYSLST PRINTER
FILEDEF SYSIDMS DISK sysidms input a
FILEDEF SYSIPT DISK bgen input a
GLOBAL LOADLIB idmslib
OSRUN ADSOBCOM
```

<i>sysidms input a</i>	filename, filetype, and filemode of the file containing the SYSIDMS input parameters
<i>bgen input a</i>	file identifier of the file containing ADSOBCOM source statements
<i>idmslib</i>	filename of the CA IDMS LOADLIB library

Note: For more information about SYSIDMS parameters, see the *CA IDMS Database Administration*.

Sample z/VM commands for Local Mode

To execute ADSOBCOM in **local mode**, add the following commands before the OSRUN command:

```
FILEDEF sysjml TAP1 SL VOLID nnnnnn (RECFM VB LRECL 111 BLKSIZE bbb
FILEDEF dictdb DISK dictdb dictfile d (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
FILEDEF dloddb DISK dloddb dictfile f (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
FILEDEF dmsgdb DISK dmsgdb dictfile e (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
```

<i>bbb</i>	block size of the tape journal file
<i>dictdb</i>	ddname of the data dictionary DDLML area
<i>dictdb dictfile d</i>	file identifier of the data dictionary DDLML area
<i>dloddb</i>	ddname of the data dictionary load area (DDLCLD)
<i>dloddb dictfile f</i>	file identifier of the data dictionary load area
<i>dmsgdb</i>	ddname of the data dictionary message area (DDLVM/ESAG)
<i>dmsgdb dictfile e</i>	file identifier of the data dictionary message area
<i>lll</i>	record length of the tape journal file
<i>nnnnnn</i>	volume serial number of the tape journal file
<i>ppp</i>	page size of the area

<i>sysjml</i>	ddname of the tape journal file
---------------	---------------------------------

Specifying Central Version or Local Mode

To specify whether ADSOBCOM executes under central version or in local mode, take one of the following actions:

1. Specify either `CVMACH=dc/ucf-machine-name` (for central version) or `*LOCAL*` (for local mode) as the first statement submitted to ADSOBCOM. *Dc/ucf-machine-name* is the 1- through 8-character user identifier of the z/VM virtual machine in which the DC/UCF system is executing.
2. Link edit ADSOBCOM with an IDMSOPTI module that specifies either `CVMACH=dc/ucf-machine-name` (for central version) or `CENTRAL=NO` (for local mode). Instructions for creating an IDMSOPTI module are given in *CA IDMS System Operations Guide*.
3. Code `PARM='CVMACH=dc/ucf-machine-name'` or `PARM='*LOCAL*'` on the OSRUN command used to invoke the compiler. This option is not allowed if the OSRUN command is issued from a z/VM EXEC program; however, it is allowed if the OSRUN command is issued from a System Product interpreter (REXX) or EXEC 2 program.

Note: For more information about central version and local mode operations in the z/VM environment, see the *Installation and Maintenance Guide— z/VM*.

ADSOBSYS

The ADSOBSYS utility builds a load module (ADSOOPTI) that supplies CA ADS system generation parameters to ADSOBCOM. ADSOBSYS must be run once for each DC/UCF system at installation and whenever CA ADS system generation parameters are changed.

The ADSOOPTI module can be either loaded at runtime by ADSOBCOM or link edited with ADSOBCOM. Note that with dynamic loading, the module must have the default ADSOOPTI module name.

ADSOBSYS can also supply system generation parameters to the CA ADS Batch runtime system.

ADSOBSYS uses standard control statements in addition to the SYSTEM statement. The control statements and the JCL used to run ADSOBSYS are presented below. Parameters given for the SYSTEM statement apply to CA ADS applications.

Control Statements

The following control statements can be used with ADSOBSYS:

ICTL

Specifies scanning a specified column range for meaningful data. The default specification is 1-72. The ICTL statement format is shown below:

Syntax

```
▶▶ ICTL = (start-column-number end-column-number) ◀◀
```

OCTL

Specifies the number of lines to appear on each page of the ADSOBSYS printed output. The default specification is 56. The OCTL statement format is shown below:

```
▶▶ OCTL = (line-count-number) ◀◀
```

ISEQ

Specifies sequence checking on source statements falling within a specified column range. The ISEQ statement format is shown below:

```
▶▶ ISEQ = (start-column-number end-column-number) ◀◀
```

The ICTL, OCTL, and ISEQ control statements must be submitted to the job stream before the SYSTEM statement.

SYSTEM Statement

Purpose

Specifies the DC/UCF system for which the ADSOOPTI module is being created.

Syntax

```
▶▶ SYStem  $\left[ \begin{array}{c} \text{IS} \\ \text{=} \end{array} \right]$  system-number MODULE  $\left[ \begin{array}{c} \text{IS} \\ \text{=} \end{array} \right]$  module-name ◀◀
```

Parameters

SYStem IS system-number

Specifies the 1- to 4-digit number of the DC/UCF system for which the ADSOOPTI module is being created.

MODULE IS *module-name*

Specifies the 1 to 8-character name of the module being created. The default module name is ADSOOPTI.

JCL and Commands

JCL and commands for running ADSOBSYS are shown below for z/OS, z/VSE, and z/VM systems.

z/OS JCL

Sample z/OS JCL for Central Version

ADSOBSYS (central version) (z/OS)

```
//ADSOBSYS EXEC PGM=ADSOBSYS,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
// DD DSN=idms.loadlib,DISP=SHR
//sysctl DD DSN=idms.sysctl,DISP=SHR
//dcmgs DD DSN=idms.sysmsg.ddldcmgs,DISP=SHR
//SYSLST DD SYSOUT=A
//SYSPCH DD DSN=&&object.,DISP=(NEW,PASS),
// UNIT=disk,SPACE=(TRK,1),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSIDMS DD *
DMCL=dmcl-name
DBNAME=system
Put other SYSIDMS parameters, as appropriate, here
/*
//SYSIPT DD *
Put ADSOBSYS parameters, as appropriate, here
/*
//LINKOPTI EXEC PGM=IEWL,REGION=1024K,PARM='LET,LIST,NCAL,XREF'
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=disk,SPACE=(CYL,(3,2))
//SYSLMOD DD DSN=idms.dba.loadlib,DISP=SHR
//SYSLIN DD DSN=&&object.,DISP=(OLD,DELETE)
// DD *
ENTRY adsoopti
NAME adsoopti(R)
/*
/*
```


Sample z/OS JCL for Local Mode ADSOBSYS (local mode) (z/OS)

```
//ADSOBSYS EXEC PGM=ADSOBSYS,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadLib,DISP=SHR
// DD DSN=idms.loadLib,DISP=SHR
//dcdml DD DSN=idms.system.ddldml,DISP=SHR
//dcmmsg DD DSN=idms.sysmsg.ddldcmmsg,DISP=SHR
//sysjrn1 DD DUMMY
//SYSLST DD SYSOUT=A
//SYSPCH DD DSN=&&object.,DISP=(NEW,PASS),
// UNIT=disk,SPACE=(TRK,1),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSIDMS DD *
DMCL=dmcl-name
DBNAME=system
Put other SYSIDMS parameters, as appropriate, here
/*
//SYSIPT DD *
Put ADSOBSYS parameters, as appropriate, here
/*
/*
//LINKOPTI EXEC PGM=HEWL,REGION=1024K,PARM='LET,LIST,NCAL,XREF'
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=disk,SPACE=(CYL,(3,2))
//SYSLMOD DD DSN=idms.dba.loadLib,DISP=SHR
//SYSLIN DD DSN=&&object.,DISP=(OLD,DELETE)
// DD *
ENTRY adsoopti
NAME adsoopti(R)
/*
/*
```

<i>idms.dba.loadlib</i>	Data set name of the load library containing the DMCL and database name table load modules
<i>idms.loadlib</i>	Data set name of the load library containing the CA IDMS executable modules
<i>sysctl</i>	DDname of the SYSCTL file
<i>idms.sysctl</i>	Data set name of the SYSCTL file
<i>dcmmsg</i>	DDname of the system message (DDLDM/ESAG) area
<i>idms.sysmsg.ddldcmmsg</i>	Data set name of the system message (DDLDM/ESAG) area
<i>&&object.</i>	Temporary data set name for the ADSOOPTI object module
<i>disk</i>	Symbolic device name for the work files
<i>dmcl-name</i>	Name of the DMCL load module

<i>system</i>	Name of the system dictionary
<i>adsoopti</i>	ADSOOPTI module name
<i>dcdml</i>	DDname of the system dictionary definition (DDLML) area
<i>idms.system.ddldml</i>	Data set name of the system dictionary definition (DDLML) area
<i>sysjml</i>	DDname of the tape journal file

Note: For more information about SYSIDMS parameters, see the *CA IDMS Database Administration*.

z/VSE JCL

Sample z/VSE JCL for Central Version

ADSOBSYS (z/VSE)

```
// UPSI b                if specified in IDMSOPTI module
// DLBL      userlib
// EXTENT    ,nnnnnn
// LIBDEF    *,SEARCH=idmslib.sublib
// LIBDEF    *,CATALOG=(userlib.cdmslib)
// DLBL      IDMSPCH,'temp.adsoopti'
// EXTENT    sysnnn,nnnnnn,,ssss,llll
// ASSGN     sysnnn,DISK,VOL=nnnnnn,SHR
// EXEC      ADSOBSYS
//          SYSTEM=nnnn,MODULE=adsoopti
SYSIDMS parameters
/*
// DLBL      IJSYSIN,'temp.adsoopti'
// EXTENT    SYSIPT,nnnnnn
// ASSGN     SYSIPT,DISK,VOL=nnnnnn,SHR
// OPTION    CATAL
//          PHASE adsoopti,*
//          INCLUDE
//          ENTRY (adsoopti)
// EXEC      LNKEDT
//          CLOSE SYSIPT,SYSRDR
//          CLOSE sysc1b,UA
```

<i>adsoopti</i>	ADSOOPTI module name
<i>SYSIDMS parameters</i>	a list of SYSIDMS parameters for this job
<i>b</i>	appropriate 1- to 8-character UPSI bit switch, as specified in the IDMSOPTI module

<i>llll</i>	number of tracks (CKD) or blocks (FBA) of the disk extent
<i>nnnn</i>	version number of the DC/UCF system
<i>nnnnnn</i>	volume serial number of the library
<i>ssss</i>	starting track (CKD) or block (FBA) of the disk extent
<i>sysnnn</i>	logical unit assignment of the temporary adsoopti module
<i>temp.adsoopti</i>	temporary file-id of the ADSOOPTI module
<i>userlib</i>	filename of the user library
<i>userlib.cdmslib</i>	file-id of the CA IDMS sublibrary

Note: For more information about SYSIDMS parameters, see the *CA IDMS Database Administration*.

Sample z/VSE JCL for Local Mode

To execute ADSOBSYS in **local mode**, perform the following steps:

1. Remove the UPSI specification.
2. Add the following statements before the EXEC ADSOBSYS statement:

```
// DLBL    dictcb,'idms.appldict.ddldml',,DA
// EXTENT  sys015,nnnnn
// ASSGN   sys015,DISK,VOL=nnnnnn,SHR
// DLBL    dloddb,'idms.appldict.ddldclod',,DA
// EXTENT  sys017,nnnnn
// ASSGN   sys017,DISK,VOL=nnnnnn,SHR
// DLBL    dmsgdb,'idms.sysmsg.ddldcmsg',,DA
// EXTENT  sys016,nnnnn
// ASSGN   sys016,DISK,VOL=nnnnnn,SHR
// TLBL    sys009,'idms.tapejrn1',, nnnnnn,,f
// ASSGN   sys009,TAPE,VOL=nnnnnn
```

<i>idms.appldict.ddldml</i>	file-id of the data dictionary DDLDML area
<i>idms.appldict.ddldclod</i>	file-id of the data dictionary load area
<i>idms.sysmsg.ddldcmsg</i>	file-id of the data dictionary message area
<i>idms.tapejrn1</i>	file-id of the tape journal file
<i>dictdb</i>	filename of the data dictionary DDLDML area
<i>dloddb</i>	filename of the data dictionary load area (DDLDCLOD)
<i>dmsgdb</i>	filename of the data dictionary message area (DDLDM/ESAG)
<i>f</i>	file number of the tape journal file

<i>nnnnnn</i>	volume serial number of the library
<i>sys009</i>	logical unit assignment for the tape journal file
<i>sys015</i>	logical unit assignment for the data dictionary DDLML area
<i>sys016</i>	logical unit assignment for the data dictionary message area
<i>sys017</i>	logical unit assignment for the data dictionary load area
<i>userlib</i>	filename of the user library
<i>userlib.cdmslib</i>	file-id of the CA IDMS sublibrary

z/VM commands

Sample z/VM Commands for Central Version

ADSOBSYS (z/VM)

```
FILEDEF SYSLST PRINTER
FILEDEF SYSPCH DISK opti TEXT a (LRECL 80 BLKSIZE 400 RECFM FB
FILEDEF SYSIDMS DISK sysidms input a
FILEDEF SYSIPT DISK bsys input a
GLOBAL LOADLIB idmslib
OSRUN ADSOBSYS
FILEDEF SYSPRINT PRINTER
TXTLIB DEL utextlib opti
TXTLIB ADD utextlib opti
FILEDEF SYSLMOD DISK uloadlib LOADLIB a6 (RECFM V LRECL 1024 BLKSIZE 1024
FILEDEF objlib DISK utextlib TXTLIB a
LKED linkctl (LET LIST NCAL
```

Linkage editor control statements (linkctl):

```
INCLUDE objlib (opti)
ENTRY opti
NAME opti(R)
```

<i>sysidms input a</i>	filename, filetype, and filemode of the file containing the SYSIDMS input parameters
<i>bsys input a</i>	file identifier of the file containing ADSOBSYS source statements
<i>idmslib</i>	filename of the CA IDMS LOADLIB library
<i>linkctl</i>	filename of the file containing the linkage editor control statements; the file must have the filetype of TEXT
<i>objlib</i>	ddname of the user TXTLIB library

<i>opti</i>	filename of the file for the ADSOOPTI module
<i>opti TEXT a</i>	file identifier of the file for the ADSOOPTI module
<i>uoadlib LOADLIB a6</i>	file identifier of the user LOADLIB library
<i>utextlib</i>	filename of a user TXTLIB library

Note: For more information about SYSIDMS parameters, see the *CA IDMS Database Administration*.

Sample z/VM Commands for Local Mode

To execute ADSOBSYS in **local mode**, add the following commands before the OSRUN command:

```
FILEDEF sysjml TAP1 SL VOLID nnnnn (RECFM VB LRECL 111 BLKSIZE bbb
FILEDEF dictdb DISK dictdb dictfile d (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
FILEDEF dloddb DISK dloddb dictfile f (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
FILEDEF dmsgdb DISK dmsgdb dictfile e (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
```

<i>bbb</i>	block size of the tape journal file
<i>dictdb</i>	ddname of the data dictionary DDLML area
<i>dictdb dictfile d</i>	file identifier of the data dictionary DDLML area
<i>dloddb</i>	ddname of data dictionary load area (DDLDCLOUD)
<i>dloddb dictfile f</i>	file identifier of the data dictionary load area
<i>dmsgdb</i>	ddname of data dictionary message area (DDLDM/ESAG)
<i>dmsgdb dictfile e</i>	file identifier of the data dictionary message area
<i>lll</i>	record length of the tape journal file
<i>nnnnn</i>	volume serial number of the tape journal file
<i>ppp</i>	page size of the area
<i>sysjml</i>	ddname of the tape journal file

Specifying Central Version or Local Mode

To specify whether ADSOBSYS executes under central version or in local mode, perform one of the following actions:

1. Specify either `CVMACH=dc/ucf-machine-name` (for central version) or `*LOCAL*` (for local mode) as the first statement to submit to ADSOBSYS. `Dc/ucf-machine-name` is the 1- through 8-character user identifier of the z/VM virtual machine in which the DC/UCF system is executing.
2. Link edit ADSOBSYS with an IDMSOPTI module that specifies either `CVMACH=dc/ucf-machine-name` (for central version) or `CENTRAL=NO` (for local mode). Instructions for creating an IDMSOPTI module are given in *CA IDMS System Operations Guide*.
3. Code `PARM='CVMACH=dc/ucf-machine-name'` or `PARM='*LOCAL*'` on the OSRUN command used to invoke the compiler. This option is not allowed if the OSRUN command is issued from a z/VM EXEC program; however, it is allowed if the OSRUN command is issued from a System Product interpreter (REXX) or EXEC 2 program.

Note: For more information about central version and local mode operations in the z/VM environment, see the *CA IDMS Installation and Maintenance Guide— z/VM*.

ADSOBTAT

What It Is

ADSOBTAT is a batch utility that allows the application developer to add, modify, and delete entries in the task application table (TAT). For example, ADSOBTAT can be used to update the TAT for a dictionary when an application is migrated to that dictionary.

Note: When an application is added, modified, or deleted by using the application compiler, the TAT is automatically updated in the applicable dictionary, and ADSOBTAT is not required. The TAT can also be updated online by using ADSOTATU, as described under 'ADSOTATU' earlier in this appendix.

How It Works

At the beginning of an ADSOBTAT run, ADSOBTAT copies the TAT stored either in the load area or, if the load area has no TAT, in the load library. If no TAT exists, ADSOBTAT creates a new TAT if the action is ADD. ADSOBTAT updates the copy of the TAT, based on the control statements provided. At the end of the run, if the control statements contain no errors, ADSOBTAT stores the copy of the TAT in the load area, replacing any previous copy.

ADSOBTAT does not update a TAT's program description element (PDE) to indicate that a new copy of the TAT exists in the load area. If a TAT is updated by ADSOBTAT and then referenced during a single DC/UCF run, the application developer should update the PDE by issuing the following command:

```
DCMT VARY PROGRAM $ACF@TAT NEW COPY
```

Note: For more information on the DCMT VARY PROGRAM command, see the *CA IDMS System Tasks and Operator Commands Guide*.

ADSOBTAT Output

ADSOBTAT produces a listing that displays the card images of all control statements processed. Error messages, if any, are listed under their associated control statements. If an error in any control statement prevents ADSOBTAT from updating the TAT, ADSOBTAT issues the following message:

```
DC474029 *** WARNING *** DUE TO ABOVE ERROR TAT WILL NOT BE  
UPDATED DURING THIS RUN; SYNTAX CHECKING ONLY
```

Note: At a site where alternate dictionaries are used, the system database name table must map network subschema IDMSNWKL (used by ADSOBTAT) to the copy of the network subschema appropriate for the alternate dictionary. The database name table is defined by the DBNAME statement.

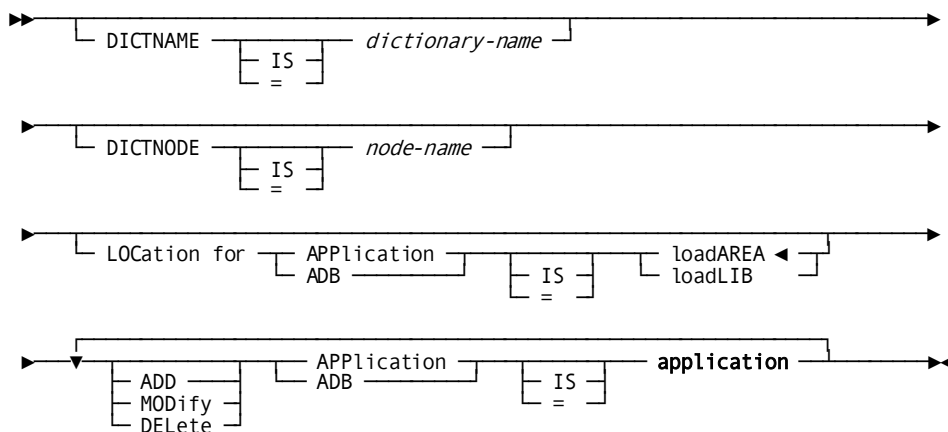
More information:

[CA ADS Application Compiler \(ADSA\)](#) (see page 51)

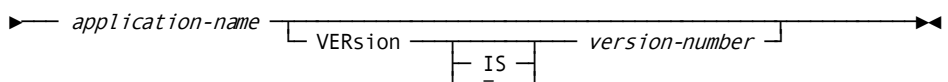
Control Statements

Purpose

Syntax



Expansion of Application



Parameters

DICTNAME IS *dictionary-name*

Specifies the 1- to 8-character name of the data dictionary in which the TAT is stored.

DICTNAME defaults to the name of the primary dictionary.

DICTNODE IS *node-name*

Specifies the node that controls the data dictionary in which the TAT is stored.

LOCation for APPLICATION IS

Introduces where the applications specified in the control statements are stored.

ADB can be used in place of APPLICATION; the equals sign (=) can be used in place of IS.

loadAREA

Specifies that the applications are stored in the load area.

LOADAREA is the default when no location for application is specified.

loadLIB

Specifies that the applications are stored in the load (core-image) library.

The load libraries in which the applications are stored must be specified in the JCL, as follows:

- **z/OS JCL** -- In the CDMSLIB statement or, if a CDMSLIB statement is not specified, in the STEPLIB statement
- **z/VSE JCL** -- In the ASSGN/EXTNT statement for the private core-image library or in the LIBDEF equivalent
- **z/VM commands** -- In the GLOBAL LOADLIB command, added to the list of libraries

ADD

Specifies that task code entries for an application are being added to the TAT.

If ADD is specified and the TAT already contains entries for the application, the action is changed to MOD and a warning message is displayed. If ADD is specified and the TAT does not exist, ADSOBTAT creates a TAT.

ADD is the default if the TAT contains no entries for the application.

MODify

Specifies that the task code entries for an application are being replaced in the TAT by the task codes defined in the current application load module.

If MOD is specified and the TAT does not contain entries for the application, ADSOBTAT treats the request like an ADD request.

MOD is the default if the TAT already contains entries for the application.

DElete

Specifies that the task code entries for an application are being deleted from the TAT.

If the TAT does not contain entries for the application, a warning message is issued. Note that the application does not have to exist when DEL is specified.

APPLICATION IS *application*

Identifies the application.

See expansion of *application* below.

application-name

Specifies the name of the application.

VERsion is *version-number*

Gives the version number (in the range 1 through 9999) of the application.

The default version number is 1.

Usage

Considerations

If specified, the DICTNODE and DICTNAME clauses must be coded first, in any order. The LOCATION clauses, if specified, must be coded next, in any order. The ADD/MODIFY/DELETE APPLICATION clause must be coded last, and can be repeated any number of times to reference several applications.

JCL and Commands

JCL for running ADSOBTAT is shown below for z/OS, z/VSE, and z/VM systems.

z/OS JCL

Sample z/OS JCL for Central Version

ADSOBTAT (central version) (z/OS)

```
//ADSOBTAT EXEC PGM=ADSOBTAT,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
// DD DSN=idms.loadlib,DISP=SHR
//sysctl DD DSN=idms.sysctl,DISP=SHR
//dcmmsg DD DSN=idms.sysmsg.ddldcmmsg,DISP=SHR
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
DMCL=dmcl-name
Put other SYSIDMS parameters, as appropriate, here
/*
//SYSIPT DD *
Put ADSOBTAT parameters, as appropriate, here
/*
/*
```

ADSOBTAT (local mode) (z/OS)

```
//ADSOBTAT EXEC PGM=ADSOBTAT,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadLib,DISP=SHR
//          DD DSN=idms.loadLib,DISP=SHR
//dloddb DD DSN=idms.appldict.ddldclod,DISP=SHR
//dcmmsg DD DSN=idms.sysmsg.ddldcmmsg,DISP=SHR
//sysjrnl DD DSN=idms.tapejrnl,DISP=(NEW,CATLG),UNIT=tape
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
DMCL=dmcl-name
Put other SYSIDMS parameters, as appropriate, here
/*
//SYSIPT DD *
Put ADSOBTAT parameters, as appropriate, here
/*
/**
```

<i>idms.dba.loadlib</i>	Data set name of the load library containing the DMCL and database name table load modules
<i>idms.loadlib</i>	Data set name of the load library containing the CA IDMS executable modules
<i>sysctl</i>	DDname of the SYSCTL file
<i>idms.sysctl</i>	Data set name of the SYSCTL file
<i>dcmmsg</i>	DDname of the system message (DDLDM/ESAG) area
<i>idms.sysmsg.ddldcmmsg</i>	Data set name of the system message (DDLDM/ESAG) area
<i>dmcl-name</i>	Name of the DMCL load module
<i>dloddb</i>	DDname of the application dictionary definition load (DDLDCLOD) area
<i>idms.appldict.ddldclod</i>	Data set name of the application dictionary definition load (DDLDCLOD) area
<i>sysjrnl</i>	DDname of the tape journal file
<i>idms.tapejrnl</i>	Data set name of the tape journal file
<i>tape</i>	symbolic device name of the tape journal file

Note: For more information about SYSIDMS parameters, see the *CA IDMS Database Administration*.

z/VSE JCL

Sample z/VSE JCL for Central Version

ADSOBTAT (z/VSE)

```
// UPSI      b                if specified in the IDMSOPTI module
// DLBL      userlib
// EXTENT    ,nnnnnn
// LIBDEF    *,SEARCH=(userlib.cdmslib)
// EXEC      ADSOBTAT
control statements
SYSIDMS parameters
```

<i>b</i>	appropriate 1- through 8-character UPSI bit switch, as specified in the IDMSOPTI module
<i>nnnnnn</i>	volume serial number of the library
<i>userlib</i>	filename of the user library
<i>userlib.cdmslib</i>	file-id of the CA IDMS sublibrary
<i>SYSIDMS parameters</i>	A list of SYSIDMS parameters for this job

Note: For more information about SYSIDMS parameters, see the *CA IDMS Database Administration*.

Sample z/VSE JCL for Local Mode

To execute ADSOBTAT in **local mode**, perform the following steps:

1. Remove the UPSI specification.
2. Add the following statements before the EXEC statement:

```
// DLBL      dloddb, 'idms.appldict.ddldclod', , DA
// EXTENT    sys017, nnnnnn
// ASSGN     sys017, DISK, VOL=nnnnnn, SHR
// DLBL      dmsgdb, 'idms.sysmsg.ddldcmsg', , DA
// EXTENT    sys016, nnnnnn
// ASSGN     sys016, DISK, VOL=nnnnnn, SHR
// TLBL      sys009, 'idms.tapejrn1', , nnnnnn, , f
// ASSGN     sys009, TAPE, VOL=nnnnnn
```

<i>idms.appldict.ddldclod</i>	file-id of the data dictionary load area
<i>idms.sysmsg.ddldcmsg</i>	file-id of the data dictionary message area
<i>idms.tapejrn1</i>	file-id of the tape journal file

<i>dloddb</i>	filename of the data dictionary load area (DDLDCLOUD)
<i>dmsgdb</i>	filename of the data dictionary message area (DDLDM/ESAG)
<i>f</i>	file number of the tape journal file
<i>nnnnn</i>	volume serial number
<i>sys009</i>	logical unit assignment for the tape journal file
<i>sys016</i>	logical unit assignment for the data dictionary message area
<i>sys017</i>	logical unit assignment for data dictionary load area

z/VM commands

Sample z/VM Commands for Central Version

ADSOBTAT (z/VM)

```
FILEDEF SYSLST PRINTER
FILEDEF SYSIDMS DISK sysidms input a
FILEDEF SYSIPT DISK btat input a
GLOBAL LOADLIB idmslib
OSRUN ADSOBTAT
```

<i>sysidms input a</i>	filename, filetype, and filemode of the file containing the SYSIDMS input parameters
<i>btat input a</i>	file identifier of the file containing ADSOBTAT source statements
<i>idmslib</i>	filename of the CA IDMS LOADLIB library

Note: For more information about SYSIDMS parameters, see the *CA IDMS Database Administration*.

Sample z/VM Commands for Local Mode

To execute ADSOBTAT in **local mode**, add the following commands before the OSRUN command:

```
FILEDEF sysjml TAP1 SL VOLID nnnnn (RECFM VB LRECL 111 BLKSIZE bbb
FILEDEF dictdb DISK dictdb dictfile d (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
FILEDEF dloddb DISK dloddb dictfile f (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
FILEDEF dmsgdb DISK dmsgdb dictfile e (RECFM F LRECL ppp BLKSIZE ppp XTENT nnn
```

<i>bbb</i>	block size of the tape journal file
------------	-------------------------------------

<i>dictdb</i>	ddname of the data dictionary DDLML area
<i>lll</i>	record length of the tape journal file
<i>nnnnnn</i>	volume serial number of the tape journal file
<i>sysjml</i>	ddname of the tape journal file
<i>dictdb dictfile d</i>	file identifier of the data dictionary DDLML area
<i>ppp</i>	page size of the area
<i>dloddb</i>	ddname of the data dictionary load area (DDLCLD)
<i>dloddb dictfile f</i>	file identifier of the data dictionary load area
<i>dmsgdb</i>	ddname of the data dictionary message area (DDLVM/ESAG)
<i>dmsgdb dictfile e</i>	file identifier of the data dictionary message area

Specifying Central Version or Local Mode

To specify whether ADSOBTAT executes under central version or in local mode, take one of the following actions:

1. Specify either *CVMACH=dc/ucf-machine-name* (for central version) or **LOCAL** (for local mode) as the first statement to submit to ADSOBTAT. *Dc/ucf-machine-name* is the 1- through 8-character user identifier of the z/VM virtual machine in which the DC/UCF system is executing.
2. Link edit ADSOBTAT with an IDMSOPTI module that specifies either *CVMACH=dc/ucf-machine-name* (for central version) or *CENTRAL=NO* (for local mode). Instructions for creating an IDMSOPTI module are given in *CA IDMS System Operations Guide*.
3. Code *PARM='CVMACH=dc/ucf-machine-name'* or *PARM='*LOCAL*'* on the OSRUN command used to invoke the compiler. This option is not allowed if the OSRUN command is issued from a z/VM EXEC program; however, it is allowed if the OSRUN command is issued from a System Product interpreter (REXX) or EXEC 2 program.

Note: For more information about central version and local mode operations in the z/VM environment, see the *CA IDMS Installation and Maintenance— z/VM*.

ADSOTATU

ADSOTATU is an online utility that allows the application developer to add, modify, and delete entries in the task application table (TAT). For example, ADSOTATU can be used to update the TAT for a dictionary when an application is migrated to that dictionary.

Note: When an application is added, modified, or deleted by using the application compiler, the TAT is automatically updated in the applicable dictionary, and ADSOTATU is not required. The TAT can also be updated in batch by using ADSOBTAT, as described under 'ADSOBTAT' later in this appendix.

ADSOTATU is invoked by specifying the task code ADSOTATU at the DC/UCF prompt.

ADSOTATU displays the TAT Update Utility screen on which the application developer specifies the name of the application whose TAT entries are being added, modified, or deleted. Additionally, the application developer can specify the dictionary that contains the TAT, the node that controls the dictionary, the application version number, and the action to take regarding the TAT entries.

Each time the application developer specifies an action, ADSOTATU makes a copy of the TAT stored in either the program pool, load area (if the program pool has no TAT), or load library (if the load area has no TAT). If no TAT exists, ADSOTATU creates a new TAT if the action is ADD. ADSOTATU updates the copy of the TAT as appropriate; stores the copy in the load area, replacing any previous TAT; and issues a DCMT VARY PROGRAM NEW COPY command to update the TAT in the program pool.

Specifying Activities

The application developer specifies activities in an ADSOTATU session by using the ENTER, CLEAR, and PF9 keys, as follows:

- **ENTER** instructs ADSOTATU to add, modify, or delete entries in the TAT, based on information specified on the screen.

If the TAT is updated successfully, ADSOTATU issues a confirming message. The screen can be used repeatedly to specify several applications.

If ADSOTATU encounters an error, it redisplay the screen with an appropriate error message. The application developer can change information on the screen, then resubmit the information by pressing ENTER.

- **CLEAR** and **PF9** terminate ADSOTATU and return control to DC/UCF.

More information:

[CA ADS Application Compiler \(ADSA\)](#) (see page 51)

TAT Update Utility Screen

Sample Screen

CA ADS REL nn.n	CA, INC.	***TAT UPDATE UTILITY***
	DICT NAME:	NODE:
ACTION: (ADD/MOD/DEL)		
APPLICATION:	VERSION:	

Field descriptions

DICT NAME

Specifies the 1- to 8-character name of the data dictionary in which the TAT is stored.

DICT NAME defaults to the name of the primary dictionary. Specifying a dictionary name is equivalent to issuing a DCUF SET DICTNAME command under DC/UCF.

NODE

(for DDS only) Specifies the DDS node that controls the data dictionary specified by DICT NAME.

NODE defaults to the system currently in use. Specifying a node name is equivalent to issuing a DCUF SET DICTNODE command under DC/UCF.

ADD

Specifies that task code entries for an application are being added to the TAT.

If ADD is specified and the TAT already contains entries for the application, the action is changed to MOD and a warning message is displayed.

ADD is the default if the TAT contains no entries for the application.

MODify

Specifies that the task code entries for an application are being replaced in the TAT by the task codes defined in the current application load module.

If MODIFY is specified and the TAT does not contain entries for the application, ADSOTATU treats the request like an ADD request.

MODIFY is the default if the TAT already contains entries for the application.

DELeTe

Specifies that the task code entries for an application are being deleted from the TAT.

If the TAT does not contain entries for the application, a warning message is issued. Note that the application does not have to exist when DEL is specified.

APPLICATION

Specifies the name of the application. If the action is ADD or MOD, the specified application must exist in the data dictionary specified by DICT NAME.

VERSION

Specifies the version number (in the range 1 through 9999) of the application. The default version number is 1.

Appendix E: Activity Logging for an CA ADS Dialog

This section contains the following topics:

[Overview](#) (see page 675)

[Data Dictionary Organization](#) (see page 676)

[Activity Logging Record Formats](#) (see page 676)

Overview

What it Does

The activity logging feature of CA ADS creates activity records that document all potential database activity for a dialog. Documentation is based on the database commands issued explicitly or implicitly by the dialog's processes. (Examples of implicit database commands are the implicit READY command, issued automatically for each subschema area when a run unit is opened for a process, and the BIND command, issued automatically for each subschema record used by the dialog.) If enabled, activity logging is performed when a dialog is compiled and has no impact on runtime performance.

The activity logging feature can be used to perform the following tasks:

- **Monitor database usage** — If runtime activity is high for a particular subschema area, set, record, or logical record, activity records can show which dialogs contain database commands that potentially access the entity.
- **Modify dictionary entity occurrences** — If a subschema area, set, record, or logical record needs to be modified, activity records can show which dialogs need to be recompiled as a result of the modification.

Enabling Activity Logging

The activity logging feature is enabled or disabled at system generation. The application developer can override the system generation default when defining a dialog.

If enabled, the activity logging feature creates database activity records when a dialog is compiled. Database activity records can be accessed by using query programs such as OnLine Query, by using the Data Dictionary Reporter, or by writing an appropriate program.

The remainder of this appendix describes the data dictionary organization and the format of activity logging records.

More information:

[CA ADS Dialog Compiler \(ADSC\)](#) (see page 91)

Data Dictionary Organization

Database activity records are stored as junction records between a dialog's PROG-051 record and the dictionary's AREA, SET, RECORD, and LOGICAL RECORD entities, as follows:

Dictionary entity	Junction record stored
SSA-024 (AREA)	AFACT-057
SSOR-034 (SET)	SETACT-061
SSR-032 (RECORD)	RCDACT-059
LR-190 (LOGICAL RECORD)	LRACT-193

Thus, if an area, set, record, or logical record must be modified, the application developer can follow a path from the dictionary entity occurrence, through the appropriate junction record, to the PROG-051 records of the dialogs that need to be recompiled because of the modification.

Activity Logging Record Formats

An activity logging record contains the following information about the database command being logged for a dialog:

- The function number of the database command being logged
- The number of times in the dialog that the database command is coded against the dictionary entity occurrence
- The name of the dictionary entity occurrence

AFACT-057

The record description of the AFACT-057 junction record is as follows:

```
02 AF-FUNCT-057    PICTURE IS S9(4) USAGE IS COMP.
02 AF-COUNT-057   PICTURE IS 9(4)  USAGE IS COMP.
02 AF-AREA-OWN-057 PICTURE IS X(32) USAGE IS DISPLAY.
02 EXTRNL-NAME-057 PICTURE IS X(32) USAGE IS DISPLAY.
02 FILLER         PICTURE IS X(4)  USAGE IS DISPLAY.
```

SETACT-061

The format of the SETACT-061 junction record is as follows:

```
02 SA-FUNCT-061    PICTURE IS 9(4)  USAGE IS COMP.
02 SA-COUNT-061    PICTURE IS 9(4)  USAGE IS COMP.
02 SA-SET-OWN-061  PICTURE IS X(32) USAGE IS DISPLAY.
02 FILLER          PICTURE IS X(4)  USAGE IS DISPLAY.
```

RCDACT-059

The format of the RCDACT-059 junction record is as follows:

```
02 RA-FUNCT-059    PICTURE IS 9(4)  USAGE IS COMP.
02 RA-COUNT--059   PICTURE IS 9(4)  USAGE IS COMP.
02 RA-RCD-OWN-059  PICTURE IS X(32) USAGE IS DISPLAY.
02 FILLER          PICTURE IS X(4)  USAGE IS DISPLAY.
```

LRACT-193

The format of the LRACT-193 junction record is as follows:

```
02 FUNCT-193      PICTURE IS S9(4) USAGE IS COMP.
02 COUNT-193      PICTURE IS S9(4) USAGE IS COMP.
02 LR-NAM-193     PICTURE IS X(16) USAGE IS DISPLAY.
```

Record Fields**FUNCT**

Contains the numeric function number that is assigned to the database command or logical record command being logged.

The function numbers for the AFACT-057 (AREA), SETACT-061 (SET), RCDACT-059 (RECORD), and LRACT-193 (LOGICAL RECORD) junction records and their associated database or logical record commands are listed in the following table.

Note: No activity records are stored for the COMMIT and ROLLBACK database commands.

COUNT

Contains the number of times the logged database command is coded in all of the processes of the dialog.

OWN

Contains the name of the record, set, or area whose activity is being documented by the record, set, or area activity record.

EXTRNL-NAME

Contains spaces.

NAM

Contains the name of the logical record whose activity is being documented by the logical record activity record.

Usage

Considerations

The COUNT field for a READY command reflects only the effective READY commands issued implicitly or explicitly in the dialog's processes. An implicit or explicit READY command sets the usage mode of a database area during a dialog's premap or response process.

If the same area is named in more than one READY command in a process, the usage mode specified in the last READY command applies to the named area for the entire process. The COUNT field of a junction record for a READY command reflects the number of processes for which the specified usage mode applies to the specified area.

Activity Logging Function Numbers and Associated Commands

Junction record	Function number	Navigational or LRF database command
AFACT-057	3	FIND
	6	KEEP and KEEP LONGTERM
	15	ACCEPT
	23	FIND KEEP
	36	READY USAGE MODE UPDATE
	37	READY USAGE MODE RETRIEVAL
	38	READY USAGE MODE PROTECTED UPDATE
	39	READY USAGE MODE PROTECTED RETRIEVAL
	40	READY USAGE MODE EXCLUSIVE RETRIEVAL
	41	READY USAGE MODE EXCLUSIVE UPDATE
SETACT-061	43	OBTAIN
	63	OBTAIN KEEP
	3	FIND
	6	KEEP and KEEP LONGTERM
	7	CONNECT
	11	DISCONNECT

Junction record	Function number	Navigational or LRF database command
	15	ACCEPT
	16	IF SET EMPTY/MEMBER
	17	RETURN
	23	FIND KEEP
	43	OBTAIN
	63	OBTAIN KEEP
RCDACT-059	2	ERASE
	3	FIND
	5	GET
	6	KEEP and KEEP LONGTERM
	7	CONNECT
	8	MODIFY
	11	DISCONNECT
	12	STORE
	14	BIND
	15	ACCEPT
	23	FIND KEEP
	43	OBTAIN
	63	OBTAIN KEEP
LRACT-193	2	ERASE
	8	MODIFY
	12	STORE
	43	OBTAIN

More information:

[Map Commands](#) (see page 449)

Appendix F: Built-in Function Support

This section contains the following topics:

[Overview](#) (see page 681)

[Internal Structure Of Built-In Functions](#) (see page 681)

[Assembler Macros](#) (see page 701)

[Changing Invocation Names](#) (see page 713)

[Creating User-Defined Built-In Functions](#) (see page 714)

Overview

About Built-In Functions

CA ADS built-in function support enables an installation to change the invocation names of built-in functions and to generate user-defined built-in functions. This appendix discusses the following topics:

- The internal structure of built-in functions
- The assembler macros that define components of built-in functions
- How to change invocation names
- How to create user-defined built-in functions

To change invocation names, the user needs only to read the following topics:

- The discussion of the master function table under 'Internal Structure of Built-in Functions'
- The discussion of the #EFUNMST macro under 'Assembler Macros'
- The instructions provided under 'Changing Invocation Names'

Internal Structure Of Built-In Functions

CA ADS supplied built-in functions and user-defined built-in functions share the same internal structure. This structure consists of the following components.

Master Function Table

The **master function table** lists the invocation names for each built-in function. The master function table is used during process compilation to associate a coded invocation name with a real (generic) function name and to point to a model XDE (expression description element) module that describes the function.

Model XDE Modules

Model XDE modules contain one or more model XDE tables. Each model XDE table describes a function, including the function's parameters, work area requirements, result field, and processing program name. During process compilation, a model XDE table is used to produce a series of XDEs that form the compiled representation of the function.

XDEs and VSDEs

XDEs and **VXDEs** describe functions at runtime. XDEs are created during process compilation; one VXDE (variable expression description element) is created for each XDE at runtime to hold variable information.

Processing Program Modules

Processing program modules contain processing logic for one or more functions. At runtime, when the XDEs and VXDEs for the function are processed, the runtime system calls the appropriate program and passes to it all required information. The program executes, then returns control to the dialog.

Master Function Table

The master function table is a dictionary load module that lists the invocation names for all CAADS supplied and user defined built-in functions. Each entry contains a function invocation name, a corresponding real (generic) function name, and the name of the model XDE module that describes the function.

The concatenate function, for example, has by default three invocation names: CONCATENATE, CONCAT, and CON. Each invocation name has an entry in the master function table. Each entry also specifies the real function name for the concatenate function, CONCAT, and the model XDE module that describes the concatenate function, RHDCEV51.

During compilation of a coded function, the dialog compiler searches the master function table for the coded invocation name. If it finds an entry, it uses the information in the entry to find the model XDE module that describes the function; if it does not find an entry, it generates a syntax error message.

Note: At runtime, an invocation name that is used in a dialog must not duplicate the name of a record element known to the dialog. If it does, CA ADS interprets the function as a subscripted reference to the record element.

The DSECT for an entry in the master function table is shown below. The load module for the master function table is stored in the data dictionary load area under the name RHDCEVBF.

DSECT for a Master Function Table Entry

EFMASDS	DSECT		11:15:30 03/06/86	00001000
*			EVAL MASTER FUNCTION TABLE ENTRY DSECT	00002000
EFMINAML	DS	H	LENGTH OF INVOCATION FUNCT NAME	00003000
EFMINAME	DS	CL32	FUNCTION NAME - INVOCATION	00004000
EFMRNAME	DS	CL8	FUNCTION NAME - REAL	00005000
EFMPPGMN	DS	CL8	PROGRAM NAME - MODEL XDE TABLE	00006000
EFMPPGMV	DS	H	PROGRAM VERSION - MODEL XDE TABLE	00007000
EFMFLAG1	DS	XL1	MASTER FUNCTION ENTRY FLAG1	00008000
EFMASFU	EQU	X'80'	AGGREGATE FUNCTION ENTRY	00009000
	DS	XL3	FILLER	00010000
EFMASLNG	EQU	*-EFMASDS	LENGTH OF MASTER ENTRY	00011000

Model XDE Module

A model XDE module is a load module that contains one or more model XDE tables, each describing a function. During process compilation of a function, the dialog compiler uses the appropriate model XDE table to generate a series of XDEs that form the compiled representation of the function. A model XDE table contains the following entries:

- A **header** entry that contains the function's processing program name, work area requirements, number and types of function parameters, and a description of the function's result field. Each model XDE table contains one header entry.
- **XDE** entries that describe the function's parameters and determine certain characteristics of the result field. One XDE entry exists for each function parameter.
- **Data type conversion** entries that define the data types and length of each function parameter. One or more data type conversion entries exist for each function parameter.

The DSECTs for these three entries are shown below. The load modules for the model XDE modules are stored in the load library. The model XDE source and load modules for the CA ADS supplied built-in functions are called RHDCEV51, RHDCEV52, RHDCEV53, and RHDCEV59, and can be used as a reference when defining user-defined built-in functions.

DSECTs for the Model XDE Table Entries

EFHDRDS	DSECT		12:01:34 05/18/84	00001000
*			EVAL FUNCTION MODEL TABLE HEADER DSECT	00002000
EFHNEXT	DS	H	OFFSET TO NEXT HDR ENTRY	00003000
EFHFUNNM	DS	CL8	FUNCTION NAME - REAL	00004000
EFHPPGMN	DS	CL8	PROCESSING PROGRAM NAME	00005000
EFHPPGMV	DS	H	PROCESSING PROGRAM VERSION	00006000
EFHFUNCN	DS	XL1	FUNCTION NUMBER	00007000
	DS	XL1	FILLER	00008000
EFHWORKL	DS	H	LENGTH OF REQUIRED WORKAREA	00009000
EFHZOPND	DS	0XL4	4 X'00'S INDICATE ZERO OPERANDS	00010000
EFHFOPDN	DS	H	NUMBER OF FIXED OPERANDS	00011000
EFHVOPDO	DS	H	OFFSET TO VARIABLE OPERAND MODEL	00012000
EFHRESLN	DS	H	RESULT LENGTH IN BYTES	00013000
EFHRDATP	DS	XL1	RESULT DATA TYPE	00014000
EFHRNDEC	DS	XL1	RESULT NUMBER DECIMALS	00015000
	DS	XL4	FILLER	00016000
EFHDLNG	EQU	*-EFHDRDS	LENGTH OF FUNCTION MODEL HEADER	00017000
EFXDDEDS	DSECT		07:36:43 05/31/84	00001000
*			EVAL FUNCTION MODEL XDE DSECT	00002000
EFXNEXT	DS	H	OFFSET TO NEXT MODEL XDE	00003000
EFXNDEC	DS	XL1	NUMBER OF DECIMALS	00004000
EFXRCLCF	DS	XL1	RESULT LENGTH CALCULATION FLAG	00005000
EFXRCLCP	EQU	X'80'	ADD LENGTH	00006000
EFXRCLCS	EQU	X'40'	SUBT LENGTH	00007000
*			IF ZERO, IGNORE	00008000
EFXFLAG1	DS	XL1	FIRST FLAG	00009000
EFXF1MAN	EQU	X'80'	ON=MANDATORY, OFF=OPTIONAL	00010000
EFXF1TRU	EQU	X'40'	ON=TRUNCATE, OFF=ROUND	00011000
EFXF1RES	EQU	X'20'	RESULT CHARACTERISTICS DEFAULT	00012000
	SPACE	1		00013000
	DS	XL3	FILLER	00014000
EFXDCTN	DS	H	NUMBER OF ENTRIES IN DATA CONV TBL	00014500
EFXLNG1	EQU	*-EFXDDEDS	BASE LENGTH OF ENTRY	00015000
	SPACE	1		00016000

*				DATA TYPE CONVERSION TABLE	00017000
EFXCNVE	DSECT			CONVERSION TBL ENTRY DSECT	00018000
EFXSRCT	DS	XL1		SOURCE DATATYPE	00019000
EFXTART	DS	XL1		TARGET DATATYPE	00020000
EFXTARL	DS	H		TARGET LENGTH	00021000
		DS	XL2	FILLER	00022000
		SPACE	1		00023000
EFXDCTL	EQU	*-EFXSRCT		LENGTH OF ENTRY	00024000

XDEs and VXDEs

XDEs (expression description elements) and VXDEs (variable expression description elements) form the compiled representation of a process at runtime. During compilation, each process statement is converted into a series of XDEs that represent the operands and operations within each statement. The XDEs of the statements are strung together to form the compiled representation of the process. At runtime, the runtime system builds a VXDE for each XDE. VXDEs contain variable runtime information; the information in XDEs does not change.

The compiled representation of a function consists of one operand XDE/VXDE for each parameter and one function XDE/VXDE for the function. These XDE/VXDE pairs contain the following information:

- **Function XDE/VXDE:**
 - Name and address of the function's processing program module
 - Function number identifying the appropriate program within the processing program module
 - Number of function operands (parameters)
 - Address of the work area available to the processing program
 - Description and address of the function's result field
 - Address of the operand VXDE for the last parameter in the parameter list
- **Operand XDE/VXDE:**
 - Description and address of the operand (parameter)
 - Address of the operand VXDE for the previous parameter in the parameter list

The DSECTs for the XDE and VXDE are shown below.

The runtime use of the XDEs and VXDEs is described under "Runtime Processing" later in this appendix.

DSECT of the XDE (Expression Description Element)

```

SPACE 1
* * * * *
*      #XDEDS - EXPRESSION DESCRIPTION ELEMENT      *
* * * * *
* THIS COPY MEMBER IS INCLUDED IN #XDEDS.  IF ANY CHANGES ARE      *
* MADE HERE, PLEASE INSURE THAT ALL MODULES CONTAINING #XDEDS ARE    *
* REASSEMBLED.                                                    *
* * * * *
* THIS MACRO ALLOWS THE ADS MODULES TO MORE EASILY REFERENCE XDE    *
* FIELDS WHEN AN XDE IS BEING BUILT WITHOUT THE XDENEXT FIELD.      *
* * * * *
SPACE
XDEDATAD DS  0F          (REAL) DATA ADDRESS (OPERAND OR RESULT)
XDEBRNXT DS  0F          ** BRANCH OPERATOR XDES ONLY
* * * * *
* IF RELOCATABLE XDE MODE TO BE USED IN
* EVAL/ADSOXDES THE OFFSET
* OF BRANCH TARGET XDE
* FROM 1ST XDE ELSE REAL ADDRESS OF TARGET
XDEDTABO DS  H          (LOGICAL)DATA ADDRESS - ADCON TABLE OFFSET
XDEDDSPL DS  H          (LOGICAL) DATA ADDRESS - DISPLACEMENT
* NOTE THAT REAL ADDRESSES ARE DISTINGUISHED FROM LOGICAL ADDRESSES
* (TABLE OFFSET/DISPLACEMENT PAIRS) BY THE X'80' BIT OF THE HIGH
* ORDER ADDRESS BYTE - ON => REAL, OFF => LOGICAL.
SPACE

XDETG     EQU  X'80'     High order bit of DYN used as temporary
* * * * *
* flag during executable code generation
* with following meaning :
* ON  -> This XDE is a target of a BRC2 XDE
* OFF -> This XDE is a NOT a BRC2 target
* This bit will ALWAYS be OFF in ALL XDEs
* in a final FDB.
XDEDYN   DS  H          OFFSET INTO DYNAMIC AREA OF VXDE
XDEDTLN  DS  0H         OPERAND LENGTH (IN BYTES)
XDEBROFF DS  0H         ** BRANCH OPERATOR XDES ONLY
* * * * *
* OFFSET IN XDES OF BRANCH TARGET FROM
* BRANCH OPERATOR.  ONLY USED WHEN CONTIG.
* XDE MODE USED IN EVAL.
* ALWAYS USED IN ADSOXDES.
XDEBITDP DS  C          BIT DISPLACEMENT (FOR MB-BIN)
XDEBITLN DS  C          LEN (NBR OF BITS) (FOR MB-BIN)

```

```

          ORG  XDEDATLN
XDEEPWR DS  C          POWER 10-1      (FOR EDIT)
XDEEPLN DS  0C        PICTURE LENGTH  (FOR EDIT)
XDEESGN DS  C          SIGN CHARACTER  (FOR EDIT)
XDENODEC DS  C          NUMBER OF DECIMAL PLACES
XDEDATYP DS  C          OPERAND DATA TYPE
          SPACE

```

* XDE DATA TYPE EQUATES :

```

XDEDGRP EQU  0          GROUP
XDEDEBCD EQU  1        EBCDIC
XDEDHBIN EQU  2        BINARY HALFWORD
XDEDFBIN EQU  3        BINARY FULLWORD
XDEDSPAK EQU  4        PACKED DECIMAL (SIGNED)
XDEDUPAK EQU  5        PACKED DECIMAL (UNSIGNED)
XDEDSZON EQU  6        ZONED DECIMAL (SIGNED)
XDEDUZON EQU  7        ZONED DECIMAL (UNSIGNED)
XDEDFLTD EQU  8        DISPLAY FLOATING POINT
XDEDSFLT EQU  9        INTERNAL FLOAT (SHORT)
XDEDLFLT EQU 10        INTERNAL FLOAT (LONG)
XDEDBIT  EQU 11        BIT
XEDDBIN  EQU 12        BINARY DOUBLEWORD
XDEDFC   EQU 13        FIGURATIVE CONSTANT
XDEDMBIN EQU 14        MULTI-BIT BINARY (PL1 STYLE)
XDEVCHR  EQU 15        VARYING CHARACTER
XDEEDIT  EQU 16        EDIT INFO
XDEEDP   EQU 17        EDIT PICTURE
XDEGEXT  EQU 18        EXTERNAL GRAPHICS SO.....SI
XDEGINTEQU 19        INTERNAL GRAPHICS

```

* FOLLOWING EQU SHOULD ALWAYS REFLECT THE HIGHEST DATA TYPE

* !!!!!!!!!!!!!!!!!!! PLEASE NOTE : !!!!!!!!!!!!!!!!!!!!!!!!!!!!!

* !!!!! ANY CHANGE TO THE FOLLOWING EQUATE REQUIRES CHGS !!!

* !!!!! TO RHDCEVAL AND ADSOXDES.

* !!!

```
XDEDMXTP EQU  XDEGINTEQU  MAX DATA TYPE VALUE
```

* NOTE THAT RHDCEVAL CURRENTLY ONLY SUPPORTS BIT FIELDS IN LOGICAL

* OPERATIONS.

```

          SPACE
XDEOPTYP DS  0C        OPERATION/OPERAND CODE
XDEEPAD  DS  C          PAD CHARACTER   (FOR EDIT)
          SPACE

```

```
* XDE OPERATOR TYPE EQUATES :
XDEOPND EQU 0 OPERAND (NOT OPERATOR)
XDEOPNOT EQU 5 LOGICAL "NOT"
XDEOPOR EQU 6 LOGICAL "OR"
XDEOPAND EQU 7 LOGICAL "AND"
XDEOPCJ EQU 9 CLASS OF LOGICAL CONJUNCTIONS
XDEOPEQ EQU 10 "EQ" (RELATIONAL OPERATION)
XDEOPNE EQU 11 "NE"
XDEOPLT EQU 12 "LT"
XDEOPLE EQU 13 "LE"
XDEOPGT EQU 14 "GT"
XDEOPGE EQU 15 "GE"
XDEOMTCH EQU 16 "MATCHES"
XDEOPCON EQU 17 "CONTAINS"
XDEOPCMP EQU 18 "COMPARE"; 8:EQUAL, 4:<, 2:>

XDEOPREL EQU 19 CLASS OF RELATIONAL OPERATORS
XDEOUNMN EQU 20 UNARY "-"
XDEOADDN EQU 21 "+"
XDEOBNMN EQU 22 BINARY "-"
XDEOMULT EQU 23 "*"
XDEOPDIV EQU 24 "/"
XDEOPDVR EQU 25 "/" WITH REMAINDER
XDEOPART EQU 29 CLASS OF ARITHMETIC OPERATORS
* NOTE : Test Under Mask used as pseudo-operator to
* process "boolean variables", e.g. Map Status tests.
* This operator will never appear directly
* in an XDE list.
```



```

XDEOPTM EQU 35          "TEST UNDER MASK"
XDEOASGN EQU 40         "ASSIGNMENT"
* NOTE : MULTIPLE ASSIGNMENT NO LONGER SUPPORTED
*       Operator code available for reuse.
*XDEOASGM EQU 41        "ASSIGNMENT",MULTIPLE TARGETS
XDEOASGR EQU 42         REVERSE ASSIGNMENT
XDEOINDX EQU 45         ARRAY "INDEX"
*
* NOTE : DATE CONVERSIONS NO LONGER SUPPORTED.
*       THESE CODES ARE AVAILABLE FOR REUSE.
*
*DEODTJA EQU 50         JULIAN DATE TO GREGORIAN (AMER.
*DEODTJG EQU 51         JULIAN DATE TO GREGORIAN (WORLD
*DEODTAJ EQU 52         GREGORIAN DATE (AMER. - MMDDYY)
*DEODTGJ EQU 53         GREGORIAN DATE (WORLD - DDDMMYY)
*DEODTES EQU 55         CLASS OF DATE CONVERSIONS
* NOTE : CONCATENATE NO LONGER SUPPORTED.
*       Operator code available for reuse.
*
*XDEOCONC EQU 60        "CONCATENATION"
* NOTE : INDA ONLY USED BY DEBUGGER.
*       Not supported by ADSOXDES.

XDEOINDA EQU 65         INDIRECT ADDRESSING
** Following is a "pseudo-opcode" used only in generation
** of machine code for True/False DXBs. This opcode will NEVER
** appear in an XDE list. It is used only for convenience
** so that generating code for T/F DXBs easily fits into the
** standard methodology of code generation.
XDEODXTF EQU 78         True/False DXB
XDEOBRCH2 EQU 79        BRANCH OPERATOR
XDEOBRCH EQU 80         BRANCH OPERATOR
* NOTE : IF A NEW OPERATOR CODE IS ADDED WHICH IS
*       GREATER THAN THE CURRENT VALUE OF XDEOMXTP,
*       WE MUST CHANGE XDEOMXTP TO THIS NEW VALUE.
*       IN THIS CASE, WE MUST ALSO ADD ENTRIES TO THE
*       RHDCEVAL/ADSOXDES OPTABLE.
*       TO MINIMIZE THE SIZE OF OPTABLE,
*       IT WOULD BE BEST TO ASSIGN NEW CODES <=
*       THE CURRENT VALUE OF XDEOMXTP.
*       RHDCEVAL/ADSOXDES CURRENTLY ASSUME THAT THE ONLY
*       VALID OPTYP > XDEOMXTP IS XDEOUFUN AND THIS IS
*       HANDLED AS A SPECIAL CASE.

```

XDEOMXTP	EQU	XDEOBRCH	MAX OPTYP VALID FOR USE WITH
*			RHDCEVAL/ADSOXDES OPTABLE.
XDEADSLR	EQU	253	"OF LR" OPERAND (USED BY ADS/ONLINE)
*			CODE NOT USED FOR AN OPERATOR
XDEOKWD	EQU	254	KEYWORD - USED BY LRF.
*			ALL OTHERS TREAT AS OPERAND
XDEOUFUN	EQU	255	USER-DEFINED FUNCTION
XDEFLAG	DS	C	FLAG BYTE
XDEFNVL	EQU	X'80'	FIELD IS NOT VALUED
XDEFNED	EQU	X'40'	NO DATA VALIDATION NEEDED
*			(FOR PACKED/ZONED FIELDS)
XDEFNCV	EQU	X'20'	NO CONVERSION NEEDED
XDEADDR	EQU	X'10'	XDEDATAD IS OPRND, VS OPRND ADR
*			XDEADDR USED ONLY BY DEBUGGER
XDEFFCZ	EQU	X'08'	FIGURATIVE CONSTANT ZERO
XDEBTF	EQU	X'08'	** BRANCH OPERATOR XDES ONLY
*			ON => BRANCH IF PREVIOUS RESULT TRUE
*			OFF => BRANCH IF PREVIOUS RESULT FALSE
XDEBNEG	EQU	X'04'	** BRANCH2 OPERATOR XDES ONLY
*			If we branch to last XDE, final
*			result is value at top of XDE stack.
*			ON => This value must be negated
*			OFF => Value is correct as is
XDEBEND	EQU	X'02'	** BRANCH2 OPERATOR XDES ONLY
*			Special flag for branch on last XDE
*			in list to say final value must be
*			negated.
XDEFTRUN	EQU	X'02'	TRUNCATE IF DST DEC < SRC DEC
XDEFRQST	EQU	X'01'	USED BY LOGICAL RECORD PROCESSING
		SPACE 1	
*			FLAG CODES FOR EDIT
XDEEF99	EQU	X'01'	SIGNIFICANCE ON HI ORDER
XDEEFLT	EQU	X'04'	FLOAT THE SIGN CHARACTER
*			Blank on zero flag never utilized by any EVAL callers
*DEEFBZ	EQU	X'08'	BLANK ON ZERO
XDEEFPI	EQU	X'10'	XDEEADR POINTS TO PICTURE
XDEEFJL	EQU	X'20'	LEFT JUSTIFY OUTPUT
XDEEFNE	EQU	X'40'	PICTURE IS ALL X'S
XDEEFNS	EQU	X'80'	DO NOT SCALE SOURCE
*			
XDELEN1	EQU	*-XDE	LENGTH OF STANDARD XDE
		SPACE	

* FOR USER DEFINED FUNCTIONS, SEVERAL ADDITION FLDS ARE REQUIRED

```
XDEUPGMN DS CL8 PROGRAM NAME
XDEUNOPS DS XL1 NBR OPERANDS
XDEUFUNC DS XL1 FUNCTION NUMBER
XDEUSTLN DS H REQ'D STORAGE LENGTH
XDEUPGMV DS H PROGRAM VERSION
XDEUFLG1 DS XL1 USER FUNCTION FLAG BYTE
XDEUAGFU EQU X'80' AGGREGATE FUNCTION
DS XL1 UNUSED
XDELEN2 EQU *-XDE LENGTH OF "USER FUNCTION" XDE
SPACE
```

* EQUATES FOR RESULTS OF COMPARISONS

```
XDECMPEQ EQU X'08' RESULT OF COMPARE IS =
XDECMPLT EQU X'04' RESULT OF COMPARE IS <
XDECMPGT EQU X'02' RESULT OF COMPARE IS >
SPACE
```

DSECT of the VXDE (Variable Expression Description Element)

```
* * * * *
* THE VXDE IS THE DYNAMIC (WRITABLE) PORTION OF THE XDE *
* * * * *
```

```
VXDE DSECT 11:16:59 04/14/87
SPACE
```

```
VXDEFLAG DS 0C FLAG BIT FOR NON-VALUED RESULT
VXDEFNVL EQU X'80' NON-VALUED RESULT
VXDESNXT DS F OPERAND STACK NEXT XDE ADDR
VXDEFLG2 DS 0C FLAG BIT FOR ALREADY VALIDATED DECIMAL
VXDEFNED EQU X'80' ALREADY VALIDATED DECIMAL
VXDEXDEA DS F CORRESPONDING XDE ADDRESS
VXDEDADR DS F REAL DATA FIELD ADDRESS
VXDEDLEN EQU *-VXDE
SPACE 1
```

```
* FOR USER-DEFINED FUNCTIONS, THE FOLLOWING FLDS
* ARE ALSO REQUIRED.
SPACE 1
```

```

VXDEUPGA DS    F          PROGRAM ADDR
VXDEUWKA DS    0F        WORK AREA ADDR
VXDEUWTO DS    H          LOGICAL ADDR  TBL OFFSET
VXDEUWDS DS    H          LOGICAL ADDR  DISPL
VXDEUFLG DS    CL1       FLAG FOR USER FUNCTIONS
VXDEUBRK EQU   X'80'     AGGREGATE FUNCTION BREAK
VXDEUINT EQU   X'40'     AGGREGATE FUNCTION INIT
VXDEUNIV EQU   X'20'     NO INITIAL VALUE FOR AGG FUN BREAK
VXDEUBAD EQU   X'10'     BAD DATA WITHIN BREAK
VXDEUOVR EQU   X'08'     OVERFLOW WITHIN BREAK
VXDELOD  #FLAG X'04'     USER PGM WAS #LOADED
          DS    XL3       UNUSED
VXDEDLN2 EQU   *-VXDE    LNG OF EXTENSION
    
```

SPACE

```

* * * * *
*   THE XDEIX IS THE DOPE VECTOR USED IN "INDEX" OPERATIONS TO DEFINE *
*   THE FORMAT OF THE ARRAY DIMENSIONS BEING ADDRESSED                *
* * * * *
    
```

XDEIX DSECT

```

XDEIXDOA DS    F          "DEPEND ON" CONTROL FIELD ADDR
XDEIXNDM DS    H          NUMBER OF DIMENSIONS (IN ARRAY)
XDEIXFLG DS    C          FLAG BYTE
XDEIXFDF EQU   X'80'     FULLWORD "DEPENDS ON" CONTROL FIELD
XDEIXFDH EQU   X'40'     HALFWORD "DEPENDS ON" CONTROL FIELD
          DS    C          UNUSED BYTE
XDEIXRLN EQU   *-XDEIX   LENGTH OF DDOPE VECTOR ROOT
    
```

SPACE

```

*   THE FOLLOWING FIELDS ARE REPEATED ONCE FOR EACH DIMENSION IN
*   THE ARRAY - FOR MBB TABLES, OFFSET AND SIZE ARE IN BITS
    
```

SPACE

```

XDEIXOFF DS    H          FIELD OFFSET WITHIN CONTAINING OCCURRENCE
XDEIXSIZ DS    H          SIZE OF A DIMENSION OCCURRENCE
XDEIXMAX DS    H          MAXIMUM SUBSCRIPT VALUE FOR DIMENSION
XDEIXDLN EQU   *-XDEIXOFF LENGTH OF ONE DIMENSION DESCRIPTOR
XDEIXLMT EQU   15        Maximum number of dimensions supported by ADS
EJECT
    
```

Processing Program Modules

Processing program modules contain one or more programs; each program processes one function. When a function XDE/VXDE is processed at runtime, the runtime system calls the appropriate processing program module. The module performs the operation, then returns control to the runtime system. The components of the source module RHDCEV01, which contains the processing programs for the CA ADS supplied string functions are shown below.

Processing program load modules are usually stored in the load library. The load modules for the CA ADS supplied built-in functions are named RHDCEV01, RHDCEV02, RHDCEV03, and RHDCEV09.

Components of Processing Program Module RHDCEV01

```

RHDCEV01 TITLE 'STRING PROCESSOR FOR RHDCEVAL'                                * RHDCEV01 EP=EV01EP1
06/29/90 14:04:31
*CONTAINS PTF# 90-05-1133                                                    EXG 05/31/90
*CONTAINS PTF# 88-07-1081                                                    JMA 02/26/90
*CONTAINS PTF# 87-06-1031                                                    MCM 08/21/87
*CONTAINS PTF# 85-08-S004                                                    MCM 03/31/86
* CONTAINS PTF # LEFT/RITE JUST SPA CRM 14:37:29 01/14/85
* CONTAINS PTF # 84-11-1067          CRM 13:37:25 12/14/84
  SPACE 1
  #MOPT CSECT=RHDCEV01,ENV=USER
  SPACE 3
*****
*
* RHDCEV01 IS THE STRING PROCESSOR FOR RHDCEVAL. ALL
* EVAL STRING-HANDLING FUNCTIONS ARE CONTAINED HEREIN.
*
* THESE FUNCTIONS ARE:
* LENGTH - RETURN LENGTH OF A CHARACTER STRING
* SUBSTRING - RETURN A SUBSET OF A STRING
* INDEX - FIND POSITION OF A SUBSTRING
* VERIFY - INSURE ONE STRING CONTAINS ANOTHER
* REPLACE - TRANSLATE CHARACTERS
* CONCATENATE - SHOVE TWO OR MORE STRINGS TOGETHER
* LIKE - STRING PATTERN MATCHING
*
* UPON ENTRY, R1 MUST CONTAIN THE ADDRESS OF THE OPERATION
* VXDE, WHICH IS BACK-CHAINED TO ALL OPERAND VXDE'S.
*
* ALL STRING INPUT AND OUTPUT WILL BE VARYING-CHARACTER.
* ALL NUMERIC INPUT AND OUTPUT WILL BE HALFWORD-BINARY.
*
*****
EJECT
RHDCEV01 CSECT
RHDCEV01 AMODE ANY
RHDCEV01 RMODE 24
        USING EV01EP1,R12          PROGRAM BASE
        USING EV01EP1+4096,R10     USE SECOND BASE REGISTER
        USING WORKAREA,R11        WORKAREA BASE
        USING XDE,R8
        USING VXDE,R7
        SPACE 1
        ENTRY EV01EP1
EV01EP1 DS 0H
        STM R14,R12,12(R13)       SAVE REGISTERS
        LR R12,R15                SET PROGRAM BASE
        L R10,BASE                SET UP SECOND BASE REGISTER
        B EV01STRT                AND GO START UP
BASE DC A(EV01EP1+4096)
EV01STRT DS 0H
        LR R7,R1                  GET RESULT VXDE ADDR
        L R8,VXDEXDEA             AND XDE ADDR
        L R11,VXDEUWKA            GET WORKAREA ADDR
        STM R7,R8,WKRESADR        AND SAVE THEM
        SPACE 1
        MVI WKERRMSG,C' '         NOW BLANK OUT
        MVC WKERRMSG+1(L'WKERRMSG-1),WKERRMSG  ERROR MSG FIELD
        SPACE 1
        SLR R2,R2                 CLEAR FOR NEXT INST
        IC R2,XDEUFUNC            GET FUNCTION NUMBER
        CLI XDEUFUNC,15           CK AGAINST MAX FUNCTION  JMA90179
        BH EV01NFC                BIF HIGH TO ERR EXIT
        SLL R2,2                  MAKE FUNCT NBR MULTIPLE OF 4
        B EV01BTB1(R2)            AND GO SELECT FUNCTION
        SPACE 1
EV01BTB1 DS 0H
        B LENGTH                  FUNC 0
        B SUBSTRNG                FUNC 1
        B INDEX                   FUNC 2
        B VERIFY                  FUNC 3
        B TRANSLAT                FUNC 4
        B CONCATEN                FUNC 5
        B REPEAT                  FUNC 6
    
```

Initialization statements

Branching statements

```

B   EXTRACT          FUNC 7
B   REPLACE          FUNC 8
B   LEFTJUS         FUNC 9
B   RITEJUS         FUNC 10
B   INSERT          FUNC 11
B   LIKE            FUNC 12
B   GOODTRL        FUNC 13
B   TRAILZN        FUNC 14
B   ZNTRAIL        FUNC 15

EJECT
EV01NFC DS 0H NO FUNCTION EXIT
LA R15,4 SET ERROR CODE
MVC WKERRMSG(L'ERMSG01),ERMSG01 SET ERROR MSG
B EV01RET AND GET OUT
SPACE 3
EV01INVAL DS 0H *MCM86253*
OI VXDEFLAG,VXDEFNVL RESULT IS NON-VALUED *MCM86253*
SPACE 3 *MCM86253*
EV01RET0 DS 0H GOOD EXIT
SLR R15,R15 SET GOOD RETURN CODE
SPACE 1
EV01RET DS 0H
L R14,12(R13) RESTORE R14
LM R0,R12,20(R13) RESTORE REGS 0-12
BR R14 AND RETURN TO CALLER
EJECT
*****
* ERROR MESSAGES *
*****
SPACE 2
ERMSG01 DC C'UNSUPPORTED STRING FUNCTION REQUESTED'
ERMSG02 DC C'INVALID OBJECT STRING LENGTH'
ERMSG03 DC C'INVALID START VALUE'
ERMSG04 DC C'INVALID LENGTH VALUE'
ERMSG05 DC C'String TO BE EXTRACTED EXCEEDS OBJECT LENGTH'
ERMSG06 DC C'RESULT STRING TOO SMALL TO CONTAIN SUBSTRING'
ERMSG07 DC C'INVALID SEARCH STRING LENGTH'
ERMSG08 DC C'SEARCH STRING LENGTH EXCEEDS OBJECT STRING LENGTH'
ERMSG09 DC C'INVALID STRING LENGTH'
ERMSG10 DC C'RESULT STRING NOT LARGE ENOUGH'
ERMSG11 DC C'INVALID INSERTION VALUE'
ERMSG12 DC C'INVALID PATTERN FOR LIKE COMPARISON'
ERMSG13 DC C'ESCAPE CHARACTER LENGTH GREATER THAN 1'
ERMSG14 DC C'INVALID ESCAPE CHARACTER STRING'
EJECT
*****
*
* LENGTH - STRING FUNCTION TO RETURN THE LENGTH OF *
* A VARYING-CHARACTER FIELD. *
*
* ONLY REQUIRES ONE OPERAND, THE VARYING-CHAR FIELD. *
* THE RESULT FIELD MUST BE HALFWORD-BINARY. *
*
*****
SPACE 2
LENGTH DS 0H
L R5,VXDESXNT GET ADDR OF OPERAND VXDE
L R6,VXDEXDEA-VXDE(,R5) AND OPERAND XDE *MCM86253*
BAL R14,CHKNOVAL *MCM86254*
LTR R15,R15 *MCM86254*
BNZ EV01INVAL *MCM86254*
L R4,VXDEDADR-VXDE(,R5) GET ADDR OF VC FLD
MVC WKFULL(2),0(R4) MOVE HALFWORD TO ALIGN
LH R4,WKFULL GET LENGTH OF FIELD
L R5,VXDEDADR GET ADDR OF RESULT FLD
STCM R4,3,0(R5) SET ANSWER -STCM FOR BS2K*MCM86090*
B EV01RET0 USE GOOD EXIT
LTOrg
EJECT
*****
*
* SUBSTRING - STRING FUNCTION TO RETURN A SPECIFIED *
* SUBSET OF A GIVEN STRING *

```

Final processing statements

Error messages

```

*
* THIS FUNCTION REQUIRES 3 OPERANDS -
* 1 OBJECT STRING (VARYING-CHARACTER)
* 2 START DISPLACEMENT (HALFWORD)
* 3 LENGTH (OPTIONAL) (HALFWORD)
* THE RESULT FIELD MUST BE VARYING-CHARACTER ALSO.
*
* THE OBJECT STRING MAY NOT BE LENGTH ZERO.
* IF AN ERROR IS DETECTED, THE RESULT STRING LENGTH
* IS SET TO ZERO, AND AN ERROR RETURNED TO RHDCEVAL.
*
* REQUIREMENTS OF THE OPERANDS ARE:
* K = OBJECT STRING LENGTH, I = START DISPLACEMENT,
* J = LENGTH
*
* 0 LE J LE K          1 LE I LE K
* I+J-1 LE K
*
* IF J IS NOT GIVEN, J = K-I+1
*
* THE OMISSION OF J (3RD OPERAND - LENGTH) IS INDICATED
* BY A NON-VALUED XDE.
*
*****
SPACE 3
SUBSTRNG DS 0H
* NOTE : SUBSTRING OP3 IS AN OPTIONAL PARAMETER. 4*
* MUST DIFFERENTIATE BETWEEN OP3 OMITTED 4*
* AND OP3 SPECIFIED BUT NON-VALUED. 4*
L R5,VXDESNXT BACK UP TO OP3 VXDE
L R6,VXDEXDEA-VXDE(,R5) AND XDE
TM VXDEFLAG-VXDE(R5),VXDEFINVL IF OP3 VXDE NON-VA *MCM86260*
BO EV0INVAL THEN SO IS RESULT *MCM86260*
TM XDEFLAG-XDE(R6),XDEFINVL IF OP3 XDE NON-VAL *MCM86260*
BO SUBS0010 THEN CHK FURTHER *MCM86260*
B SUBS0040 CONTINUE WITH OP2 *MCM86260*
SPACE 1 *MCM86260*
SUBS0010 DS 0H *MCM86260*
CLC XDEDATAD-XDE(,R6),=X'80000000' OP3 OMITTED?*MCM86260*
BE SUBS0040 YES - CONTINUE *MCM86260*
B EV0INVAL NO - OP3 NON-VALUED *MCM86260*
SPACE 1 *MCM86260*
SUBS0040 DS 0H *MCM86260*
STM R5,R6,WKOP3SV SAVE OP3 XDE,VXDE *MCM86260*
L R5,VXDESNXT-VXDE(,R5) BACK UP TO OP2 VXDE
L R6,VXDEXDEA-VXDE(,R5) AND XDE
BAL R14,CHKNOVAL
*MCM86254*
LTR R15,R15
*MCM86254*
BNZ EV0INVAL
*MCM86254*
STM R5,R6,WKOP2SV SAVE OP2 XDE,VXDE
*MCM86253*
SPACE 1
L R5,VXDESNXT-VXDE(,R5) BACK UP TO OP1 VXDE
L R6,VXDEXDEA-VXDE(,R5) AND XDE
BAL R14,CHKNOVAL
*MCM86254*
LTR R15,R15
*MCM86254*
BNZ EV0INVAL
*MCM86254*
STM R5,R6,WKOP1SV SAVE OP1 XDE,VXDE
*MCM86253*
EJECT
*****
*
* OBJECT STRING LENGTH MUST BE GREATER THAN ZERO *****
L R4,VXDEDADR-VXDE(,R5) GET OP1 DATA ADDR
MVC WKFULL,0(R4) GET HALFWORD LNG FROM VC FLD
LH R4,WKFULL PUT INTO A REGISTER
LTR R4,R4 CK IT FOR ZERO
BP SUBS0050 GTR ZERO IS OKAY - BRANCH
SPACE 1
MVC WKERRMSG(L'ERMSG02),ERMSG02 SET ERROR MSG

```

Processing program

Processing program (cont'd)


```

B      SUBS0950          USE ERROR EXIT
SPACE 1
SUBS0065 DS      0H          R4 NOW CONTAINS LENGTH OF OBJECT STRING
***** CHECK STARTING DISPLACEMENT *****
LM     R7,R8,WKOP2SV     GET VXDE/XDE ADDRS, OP2
L      R3,VXDEDADR      GET OP2 DATA ADDR
MVC   WKFULL,0(R3)      GET HALFWORD DATA FIELD
LH    R3,WKFULL         GET THE VALUE
LTR   R3,R3             CK FOR ZERO OR LESS
BP    SUBS0080          IF POSITIVE, BRANCH
SPACE 1
SUBS0075 DS      0H          INVALID START FIELD
MVC   WKERRMSG(L'ERMSG03),ERMSG03 SET ERROR MSG
B      SUBS0950          USE ERROR EXIT
SPACE 1
SUBS0080 DS      0H
CR    R3,R4             COMPARE TO MAX START
BH    SUBS0075          ERR IF START GTR LENGTH
SPACE 1
***** R3 NOW HAS STARTING DISPLACEMENT RELATIVE TO ONE *****
***** NOW GET EXTRACT LENGTH, WHICH MIGHT HAVE BEEN OMITTED ***
LM     R7,R8,WKOP3SV     GET VXDE/XDE ADDRS
TM    XDEFLAG,XDEFNVL   CK FOR PARAMETER OMITTED
BZ    SUBS0090          BIF IT IS PRESENT
SPACE 1
SUBS0085 DS      0H
LR    R2,R4             ELSE SET
SR    R2,R3             EXTRACT LNG TO
LA    R2,1(,R2)         TOTAL-START+1
B      SUBS0120          AND BYPASS NEXT EDIT
SPACE 1
SUBS0090 DS      0H
L      R2,VXDEDADR      GET DATA ADDR, OP3
MVC   WKFULL,0(R2)      GET HALFWORD LENGTH
LH    R2,WKFULL         PUT INTO A REGISTER
LTR   R2,R2             CK FOR ZERO OR LESS
BZ    SUBS0085          ZERO - TAKE DEFAULT ABOVE
BP    SUBS0100          POSITIVE IS OKAY - BRANCH
SPACE 1
SUBS0095 DS      0H          INVALID LENGTH FIELD
MVC   WKERRMSG(L'ERMSG04),ERMSG04 SET ERROR MESSAGE
B      SUBS0950          USE ERROR EXIT
SPACE 1
SUBS0100 DS      0H
CR    R2,R4             MUST BE LESS THAN OBJECT-LNG
BH    SUBS0095          IF NOT, ERROR
EJECT
SUBS0120 DS      0H
***** INSURE START + EXTRACT LNG DOESN'T EXCEED OBJECT STRING
LNG**
LR    R1,R3             GET START DISPLACEMENT
AR    R1,R2             ADD EXTRACT LENGTH
BCTR  R1,0             DECREMENT BY ONE
CR    R1,R4             COMPARE TO TOTAL AVAIL
BNH   SUBS0140          EQ OR LOW IS OKAY - BRANCH
SPACE 1
MVC   WKERRMSG(L'ERMSG05),ERMSG05 SET ERROR MESSAGE
B      SUBS0950          USE ERROR EXIT
SPACE 1
SUBS0140 DS      0H
***** R4 HAS TOTAL STRING LENGTH OF OBJECT *****
***** R3 HAS DISPLACEMENT TO START OF EXTRACT *****
***** R2 HAS LENGTH TO EXTRACT *****
***** MUST NOW TEST RESULT FIELD SIZE TO INSURE IT CAN *****
***** CONTAIN THE EXTRACTED SUBSTRING *****
SPACE 1
LM     R7,R8,WKRESADR     GET VXDE/XDE ADDRS OF RESULT
LH    R5,XDEDATLN       GET MAX RESULT SIZE
CR    R2,R5             EXTRACT LNG CAN'T EXCEED TARG LEN
BNH   SUBS0150          BRANCH IF OKAY
SPACE 1
MVC   WKERRMSG(L'ERMSG06),ERMSG06 SET ERROR MESSAGE
B      SUBS0950          USE ERROR EXIT
SPACE 1

```

Processing program (cont'd) →

Processing program (cont'd)

```

SUBS0150 DS 0H      NOW READY TO EXTRACT THE SUBSTRING
          L  R5,VXDEDADR  GET RESULT FLD ADDR
          STH R2,WKFULL   ALIGN THE SUBSTRING LENGTH
          MVC 0(2,R5),WKFULL  PUT LENGTH FIELD INTO RESULT (VC)
          LA  R5,2(,R5)     AND ADVANCE RESULT FIELD POINTER
          SPACE 1
          L  R6,WKOP1SV   GET VXDE ADDR OBJECT STRING
          L  R6,VXDEDADR-VXDE(,R6)  GET DATA ADDR
          LA  R6,2(,R6)    BUMP PAST LENGTH FIELD
          BCTR R3,0        MAKE START RELATIVE TO ZERO
          AR  R6,R3       CALC ADDR OF SUBSTRING
          SPACE 1
          LR  R4,R2       GET LENGTH IN RIGHT REGISTER
          BAL R14,MOVEIT  MOVE THE SUBSTRING
          B   SUBS0980    AND USE SUCCESS EXIT
          EJECT

SUBS0950 DS 0H      ERROR EXIT
          LM  R7,R8,WKRESADR  GET RESULT VXDE/XDE ADDRS
          L  R6,VXDEDADR  GET DATA FIELD ADDR
          XC 0(2,R6),0(R6)  SET LNG TO NULL
          LA  R15,4       SET ERROR RETURN CODE
          B   EV01RET     AND USE ERROR EXIT
          SPACE 2

SUBS0980 DS 0H      SUCCESS EXIT
          B   EV01RET0    USE GOOD EXIT
          LTORG
          EJECT
    
```

Work area storage definition

```

*****
*
*
*
*      WORKAREA - PASSED BY CALLER
*
*
*****
*
          SPACE 2
WORKAREA DSECT
WKERRMSG DS CL80
WKFULL   DS F
          ORG WKFULL
WKHALF   DS H          *THIS CAN'T BE USED WITH WKFULL !!
WKFLAG1 DS CL1        WORK FLAG1
WKFIASR EQU X'80'     GOT ME AN ARBITRARY STRING WORKING
WKFIOP30 EQU X'40'    ESCAPE CHAR IN REQUEST
WKFIOP35 EQU X'20'    ESCAPE CHAR ENCOUNTERED
WKFI1X10 EQU X'10'    X'10' BIT FOR FLAG1
WKFI1PON EQU X'08'    PROCESSING % OPERATOR
WKFI1PSET EQU X'04'   1ST CHAR IN % STR MATCHED IN OBJ
          WKFI1X02 EQU X'02'  X'02' BIT FOR FLAG1
          WKFI1X01 EQU X'01'  X'01' BIT FOR FLAG1
WKFLAG2 DS CL1        WORK FLAG2
WKRESADR DS 2F        VXDE/XDE ADDRS, RESULT FIELD
WKPSLADR DS F         RESULT ADDR
WKPSLLNG DS F         RESULT LENGTH
WKOP1SV  DS 2F        VXDE/XDE ADDRS, OPERAND 1 FIELD
WKOP2SV  DS 2F        VXDE/XDE ADDRS, OPERAND 2 FIELD
          ORG WKOP2SV
WKPATCNT DS F         STARTING COUNT FOR % PATTERN
WKPATADR DS F         STARTING POSITION FOR % PATTERN
WKOP3SV  DS 2F        VXDE/XDE ADDRS, OPERAND 3 FIELD
          ORG WKOP3SV
WKOBJCNT DS F         STARTING COUNT FOR % PATTERN
WKOBJADR DS F         STARTING POSITION FOR % PATTERN
WKOP1LNG DS F         OPERAND 1 LENGTH
WKOP2LNG DS F         OPERAND 2 LENGTH
WKOP3LNG DS F         OPERAND 3 LENGTH
WKOP1ADR DS F         OPERAND 1 ADDRESS
WKOP2ADR DS F         OPERAND 2 ADDRESS
WKOP3ADR DS F         OPERAND 3 ADDRESS
          SPACE 2
    
```

```

WKLENGTH EQU *-WORKAREA          LENGTH OF WORKAREA
EJECT
COPY #XDEDS
END EV01EP1

```

Runtime Processing of Built-In Functions

At runtime, the following processing sequence occurs for each function:

1. The **runtime system** begins processing the function, as follows:
 - Moves each parameter in the function to an intermediate result area (IRA). If a parameter is coded as a multi-operand expression, the expression is evaluated and only the result is moved to the IRA. Data conversions are performed as necessary.

The runtime system maintains an operand XDE/VXDE for each function parameter. The XDE/VXDE pair describes the parameter as it is stored in the IRA and contains the parameter's IRA address.
 - Passes control to the processing program module named in the function XDE. The runtime system places in register 1 the address of the function VXDE. (A function VXDE contains addresses that enable access to the work area, the result field in the IRA, and the function's operands/VXDEs.)
2. The **processing program** continues processing the function, as follows:
 - Processes initialization statements, as illustrated earlier in this appendix. Register information is saved by using standard z/OS conventions. Register 1 is used to access the function VXDE, which in turn enables access to all the information required for processing the function, as illustrated below.
 - Branches to the appropriate program by using a branching routine in conjunction with the XDEUFUNC field of the function XDE.
 - Processes the program statements.
 - Processes final statements. If the program did not encounter an error condition, register 15 is set to 0. If an error was encountered, register 15 is set to a nonzero value and an optional error message is moved to the first 80 bytes of the work area. Registers are restored and the module passes control back to the runtime system.
3. The **runtime system** finishes processing the function by checking the value returned in register 15. If register 15 equals 0, the runtime system resumes the process. The result of the function is used in the statement in which the function is coded. If register 15 is greater than 0, the runtime system aborts the dialog and displays the Dialog Abort Information screen along with the optional error message.

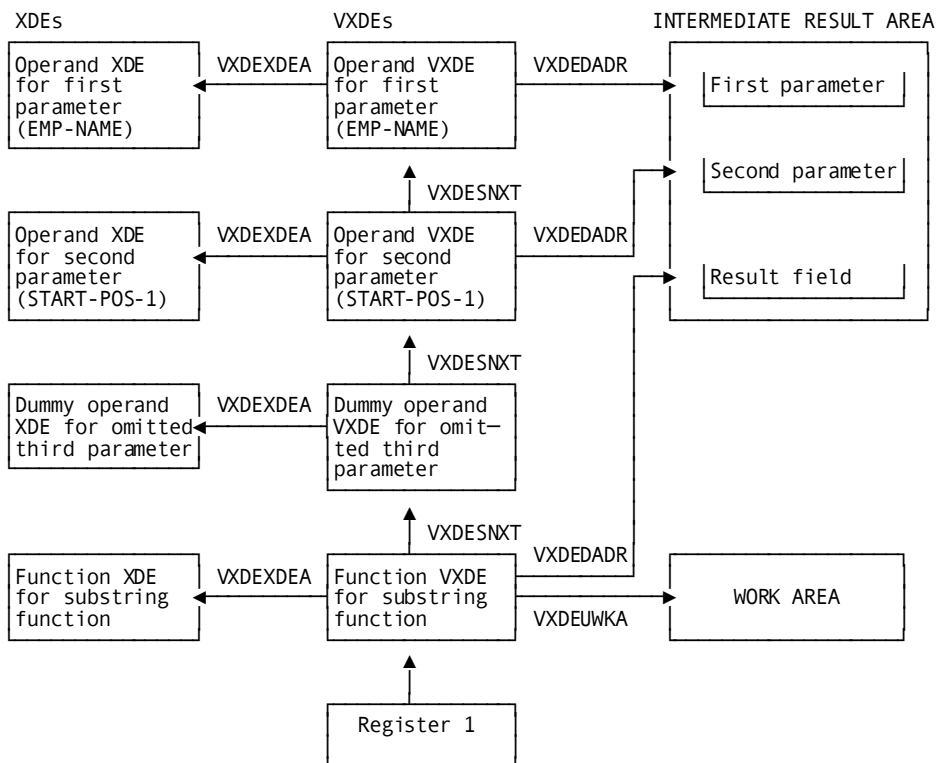
Internal Representation of a Function at Run Time

The internal representation of a substring function is illustrated below.

An arrow indicates that the source structure contains the address of the object structure, illustrating that the processing program module can use register 1 to gain access to all required information.

Field names containing the addresses are listed next to the arrows.

SUBSTRING(EMP-NAME, START-POS-1)



More information:

[Steps for Generating a User-Defined Built-In Function](#) (see page 714)

Assembler Macros

Assembler macros (#EFUNMST and #EFUNMOD) are used in assembler programs to define the master function table and the model XDE modules. The programs are assembled to create object modules. The object module for the master function table is then placed in the data dictionary load area by using the DDDL compiler. The object module for a model XDE module is placed in the load library by using the linkage editor.

The macros #EFUNMST and #EFUNMOD are discussed separately below.

#EFUNMST

Purpose

Defines the master function table. Three types of #EFUNMST macros are coded in a source assembler program, as follows:

Syntax

```

▶▶—— #EFUNMST TYPE = ————┌ INITIAL —————▶
                           └ FINAL —————▶
                           ENTRY, INVOKE = invocation-name — , ————▶
▶————▶
——— FUNCT = ————┌ function-name ————▶
                  └ * —————▶
▶————▶
——— PROGRAM = ————┌ model-xde-module ————┐
                  └ * —————┐ , VERSION = version-number ┐
▶————▶

```

Parameters

INITIAL

Generates header information and automatically generates the #EFUNMST TYPE=ENTRY macros for the CA ADS supplied built-in functions.

TYPE=INITIAL is coded first and only once in the assembler program.

FINAL

Defines the end of the table.

ENTRY

Generates an entry in the master function table.

TYPE=ENTRY macros are coded once for each entry in the table.

INVOKE= *invocation-name*

Specifies a user-defined, 1- to 32-character invocation name for the function.

FUNCT= *function-name*/*

Specifies a user-defined, 1- to 8-character real (generic) function name for the function.

This function name is used to associate the coded invocation name with the model XDE table that describes the function in the model XDE module. The character * can be specified if the real function name is the same as the real function name for the previous entry in the master function table.

PROGRAM= *model-xde-module*

Specifies the 1- to 8-character name of the model XDE module in which the function is described.

,VERSION= *version-number*

Specifies the 1- to 4-digit version number of the model XDE module in which the function is described.

*

The character * can be specified if the model XDE module name is the same as the model XDE module name for the previous entry in the master function table (see RHDCEVBF below).

Usage

Considerations

- A source module must begin with one TYPE=INITIAL macro and end with one TYPE=FINAL macro.
- Any number of TYPE=ENTRY macros can be coded between the INITIAL and FINAL type macros.

RHDCEVBF

The master function table is defined in a source assembler program called RHDCEVBF. RHDCEVBF is shown below as it appears when CA ADS is installed. Entries for user-defined functions are defined by coding #EFUNMST TYPE=ENTRY macros between the INITIAL and FINAL type macros.

Source Assembler Program RHDCEVBF

```

RHDCEVBF TITLE 'EVAL - BUILT-IN FUNCTIONS - MASTER TABLE'
* RHDCEVBF EP=RHDCEVBF                                06/25/90 14:52:50
  #EFUNMST TYPE=INITIAL                               12/08/88 15:52:14
  EJECT
  #EFUNMST TYPE=FINAL
  END

```

The TYPE=INITIAL macro automatically generates the entries for the CA ADS supplied built-in functions. It does this by copying the TYPE=ENTRY macros coded in the source module #EFMBIFS. A segment of source module #EFMBIFS is shown below. Invocation names for the CA ADS supplied built-in functions can be changed by modifying the source module #EFMBIFS, as described under Changing Invocation Names.

Segment of Source Module #EFMBIFS

```

*      #EFMBIFS  EVAL  BUILT-IN  FUNCTIONS  -  MASTER  DEFS
*****  FUNCTION  =  LENGTH  (STRING  FUNCTION)  *****
      SPACE 2
****  INVOCATION  NAME  =  SLENGTH  ****
      #EFUNMST  TYPE=ENTRY,                                X
          INVOKE=SLENGTH,                                  X
          FUNCT=LENGTH,                                    X
          PROGRAM=RHDCEV51
      SPACE 2
****  INVOCATION  NAME  =  STRING-LENGTH  ****
      #EFUNMST  TYPE=ENTRY,                                X
          INVOKE=STRING-LENGTH,                            X
          FUNCT=*,                                          X
          PROGRAM=*
      SPACE 2
****  INVOCATION  NAME  =  SLEN  ****
      #EFUNMST  TYPE=ENTRY,                                X
          INVOKE=SLEN,                                     X
          FUNCT=*,                                          X
          PROGRAM=*
      EJECT
*****  FUNCTION  =  SUBSTRING  (STRING  FUNCTION)  *****
      SPACE 2
****  INVOCATION  NAME  =  SUBSTRING  ****
      #EFUNMST  TYPE=ENTRY,                                X
          INVOKE=SUBSTRING,                                X
          FUNCT=SUBSTRNG,                                  X
          PROGRAM=RHDCEV51

```

```

SPACE 2
**** INVOCATION NAME = SUB-STRING ****
      #EFUNMST TYPE=ENTRY,           X
      INVOKE=SUBSTR,                 X
      FUNCT=*,                       X
      PROGRAM=*

SPACE 2
**** INVOCATION NAME = SUBS ****
      #EFUNMST TYPE=ENTRY,           X
      INVOKE=SUBS,                   X
      FUNCT=*,                       X
      PROGRAM=*

EJECT

***** FUNCTION = INDEX (STRING FUNCTION) *****

SPACE 2
**** INVOCATION NAME = INDEX ****
      #EFUNMST TYPE=ENTRY,           X
      INVOKE=INDEX,                  X
      FUNCT=INDEX,                   X
      PROGRAM=RHDCEV51

SPACE 2
**** INVOCATION NAME = STRING-INDEX ****
      #EFUNMST TYPE=ENTRY,           X
      INVOKE=STRING-INDEX,           X
      FUNCT=*,                       X
      PROGRAM=*

SPACE 2
**** INVOCATION NAME = INDX ****
      #EFUNMST TYPE=ENTRY,           X
      INVOKE=INDX,                   X
      FUNCT=*,                       X
      PROGRAM=*

```

More information:

[Changing Invocation Names](#) (see page 713)

#EFUNMOD

Purpose

Defines a model XDE module and the model XDE tables within the module.

Each model XDE table describes one function. During process compilation of a built-in function, the dialog compiler uses the appropriate model XDE table to convert the built-in function into a series of XDEs, which represents the function at runtime.

Syntax

```

▶▶ #EFUNMOD TYPE = [ INITIAL, NAME = model-xde-module-name ]
                    [ HDR, hdr-options ]
                    [ XDE, DECS = decimal-options ]
                    [ DATA, CONV = ( conv-options ) ]
                    [ FINAL ]

```

Expansion of **hdr-options**

```

▶▶ FUNCNAM = function-name ,
▶▶ PROGRAM = processing-program-name ,
▶▶ FUNCNBR = function-number ,
▶▶ WORKLNG = work-area-length ,
▶▶ FIXOPND = fixed-operands-count ,
▶▶ VAROPND = [ YES ] , [ NO ]
▶▶ RESLNG = [ CALC ] , [ OPND ]
              [ result-length ]
▶▶ RESDATP = [ OPND ] ,
              [ data-type ]
▶▶ RESDEC = [ OPND ]
              [ result-decimal-places ]

```

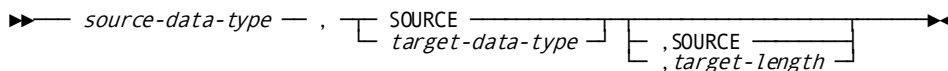
Expansion of **decimal-options**

```

▶▶ [ SOURCE ] , [ OPT = [ YES ] , [ NO ] ]
   [ decimal-places ]
▶▶ [ ROUND = [ YES ] , [ NO ] ] , [ RESLCAL = [ ADD ] , [ SUBT ] ]
▶▶ [ RESDEFL = [ YES ] , [ NO ] ]

```

Expansion of source-specification



Expansion of hdr-options

Expansion of decimal-options

Expansion of source-specification

Parameters

INITIAL, NAME = *model-xde-module-name*

Specifies the 1- to 8-character model XDE module name.

The TYPE=INITIAL macro is coded first and only once in the assembler program.

HDR, *hdr-options*

Defines the beginning of a model XDE table and specifies function XDE information.

One TYPE=HDR macro is coded for each model XDE table.

See expansion of *hdr-options* below.

XDE, DECS = *decimal-options*

Specifies operand XDE information that describes a target parameter.

The TYPE=XDE macro describes a function parameter. One TYPE=XDE macro is coded for each parameter in the order that the parameter is to appear in the parameter list.

DECS = *decimal-options* is used to specify the number of decimal places in the target parameter being described.

See the expansion of *decimal-options* below.

DATA, CONV = (*conv-options*)

Specifies the data type and length of the target parameter (that is, the parameter as it is stored in the IRA for use by the processing program), based on the data type of the source parameter (that is, the parameter as it is coded in the parameter list).

See expansion of *conv-options* below.

At least one TYPE=DATA macro must be coded following a TYPE=XDE macro. If two or more are specified, the dialog compiler uses the TYPE=DATA macro whose *source-data-type* specification matches the data type of the source parameter. If no *source-data-type* specification matches, the last TYPE=DATA macro is used.

Note: During process compilation, any combination of source and target parameter data types is accepted. At runtime, the runtime system attempts to make any required data type conversions; if it cannot, the dialog aborts.

FINAL

Defines the end of the model XDE module.

Expansion of *hdr-options*

FUNCNAM= *function-name*

User-defined parameter specifying the 1- to 8-character real (generic) function name.

The real function name associates a master function table entry with the model XDE table.

PROGRAM= *processing-program-name*

User-defined parameters specifying the 1- to 8-character name of the processing program module that contains the processing program for the function.

FUNCNBR= *function-number*

User-supplied numeric literal specifying a number from 0 to 255 that uniquely identifies the associated processing program within the processing program module.

WORKLNG= *work-area-length*

User-supplied numeric literal specifying the number of bytes of work area required by the processing program module for the function.

The WORKLNG specification should not include work space required by the runtime system, which is automatically added by the macro.

Note: *Work-area-length* must be at least 80.

FIXOPND= *fixed-operands-count*

User-supplied numeric literal specifying the number of fixed parameters for the function.

A fixed parameter is a parameter that can be specified only once in a parameter list. A function can have from 0 to 50 fixed parameters.

VAROPND=YES/NO

Specifies whether one parameter in the parameter list is variable.

A variable parameter can be specified repeatedly in a parameter list. (An example of a variable parameter is *'string'* or *string-variable* in the concatenate function.)

A function can have only one variable parameter and it must follow all fixed parameters.

The default VAROPND specification is NO.

RESLNG=

Clause introducing the length, in bytes, of the function's result field,

CALC

Specifies that the result field length is calculated from the lengths of the function parameters, based on the RESLNG specification of each parameter's TYPE=XDE macro.

If CALC is specified, the result field length is calculated as the sum of the lengths of the function parameters whose RESLNG specification is ADD, minus the sum of the lengths of the function parameters whose RESLNG specification is SUBT. Parameters without a RESLNG specification are not included in the calculation.

OPND

Specifies that the result length is equal to the length of the function parameter whose RESDEFL specification is YES.

result-length

Specifies a result length, in bytes, from 1 to 32767.

RESDATP=

Clause introducing the data type of the function's result field of the function.

OPND

Specifies that the result data type is the same as the data type of the function parameter whose RESDEFL specification is YES.

data-type

User-defined parameter specifying the result field data type.

Data-type is one of the three-character data type abbreviations shown in the table under **Usage** below.

RESDEC=

Clause introducing the number of decimal places in the function's result field.

OPND

Specifies that the number of decimal places is equal to the number of decimal places in the function parameter whose RESDEFL specification is YES.

result-decimal-places

Specifies the number of result decimal places, from 0 to 32.

Expansion of decimal-options

SOURCE

Specifies that the number of decimal places equals the number of decimal places in the source parameter.

decimal-places

Specifies the number of decimal places, from 0 to 32.

OPT=YES/NO

Specifies whether the parameter is optional and can be omitted from the coded parameter list.

NO is the default when neither YES or NO is specified.

ROUND=YES/NO

Specifies whether rounding or truncation is used when converting from the source parameter to the target parameter. NO indicates truncation.

YES is the default when neither YES or NO is specified.

RESLICAL=

Clause introducing the action to be taken to the parameters length in the calculation of the length of the result field.

ADD

Specifies that the parameter's length is added in the calculation of the length of the result field.

SUBT

Specifies that the parameter's length is subtracted in the calculation of the length of the result field.

The RESLICAL specification should be included only if the RESLNG specification of the preceding TYPE=HDR macro is CALC. If the RESLICAL specification is omitted, the parameter's length is not considered in the calculation of the length of the result field.

RESDEFL=YES/NO

Specifies whether the parameter is used to determine result field characteristics that are specified in the associated TYPE=HDR macro as OPND.

Only one TYPE=XDE macro for a function can specify RESDEFL=YES.

The default RESDEFL specification is NO.

Expansion of conv-options***source-data-type***

User-defined parameters specifying the three-character abbreviation of the data type of the source parameter; these abbreviations are listed in the table under Usage below.

During process compilation, if the data type of the source parameter is *source-data-type*, then the target parameter is assigned a data type of SOURCE/*target-data-type* and a length of SOURCE/*target-length*. The target parameter's data type and length are stored in the parameter's operand XDE.

SOURCE

Specifies that the data type of the target parameter is the same as the data type of the source parameter.

target-data-type

User-defined parameters specifying the three-character abbreviation of the data type of the target parameter; these abbreviations are listed in the table under Usage below.

SOURCE

Specifies that the length of the target parameters is the same as the length of the source parameter.

target-length

Specifies the length of the target parameter in bytes.

If neither is specified, a length is generated based on the data type of the target parameter, if possible.

Usage

Considerations

- The source assembler program must begin with one TYPE=INITIAL macro and end with one TYPE=FINAL macro.
- One TYPE=HDR macro is coded for each function that is described in the module.
- One TYPE=XDE macro is coded for each parameter of each function; the macro applies to the function described by the preceding TYPE=HDR macro and is coded in the order that the parameter is to appear in the parameter list.
- One or more TYPE=DATA macros are coded for each data type conversion specification for each parameter; the macro applies to the parameter described by the preceding TYPE=XDE macro.

Data Type Abbreviations

Data type	Abbreviation
Display floating point	DFL
Doubleword binary	DWB
EBCDIC	EBD
Fullword binary	FWB
Group	GRP
Halfword binary	HWB

Data type	Abbreviation
Long floating point	LFL
Multibit binary	MBB
Short floating point	SFL
Signed packed decimal	SPK
Signed zoned decimal	SZN
Unsigned packed decimal	UPK
Unsigned zoned decimal	UZN
Varying character	VCH

Note: Only target parameters and the result field can have the varying character data type. A varying character field consists of a halfword binary field that specifies the length of the varying character string, followed by a fixed field that contains the string itself.

Model XDE Modules

The model XDE modules for the CA ADS supplied built-in functions are defined by the source assembler programs called RHDCEV51, RHDCEV52, and RHDCEV53. Segments of RHDCEV51 are shown below. An installation should not change these modules, but can reference them as guides for creating user-defined built-in functions.

Segments of Source Assembler Program RHDCEV51

```

RHDCEV51 TITLE 'EVAL - BUILT-IN STRING FUNCTIONS - MODEL XDE TBL'
* RHDCEV51 EP=RHDCEV51                                06/29/90 14:05:40
      SPACE 3
RHDCEV51 AMODE ANY
RHDCEV51 RMODE 24
      #EFUNMOD TYPE=INITIAL,NAME=RHDCEV51
      EJECT
*****
*      FUNCTION = LENGTH                                *
*****
      SPACE 3
LENGTH #EFUNMOD TYPE=HDR,                                X
      FUNCNAM=LENGTH,                                  X
      FUNCNBR=0,                                       X
      PROGRAM=RHDCEV01,                                X
      WORKLNG=148,                                     X
      FIXOPND=1,                                       X
      RESLNG=2,                                        X
      RESDATP=HWB,                                     X
      RESDEC=0
      SPACE 1
      #EFUNMOD TYPE=XDE,DECS=0,OPT=NO
      SPACE 1
      #EFUNMOD TYPE=DATA,CONV=(EBD,VCH,SOURCE)
      EJECT

*****
*      FUNCTION = SUBSTRING                            *
*****
      SPACE 3
SUBSTRNG #EFUNMOD TYPE=HDR,                                X
      FUNCNAM=SUBSTRNG,                                 X
      FUNCNBR=1,                                       X
      PROGRAM=RHDCEV01,                                X
      WORKLNG=148,                                     X
      FIXOPND=3,                                       X
      RESLNG=CALC,                                    X
      RESDATP=VCH,                                     X
      RESDEC=0

```



```

SPACE 1
#EFUNMOD TYPE=XDE, DECS=0, OPT=NO, RESL CAL=ADD
SPACE 1
#EFUNMOD TYPE=DATA, CONV=(EBD, VCH, SOURCE)
SPACE 1
#EFUNMOD TYPE=XDE, DECS=0, OPT=NO
SPACE 1
#EFUNMOD TYPE=DATA, CONV=(EBD, HWB, 2)
SPACE 1
#EFUNMOD TYPE=XDE, DECS=0, OPT=YES
SPACE 1
#EFUNMOD TYPE=DATA, CONV=(EBD, HWB, 2)
EJECT
...
#EFUNMOD TYPE=FINAL
SPACE 2
END RHDCEV51 *CRM84199*
```

Changing Invocation Names

An installation can add, modify, or delete any invocation name for any CA ADS supplied or user-defined built-in function. The following steps update the master function table, which contains the valid invocation names:

1. **Modify source macro #EFMBIFS**— #EFMBIFS contains the assembler macros that define the entries in the master function table for the CA ADS supplied built-in functions. Invocation names can be changed by adding, modifying, and/or deleting the appropriate #EFUNMST macros in #EFMBIFS, then following the steps listed below. Refer to the syntax rules for the #EFUNMST macro earlier in this appendix.
2. **Modify source module RHDCEVBF**— RHDCEVBF contains the #EFUNMST assembler macros that define the master function table, including the TYPE=INITIAL macro, which automatically generates the macros stored in #EFMBIFS.

Invocation names for user-defined functions are defined by TYPE=ENTRY macros coded between the TYPE=INITIAL and TYPE=FINAL macros in RHDCEVBF. TYPE=ENTRY macros can be added, modified, and deleted as required.

RHDCEVBF also contains PUNCH statements that prefix the module with the required IDD statement to place the master function table in the data dictionary load area. Change the action ADD to MOD if it has not already been changed.

3. **Assemble source module RHDCEVBF**— The object module generated should also be called RHDCEVBF.
4. **Place RHDCEVBF in the data dictionary load area**— Use the DDDL compiler.

Note: For more information on JCL for the DDDL compiler, see the *CA IDMS IDD Quick Reference Guide*.

Creating User-Defined Built-In Functions

Built-in functions can be created to meet site-specific needs. User-defined built-in functions are coded like the CA ADS supplied functions.

The following topics are discussed below:

- Steps for generating a user-defined built-in function
- LRF Considerations
- Calling a user-defined built-in function

Steps for Generating a User-Defined Built-In Function

An installation can generate a user-defined function by following the instructions listed below in any order:

- **Create a processing program module**, as follows:
 1. **Create the source module**— As a guide, refer to the source module RHDCEV01 which contains some of the source code for the CA ADS supplied built-in functions.

Processing logic for several functions can be included in one processing program module, thereby reducing the number of modules that must be loaded at runtime. Each function is distinguished by a unique function number. The function number is defined in the model XDE module by the FUNCNBR parameter of the #EFUNMOD TYPE=HDR macro. At runtime, the function number is contained in the XDEUFUNC field of the function XDE, and can be used by the processing program module to branch to the appropriate processing program.

Note: A built-in function that supports entry of optional parameters must be able to determine at execution time whether the optional parameters have been entered. To do this, the built-in function must perform a runtime check of the XDE/VXDE for each such parameter.

If an optional parameter is omitted, the runtime system passes a dummy operand VDE/VXDE to the built-in function for the omitted parameter. The dummy XDE/VXDE for the parameter has the following characteristics, for which a built-in function can test:

- The XDEFNVL bit in the XDEFLAG field is set to 1.
- The XDEDATAD field is set to X'80000000'.

2. **Assemble the processing program source module.**
3. **Link edit the module into the load library.**
4. **Add the program at system generation with the PROGRAM statement.**

- **Create a model XDE module**, as follows:
 1. **Create the source module**— The source module consists of #EFUNMOD macros. As a guide, refer to RHDCEV51, RHDCEV52, RHDCEV53, RHDCEV59, and RHDCEV60, the model XDE source modules for the CA ADS supplied built-in functions.
 2. **Assemble the source module.**
 3. **Link edit the module into the load library.**
- **Update the master function table** by following the steps described under [Changing Invocation Names](#) (see page 713).

More information:

[CA ADS Runtime System](#) (see page 119)

LRF Considerations for User-Defined Built-In Functions

If a site-defined built-in function is used with the Logical Record Facility (LRF) WHERE clause, the function must check each parameter to determine if the record containing the parameter value has been read by LRF processing. If the value has been read, the parameter is considered *valued*. If the value has not yet been read, the parameter is *nonvalued*.

A parameter is checked for being nonvalued by examining its associated XDE and VXDE. The exact checks that need to be made depend on whether a parameter is optional or required for that particular built-in function. The following considerations apply:

- **If a parameter is optional**, it is nonvalued if VXDEFNVL is ON, or if XDEFNVL is ON and XDEDATAD is not equal to X'80000000'.
- **If a parameter is required**, it is nonvalued if either of its XDEFNVL or VXDEFNVL bits is ON.

If any parameter is nonvalued, the built-in function must react accordingly. The proper action to take depends on the function being performed and which parameter is nonvalued. In most cases, the built-in function will return a nonvalued result by setting the VXDEFNVL flag in the result VXDE.

Calling a User-Defined Built-In Function

Purpose

This is the generalized syntax for calling a user-defined built-in function.

Syntax

►► *invocation-name* (*parameter*)

Parameters

invocation-name

Specifies the invocation name for the user-defined function.

parameter

Specifies the parameters for the user-defined function.

Optional parameters that are not included must be replaced by the character @, unless no included parameters follow the omitted parameter.

Appendix G: Security Features

This section contains the following topics:

[Overview](#) (see page 717)

[CA ADS Compiler Security](#) (see page 718)

[CA ADS Application Security](#) (see page 719)

Overview

In the CA IDMS environment, use of the CA ADS compilers and use of the CA ADS applications that you develop with the compilers can be secured.

Compiler Security

Use of the CA ADS compilers can be secured through the CA IDMS central security system at various levels, such as at the task level and at the program level. A dictionary that the compiler accesses can be secured as a database.

Note: For more information about CA IDMS central security, see the *CA IDMS Security Administration Guide*.

Use of the CA ADS compilers to access particular dictionary entities can also be controlled. You can secure access to dictionaries that the CA ADS compilers access using DDDL statements.

Application Security

Use of CA ADS applications can be secured through CA IDMS central security at various levels, such as at the task level, the program level, and the activity level. Databases that the application accesses, including dictionaries, can be secured.

CA ADS security classes are used when activities are secured in CA IDMS. You can also specify a security class for the CA ADS application and security classes for application responses. At runtime, the application issues a request for a security check when a user tries to execute an application or an application response for which you have specified a security class. If activity security has been enabled, CA IDMS central security checks to see whether the user has authority to execute the activity whose activity number matches the security class.

Note: For more information about CA IDMS activity security, see the *CA IDMS Security Administration Guide*.

When you define an CA ADS application, you can specify that the user must sign on to the application in order to execute it.

More information:

[Response Security](#) (see page 719)

[Signon Security](#) (see page 720)

[CA ADS Compiler Security](#) (see page 718)

CA ADS Compiler Security

Compiler security for the Application Compiler (ADSA) and the Dialog Compiler (ADSC) prohibits unauthorized users from adding, modifying, displaying, or deleting applications and dialogs. The compilers perform a security check whenever a user begins a compiler session (that is, when a user specifies the name of a application/dialog to add, modify display, or delete). If the security check fails, the user cannot perform the specified action.

Security is established by using the Integrated Data Dictionary (IDD) when the following is true:

- IDD SECURITY is ON in the dictionary
- You are assigned the IDD authority through the AUTHORITY clause of the DDDL USER statement

Note: For more information on IDD security and the DDDL USER statement, see the *CA IDMS IDD Quick Reference Guide*.

DDDL Statements Governing Compiler Security

Security at the compiler level restricts the actions that a user can specify for any application and dialog. Security at the compiler level is governed by the following two DDDL statements:

- **SET OPTIONS ... SECURITY FOR ADS IS ON/OFF**— Specifies whether compiler level security is in effect. If security for CA ADS is off, the user passes the compiler level security check. If security is on and the user has not signed on to DC/UCF, the user immediately fails the security check. Otherwise, the user passes or fails the security check based on the USER statement discussed below.
- **ADD/MOD USER user-name ... INCLUDE/EXCLUDE AUTHORITY FOR UPDATE/ADD/MODIFY/REPLACE/DELETE/DISPLAY IS ADS**— Specifies the actions that the user has the authority to perform using the application or dialog compiler. The user passes or fails the security check depending on whether the user has specified an authorized action.

Note: For more information on the SET OPTIONS and USER statements, see the *CA IDMS IDD Quick Reference Guide*.

If the user fails the application or dialog compiler level security check, the compiler displays an error message. If the user passes the security check, the compiler performs a security check at the application/dialog-specific level.

CA ADS Application Security

There are two ways in which CA ADS applications can be protected from unauthorized use.

Outside the Application

Protection can be provided at the application level through the CA IDMS central security system. For example, a user's authority to execute programs and dictionary load modules can be controlled through the CA IDMS central security facility.

Within the Application

In addition to defining security outside the application, the CA ADS application compiler provides two security features that allows you to define security within the application:

- Security for responses
- Signon security

Security for responses is based on application activity security controlled through the CA IDMS central security facility. Within the security facility, application activities can be defined as secured resources and authority to execute those activities granted to one or more users.

Response security, and signon security are discussed separately below.

Response Security

Response security enables you to define security for individual application functions. To implement response security, you enter a number in the **Security class** field of the ADSA Response Definition screen. When the application is compiled, the application load module includes the activity number of each response.

At runtime, response security is enforced if the security administrator has secured activities and has defined activities that correspond to the application functions for which response security is defined. When the application issues a security check on a response, CA IDMS central security looks for an activity definition in which the application name matches the CA ADS application name and the activity number matches the CA ADS response security class.

CA ADS makes no calls to CA IDMS central security for security class 0, which is defined always as unsecured.

Note: For more information on defining and controlling application activities, see the *CA IDMS Security Administration Guide*.

If a user without execute authority for the corresponding activity, attempts to execute a secured response, the runtime system redisplay the screen from which the response was selected, along with the following message:

UNACCEPTABLE RESPONSE. PLEASE TRY AGAIN

Because the response is secured, the function invoked by the response cannot be accessed unless the security administrator has authorized the appropriate users to execute the corresponding application activity defined to CA IDMS central security.

Response security is complemented by the CA ADS security-tailored menus feature. At runtime, security-tailored menus list only those responses that the user has authority to select. Menus are security-tailored by selecting option 2, **Security tailored**, on the second page of the ADSA General Options screen.

Signon Security

Signon security can be implemented for any application defined using the application compiler. With signon security, a user begins executing an application by entering a user ID and password, which the runtime system validates. To implement signon security for an application, follow the steps listed below:

1. **Specify SIGNON IS OPTIONAL or SIGNON IS REQUIRED on the second page of the ADSA General Options screen.** If signon is optional, the user can sign on before executing the application, but is not required to. If signon is required, the user must enter a valid user ID and password before executing the application.
2. **Specify the name of the signon menu function on the second page of the ADSA General Options screen.** The signon menu function is executed first when the user begins executing the application. The function displays a signon menu screen, which provides fields in which to enter a user ID and password.
3. **Define an immediate response that invokes the SIGNON system function on the Response Definition screen.** When invoked at runtime, the SIGNON function validates the user ID and password entered by the user, then returns control to the signon menu function.

4. **Define the signon menu function on the Function Definition screen and any appropriate secondary screen**, as follows:
 - On the Function Definition screen, define the function as a menu function and specify the function name supplied on the Security screen. Optionally, specify that the response that invokes the SIGNON system function is the default response for the signon menu function; if this response is the default, the user need only press the [Enter] from the function at runtime to invoke the SIGNON function.
 - On the Menu Function Definition screen, specify that the menu function is a signon menu function by entering a slash (/) in the **Use signon menu** field.
 - On the Valid Responses screen, specify that the response that invokes the SIGNON system function is a valid response for the signon menu function.

Runtime Processing

At runtime, processing is performed as follows:

1. When the application begins execution, the runtime system displays the signon menu function. If signon is optional, all valid responses for the function are displayed. If signon is required and menus are security tailored, only authorized responses are displayed.
2. On the signon menu screen, the user signs on by entering a user ID and password in the appropriate fields, then selecting the response that invokes the SIGNON system function. If signon is optional, the user can instead begin executing the application immediately.
3. The SIGNON system function validates the signon, then redisplay the signon menu screen with one of the following messages:
 - SIGNON ACCEPTED
 - SIGNON FAILED; UNKNOWN USER ID
 - SIGNON FAILED; INVALID PASSWORD
4. If the signon is accepted, all valid responses for the signon menu function are displayed; the user can execute the application. If the signon fails, the user can attempt to sign on again.

The signon menu function may be different than the function invoked by the initiating application task code. In such cases, the application begins by executing the signon menu function. The function associated with the application task code is executed when the user presses [Enter] from the redisplayed signon menu screen after signing on successfully. If signon is optional, the user can press [Enter] without signing on.

The SIGNOFF system function can be used in conjunction with signon security. When selected at runtime, the SIGNOFF function signs the user off the application, then redisplay the screen from which the function was selected. If signon is required, the next user must sign on successfully before executing the application.

The SIGNOFF Function

To implement the SIGNOFF system function, perform the following steps using the application compiler:

1. Define a response that invokes the SIGNOFF system function on the Response Definition screen.
2. Make the response a valid response for the signon menu function on the Valid Responses screen.
3. Define the application structure so that the user, at runtime, can return to the signon menu function to sign off.

At runtime, when the SIGNOFF system function is invoked, the runtime system signs the user off the application, then redisplay the screen with the following message:

SIGNOFF ACCEPTED

If signon is required, the runtime system additionally blanks out all responses listed on the screen.

More information:

[System-Defined Menu Maps](#) (see page 126)

Appendix H: Debugging an CA ADS Dialog

This section contains the following topics:

[Overview](#) (see page 723)

[Creating a Symbol Table](#) (see page 723)

[Trace Facility](#) (see page 725)

[Online Debugger](#) (see page 727)

Overview

About this Appendix

To debug dialogs, you can use the CA ADS trace facility and the CA IDMS online debugger. This appendix explains the use of both facilities.

Creating a Symbol Table

Prerequisite for Debugging

To use either the trace facility or the online debugger to debug a dialog, you must first compile the dialog with a symbol table. A symbol table contains information such as data field names and process command line numbers that enable the trace facility and the online debugger to execute.

How to Create a Symbol Table

When defining the dialog using the dialog compiler, invoke the Options and Directives screen and enter a nonblank character next to the **Symbol table is enabled** prompt, as shown below:

```
Options and Directives
Dialog JPKTD10 Version 1
Type and select each option and directive. Then Enter.
Message prefix . . . . . DC
Autostatus record . . . . . ADS0-STAT-DEF-REC
Version . . . . . 1
Options and directives . . . . .
_ Mainline dialog
x 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
```

In ADSOBCOM, use the symbol table option of the DIALOG expression.

Compile the dialog. When the dialog successfully compiles, a load module with interpretable CMEs is created.

More information:

[CA ADS Dialog Compiler \(ADSC\)](#) (see page 91)

[Application and Dialog Utilities](#) (see page 621)

Trace Facility

The CA ADS trace facility is a debugging aid used to trace the flow of control and commands executed in an CA ADS application at runtime.

The CA ADS trace facility writes trace records to the DC/UCF system log as DEBUG records. Trace records can be viewed by using online PLOG or the batch print-log utility (PRINT LOG). Use the MESSAGES parameter to print the CA ADS trace records.

Note: The PRINT LOG TRACES parameter will not print the CA ADS trace records. For more information about online PLOG, see the *CA IDMS System Tasks and Operator Commands Guide*. For more information about the batch print-log utility, see the *CA IDMS Utilities Guide*.

Information in a Trace Facility Report

The table below describes the information contained in a trace facility report.

Field	Contents
Dialog name	The name of the currently executing dialog.
Process name	The name of the currently executing premap or response process. ***** DIALOG-ENTRY ***** in this field documents the beginning of a dialog.
Subroutine name	The name of the process subroutine currently executing. **MAIN** in this field documents the beginning of a dialog, process, or process command that is not in a subroutine.
Sequence number	The IDD sequence number of the command currently executing. 00000000 appears in this field at the beginning of a dialog, process, or subroutine.
Process command	The process command currently executing. ENTRY in this field documents the beginning of a dialog, process, or subroutine.
Command offset	The hexadecimal offset of the command from the beginning of the dialog's fixed dialog block (FDB).
Included module name	The name of each included module.

Format of a Trace Facility Record for a Non-SQL DML Statement

The format of a trace facility record for a non-SQL DML statement is shown below.

Blank	Dialog Name	Process Name	Sub-routine Name	Sequence Number	Process Command	Command Offset	Included Module Name
0	8	17	50	61	70	79	86
8							11

Format of a Trace Facility Record for an SQL DML Statement

The trace facility report for an SQL statement follows the format below:

1. **SQL CMT** = followed by the SQL command (for example, SELECT).
2. **CODE** = followed by the appropriate SQL code
3. **ERROR** = followed by the 5-digit error.

Below this information is the database message passed from the SQLCA.

Note: For more information about the SQLCA, see the *CA IDMS SQL Programming Guide*.

Initiating the Trace Facility

You can initiate the trace facility in one of these ways:

- Use the TRACE keyword in the runtime system initiating statement when requesting execution of the application
- Coding the TRACE command in the dialog process

Specifying TRACE when initiating CAADS results in tracing the execution of all dialogs in the application that have been compiled with a symbol table. You use TRACE and TRACE OFF in process logic to limit the trace.

The System Log

Using the CA ADS trace facility can fill the DC/UCF system log quickly. Information on the entire application is collected for each process command in dialogs that have a symbol table enabled and TRACE=ALL specified. A record is written to the system log for each command.

To avoid overloading the system log, the system log can be defined to sequential log files instead of the DDLDCLOG area. Assigning the system log to sequential log files facilitates offloading the system log when it becomes full.

Note: For more information about log files, see the *CA IDMS System Generation Guide*.

More information:

[Initiating the CA ADS Runtime System](#) (see page 119)

[TRACE](#) (see page 518)

Online Debugger

What You Can Do

The online debugger enables the application developer to interrupt execution of a dialog's premap or response process, display and change the contents of data fields, and restart execution from any point in the interrupted dialog. If a dialog aborts during a debugger session, the application developer can review the contents of the data fields at the time of the abend, change the contents of the data fields, and resume dialog execution at any point. For example, a breakpoint can be set at line 200 in a premap process and can specify that an interruption is to occur every second time the process command is executed at runtime.

If CA ADS encounters a potential breakpoint at runtime, it passes control to the online debugger. If the conditions for interrupting the dialog are not met, control returns to the runtime system and execution continues. If the conditions are met, the online debugger keeps control and allows the application developer to perform functions such as reviewing and modifying data fields, modifying breakpoint specifications, aborting dialog execution, and resuming execution at a specified point.

If a dialog aborts during a debugger session, the CA ADS runtime system displays a special version of the Dialog Abort Information screen and then links to the online debugger.

The special screen version allows the application developer to continue the debugging session for the dialog. The application developer can enter debugger commands at the prompt that appears on the screen.

Procedures

Procedures for debugging an CA ADS dialog are the same as those used with any other program running under DC/UCF: once the dialog is defined to the debugger with the DEBUG command, debugging procedures can take place. It is easier, however, to find dialog records or to find the command elements (CMEs) for breakpoints when the load module is generated in interpretable code and symbol recognition is in effect.

Note: For more information about debugging, see the *CA IDMS Online Debugger Guide*.

Recommended Steps

It is recommended that you take the following steps when debugging an CA ADS dialog:

1. Create a symbol table for the dialog
2. Compile the dialog
3. Run ADSORPTS for the dialog
4. Issue the DEBUG task code to invoke the debugger
5. Define the dialog to the debugger
6. Set breakpoints (as required)
7. Issue the EXIT command to leave the debugger
8. Invoke the CA ADS runtime system and execute the dialog
9. Continue processing the dialog
10. Issue the EXIT or QUIT command when debugging is completed

Run ADSORPTS for the Dialog

Run the CA ADS Dialog Report (ADSORPTS) for the given dialog. Specify the PROCESS and/or FDBLIST options when submitting the report: PROCESS displays the sequence line numbers that are assigned to the process source; FDBLIST provides the line numbers (SEQ#) and the offsets of the CMEs. The address or line number of a CME can then be used to set a valid breakpoint within the premap or response process.

Issue the DEBUG Task Code

Issue the DEBUG task code to invoke the debugger.

Note: For more information on initiating a debugger session, see the *CA IDMS Online Debugger Guide*.

Define the Dialog to the Debugger

Define the dialog to the debugger by issuing a command similar to the one in the following example:

```
DEBUG>  
debug dialog medduins
```

The debugger responds to the above command with the following message:

```
DEBUG DIALOG MEDDUINS  
DEBUG> DEBUGGING INITIATED FOR MEDDUINS VERSION 1  
DEBUG>
```


If you omit the word *dialog*, the debugger issues an error message:

```
DEBUG>
debug medduins
DEBUG MEDDUINS
DEBUG> INCONSISTENT ENTITY TYPE
        - MEDDUINS VERSION 1 DEFINED AS A DIALOG
DEBUG>
```

This message indicates that the debugger has tried to process MEDDUINS as a program but can only find a PDE (program descriptor element) that defines MEDDUINS as a dialog. The command needs to be modified to state that MEDDUINS is a dialog:

```
DEBUG>
debug dialog medduins
```

Set Breakpoints

Set breakpoints as required. Breakpoints must be set at line numbers or addresses that contain valid command instructions (valid CMEs).

Note: For more information about setting breakpoints, see the *CA IDMS Online Debugger Guide*.

Issue EXIT

Issue the EXIT command and leave the debugger.

Invoke the CA ADS Runtime System

Invoke the CA ADS runtime system and execute the dialog in the standard manner.

When a breakpoint is encountered during the execution of the dialog, a message appears on the screen that identifies the breakpoint. The DEBUG> prompt or menu mode screen is displayed, signalling that you are now in the runtime phase of the debugger and can enter any of the debugger commands except DEBUG.

Note: For more information on command syntax, see the *CA IDMS Online Debugger Guide*.

Continue Processing the Dialog

Continue processing the dialog. When the single command RESUME is issued without any qualifying parameters, processing continues from the current CME (the instruction immediately following the breakpoint). When a RESUME *debug-expression* is issued, processing resumes at the address specified by the expression.

When you issue a RESUME *dialog-expression* command from a point within the main body of the dialog process, the debug expression must resolve to an address also within the main body of the dialog. Similarly, when a RESUME *dialog-expression* is issued from a subroutine, the debug expression must resolve to an address within the same subroutine. Results are unpredictable when execution is not resumed in accordance with these rules.

In the Event of an Abend

In the event of an abend, you see the CA ADS Debug screen with dialog abort information, the DEBUG> prompt, and the menu mode selection area. Any valid debugger command can be entered on the prompt line or can be selected from the menu. When a selection is made from the menu, the debugger automatically operates in menu mode and displays the specified screen. A sample Debug screen is shown below. Note the DEBUG> prompt and menu selection area located at the bottom of the screen. All commands, except DEBUG, can be issued in response to the prompt or can be selected in the menu area:

```

CA-ADS RELEASE nn.n          *** DIALOG ABORT INFORMATION ***   DEBUG
DC175020 APPLICATION ABORTED. PGM CHECK (DATA EXCEPTION).

DATE.....: 91.220          TIME.....: 17:10:23.55          TERMINAL.....: LV81001

ERROR OCCURRED IN DIALOG.....: MISINCD
      AT OFFSET.....: 3D8
      IN PROCESS.....: MIS-MAIN1                          VERSION: 1
      AT IDD SEQ NO. : 000000100                          INTERNAL COMMAND: 2
      INCLUDED MODULE : MIS-INC1                          VERSION: 1

SEQUENCE
NUMBER:          SOURCE :
00000000
00000100      ADD 1 TO MIS-NUM.
00000200      ! THIS IS MIS-INC1

DEBUG>
NEXT _ ACTIVITY OR _ HELP:
  _ AT      _ LIST      _ SET      _ SNAP      _ RESUME      _ DEBUG      _ WHERE

      _ EXIT      _ PROMPT      _ QUIT      _ IOUSER
HELP _ SCREENS:  _ USAGE      _ SYMBOLS      _ KEYS

```

Issue the RESUME ABEND or a RESUME *debug-expression* command to continue processing the dialog.

QUIT or EXIT

Issue the QUIT or EXIT command when debugging is completed. QUIT clears the debugger control blocks and ends the debugger session; EXIT returns you to the ENTER NEXT TASK CODE prompt. but leaves the control blocks intact so that the debugger session can continue.

More information:

[Dialog Abort Information Screen](#) (see page 145)

Appendix I: Compiler Overview and Default Control Keys

This section contains the following topics:

[Summary of Application Compiler Process](#) (see page 733)

[Default Control Keys](#) (see page 734)

[Summary of Dialog Compiler Process](#) (see page 735)

[Default Control Keys](#) (see page 736)

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

Step in process	Screen	Purpose
	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

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

Activity	Control key	Description
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

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 and 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

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

Appendix J: Runtime Error-Status Codes

This section contains the following topics:

[Status Codes Returned by the Autostatus Facility](#) (see page 737)

[Major DB Status Codes](#) (see page 738)

[Minor DB Status Codes](#) (see page 738)

[Major DC Status Codes](#) (see page 743)

[Minor DC Status Codes](#) (see page 744)

[ERROR-STATUS Condition Names](#) (see page 748)

[Autostatus Return Codes](#) (see page 748)

[Default Level-88 Values](#) (see page 749)

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.

Major DB Status Codes

Major Code	Database Function
00	Any DML statement
01	FINISH
02	ERASE
03	FIND/OBTAIN
05	GET
06	KEEP
07	CONNECT
08	MODIFY
09	READY
11	DISCONNECT
12	STORE
14	BIND
15	ACCEPT
16	IF
17	RETURN
18	COMMIT
19	ROLLBACK
20	LRF requests

Minor DB Status Codes

Minor Code	Database Function Status
00	Combined with a major code of 00, this code indicates successful completion of the DML operation. Combined with a nonzero major code, this code indicates that the DML operation was not completed successfully due to central version causes, such as time-outs and program checks.
01	An area has not been readied. When this code is combined with a major code of 16, an IF operation has resulted in a valid false condition.

Minor Code	Database Function Status
02	Either the db-key used with a FIND/OBTAIN DB-KEY statement or the direct db-key suggested for a STORE is not within the page range for the specified record name.
03	Invalid currency for the named record, set, or area. This can only occur when a run unit is sharing a transaction with other database sessions. The 03 minor status is returned if the run unit tries to retrieve or update a record using a currency that has been invalidated because of changes made by another database session that is sharing the same transaction.
04	The occurrence count of a variably occurring element has been specified as either less than zero or greater than the maximum number of occurrences defined in the control element.
05	The specified DML function would have violated a duplicates -not-allowed option for a CALC, sorted, or index set.
06	No currency has been established for the named record, set, or area.
07	The end of a set, area, or index has been reached or the set is empty.
08	The specified record, set, procedure, or LR verb is not in the subschema or the specified record is not a member of the set.
09	The area has been readied with an incorrect usage mode.
10	An existing access restriction or subschema usage prohibits execution of the specified DML function. For LRF users, the subschema in use allows access to database records only. Combined with a major code of 00, this code means the program has attempted to access a database record, but the subschema in use allows access to logical records only.
11	The record cannot be stored in the specified area due to insufficient space.
12	There is no db-key for the record to be stored. This is a system internal error and should be reported.
13	A current record of run unit either has not been established or has been nullified by a previous ERASE statement.
14	The CONNECT statement cannot be executed because the requested record has been defined as a mandatory automatic member of the set.
15	The DISCONNECT statement cannot be executed because the requested record has been defined as a mandatory member of the set.
16	The record cannot be connected to a set of which it is already a member.
17	The transaction manager encountered an error.
18	The record has not been bound.
19	The run unit's transaction was forced to back out.

Minor Code	Database Function Status
20	The current record is not the same type as the specified record name.
21	Not all areas being used have been readied in the correct usage mode.
22	The record name specified is not currently a member of the set name specified.
23	The area name specified is either not in the subschema or not an extent area; or the record name specified has not been defined within the area name specified.
25	No currency has been established for the named set.
26	No duplicates exist for the named record or the record occurrences cannot be found.
28	The run unit has attempted to ready an area that has been readied previously.
29	The run unit has attempted to place a lock on a record that is locked already by another run unit. A deadlock results. Unless the run unit issued either a FIND/OBTAIN KEEP EXCLUSIVE or a KEEP EXCLUSIVE, the run unit is aborted.
30	An attempt has been made to erase the owner record of a nonempty set.
31	The retrieval statement format conflicts with the record's location mode.
32	An attempt to retrieve a CALC/DUPLICATE record was unsuccessful; the value of the CALC field in variable storage is not equal to the value of the CALC control element in the current record of run unit.
33	At least one set in which the record participates has not been included in the subschema.
40	The WHERE clause in an OBTAIN NEXT logical-record request is inconsistent with a previous OBTAIN FIRST or OBTAIN NEXT command for the same record. Previously specified criteria, such as reference to a key field, have been changed. A path status of LR-ERROR is returned to the LRC block.
41	The subschema contains no path that matches the WHERE clause in a logical-record request. A path status of LR-ERROR is returned to the LRC block.
42	An ON clause included in the path by the DBA specified return of the LR-ERROR path status to the LRC block; an error has occurred while processing the LRF request.

Minor Code	Database Function Status
43	A program check has been recognized during evaluation of a WHERE clause; the program check indicates that either a WHERE clause has specified comparison of a packed decimal field to an unpacked nonnumeric data field, or data in variable storage or a database record does not conform to its description. A path status of LR-ERROR is returned to the LRC block unless the DBA has included an ON clause to override this action in the path.
44	The WHERE clause in a logical-record request does not supply a key element (sort key, CALC key, or db-key) expected by the path. A path status of LR-ERROR is returned to the LRC block.
45	During evaluation of a WHERE clause, a program check has been recognized because a subscript value is neither greater than 0 nor less than its maximum allowed value plus 1. A path status of LR-ERROR is returned to the LRC block unless the DBA has included an ON clause to override this action in the path.
46	A program check has revealed an arithmetic exception (for example: overflow, underflow, significance, divide) during evaluation of a WHERE clause. A path status of LR-ERROR is returned to the LRC block unless the DBA has included an ON clause to override this action in the path.
53	The subschema definition of an indexed set does not match the indexed set's physical structure in the database.
54	Either the prefix length of an SR51 record is less than zero or the data length is less than or equal to zero.
55	An invalid length has been defined for a variable-length record.
56	An insufficient amount of memory to accommodate the CA IDMS compression/decompression routines is available.
57	A retrieval-only run unit has detected an inconsistency in an index that should cause an 1143 abend, but optional APAR bit 216 has been turned on.
58	An attempt was made to rollback updates in a local mode program. Updates made to an area during a local mode program's execution cannot be automatically rolled out. The area must be manually recovered.
60	A record occurrence type is inconsistent with the set named in the ERROR-SET field in the IDMS communications block. This code usually indicates a broken chain.
61	No record can be found for an internal db-key. This code usually indicates a broken chain.
62	A system-generated db-key points to a record occurrence, but no record with that db-key can be found. This code usually indicates a broken chain.

Minor Code	Database Function Status
63	The DBMS cannot interpret the DML function to be performed. When combined with a major code of 00, this code means invalid function parameters have been passed on the call to the DBMS. For LRF users, a WHERE clause includes a keyword that is longer than the 32 characters allowed.
64	The record cannot be found; the CALC control element has not been defined properly in the subschema.
65	The database page read was not the page requested.
66	The area specified is not available in the requested usage mode.
67	The subschema invoked does not match the subschema object tables.
68	The CICS interface was not started.
69	A BIND RUN-UNIT may not have been issued; the CV may be inactive or not accepting new run units; or the connection with the CV may have been broken due to time out or other factors. When combined with a major code of 00, this code means the program has been disconnected from the DBMS.
70	The database will not ready properly; a JCL error is the probable cause.
71	The page range or page group for the area being readied or the page requested cannot be found in the DMCL.
72	There is insufficient memory to dynamically load a subschema or database procedure.
73	A central version run unit will exceed the MAXERUS value specified at system generation.
74	The dynamic load of a module has failed. If operating under the central version, a subschema or database procedure module either was not found in the data dictionary or the load (core image) library or, if loaded, will exceed the number of subschema and database procedures provided for at system generation.
75	A read error has occurred.
76	A write error has occurred.
77	The run unit has not been bound or has been bound twice. When combined with a major code of 00, this code means either the program is no longer signed on to the subschema or the variable subschema tables have been overwritten.
78	An area wait deadlock has occurred.
79	The run unit has requested more db-key locks than are available to the system.

Minor Code	Database Function Status
80	The target node is either not active or has been disabled.
81	The converted subschema requires specified database name to be in the DBNAME table.
82	The subschema must be named in the DBNAME table.
83	An error has occurred in accessing native VSAM data sets.
87	The owner and member records for a set to be updated are not in the same page group or do not have the same db-key radix.
91	The subschema requires a DBNAME to do the bind run unit.
92	No subschema areas map to DMCL.
93	A subschema area symbolic was not found in DMCL.
94	The specified dbname is neither a dbname defined in the DBNAME table, nor a SEGMENT defined in the DMCL.
95	The specified subschema failed DBTABLE mapping using the specified dbname.

Note: For a complete description of DB runtime status codes, see the chapter "CA IDMS Status Codes" in the *Messages and Codes Guide*.

Major DC Status Codes

Major Code	Function
00	Any DML statement
30	TRANSFER CONTROL
31	WAIT/POST
32	GET STORAGE/FREE STORAGE
33	SET ABEND EXIT/ABEND CODE
34	LOAD/DELETE TABLE
35	GET TIME/SET TIMER
36	WRITE LOG
37	ATTACH/CHANGE PRIORITY
38	BIND/ACCEPT/END TRANSACTION STATISTICS

Major Code	Function
39	ENQUEUE/DEQUEUE
40	SNAP
43	PUT/GET/DELETE SCRATCH
44	PUT/GET/DELETE QUEUE
45	BASIC MODE TERMINAL MANAGEMENT
46	MAPPING MODE TERMINAL MANAGEMENT
47	LINE MODE TERMINAL MANAGEMENT
48	ACCEPT/WRITE PRINTER
49	SEND MESSAGE
50	COMMIT TASK/ROLLBACK TASK/FINISH TASK/WRITE JOURNAL
51	KEEP LONGTERM
58	SVC SEND/RECEIVE

Minor DC Status Codes

Minor Code	Function Status
00	Combined with a major code of 00, this code indicates either successful completion of the DML function or that all tested resources have been enqueued.
01	The requested operation cannot be performed immediately; waiting will cause a deadlock.
02	Either there is insufficient storage in the storage pool or the storage required for control blocks is unavailable.
03	The scratch area ID cannot be found.
04	Either the queue ID (header) cannot be found or a paging session was in progress when a second STARTPAGE command was received (that is, an implied ENDPAGE was processed before this STARTPAGE was executed successfully).
05	The specified scratch record ID or queue record cannot be found.
06	No resource control element (RCE) exists for the queue record; currency has not been established.

Minor Code	Function Status
07	Either an I/O error has occurred or the queue upper limit has been reached.
08	The requested resource is not available.
09	The requested resource is available.
10	New storage has been assigned.
11	A maximum task condition exists.
12	The named task code is invalid.
13	The named resource cannot be found.
14	The requested module is defined as nonconcurrent and is currently in use.
15	The named module has been overlaid and cannot be reloaded immediately.
16	The specified interval control element (ICE) address cannot be found.
17	The record has been replaced.
18	No printer terminals have been defined for the current DC system.
19	The return area is too small; data has been truncated.
20	An I/O, program-not-found, or potential-deadlock status condition exists.
21	The message destination is undefined, the long term ID cannot be found, or a KEEP LONGTERM request was issued by a nonterminal task.
22	A record already exists for the scratch area specified.
23	No storage or resource control element (RCE) could be allocated for the reply area.
24	The maximum number of outstanding replies has been exceeded.
25	An attention interrupt has been received.
26	There is a logical error in the output data stream.
27	A permanent I/O error has occurred.
28	The terminal dial-up line is disconnected.
29	An invalid parameter has been passed in the list set up by the DML processor.
30	The named function has not yet been implemented.
31	An invalid parameter has been passed; the TRB, LRB, or MRB contains an invalid field; or the request is invalid because of a possible logic error in the application program. In a DC-BATCH environment, a possible cause is that the record length specified by the command exceeds the maximum length based on the packet size.

Minor Code	Function Status
32	The derived length of the specified variable storage is negative or zero.
33	Either the named table or the named map cannot be found in the data dictionary load area.
34	The named variable-storage area must be an 01-level entry in the LINKAGE SECTION.
35	A GET STORAGE request is invalid because the LINKAGE SECTION variable has already been allocated.
36	The program either was not defined during system generation or is marked out-of-service.
37	A GET STORAGE operand is invalid because the specified variable storage area is in the WORKING-STORAGE SECTION instead of the LINKAGE SECTION.
38	Either no GET STORAGE operand was specified or the specified LINKAGE SECTION variable has not been allocated.
39	The terminal device being used is out of service.
40	NOIO has been specified but the datastream cannot be found.
41	An IF operation resulted in a valid true condition.
42	The named map does not support the terminal device in use.
43	A line I/O session has been cancelled by the terminal operator.
44	The referenced field does not participate in the specified map; a possible cause is an invalid subscript.
45	An invalid terminal type is associated with the issuing task.
46	A terminal I/O error has occurred.
47	The named area has not been readied.
48	The run unit has not been bound.
49	NOWAIT has been specified but WAIT is required.
50	Statistics are not being kept.
51	A lock manager error occurred during the processing of a KEEP LONGTERM request
52	The specified table is missing or invalid.
53	An error occurred from a user-written edit routine.
54	Either there is invalid internal data or a data conversion error has occurred.
55	The user-written edit routine cannot be found.

Minor Code	Function Status
56	No DFLDS have been defined for the map.
57	The ID cannot be found, is not a long-term permanent ID, or is being used by another run unit.
58	Either the LRID cannot be found, the maximum number of concurrent task threads was exceeded, or an attempt was made to rollback database changes in local mode.
59	An error occurred in transferring the KEEP LONGTERM request to IDMSKEEP
60	The requested KEEP LONGTERM lockid was already in use with a different page group
63	Invalid function parameters have been passed on the call to the DBMS.
64	No detail exists currently for update; no action has been taken. Alternatively, the requested node for a header or detail is either not present or not updated.
68	There are no more updated details to MAP IN or the amount of storage defined for pageable maps at sysgen is insufficient. In the latter case, subsequent MAP OUT DETAIL statements are ignored.
72	No detail occurrence, footer, or header fields exist to be mapped out by a MAP OUT RESUME command, or the scratch record that contains the requested detail could not be accessed. The latter case is a mapping internal error and should be reported.
76	The firstscreen page has been transmitted to the terminal.
77	Either the program is no longer signed on to the subschema or the variable subschema tables have been overwritten.
80	The target node is either not active or has been disabled.
97	An error was encountered processing a syncpoint request; check the log for details.
98	An unsupported COBOL compiler option (for example, DEBUG) has been specified for an online program or a program running in a batch region has issued a DML verb that is only valid when running online under CA IDMS/DC/UCF.
99	An unexpected internal return code has been received; the terminal device is out of service.

Note: For a complete description of DC runtime status codes, see the chapter "CA IDMS Status Codes" in the *Messages and Codes Guide*.

ERROR-STATUS Condition Names

Code	Condition name	Explanation
0000	DB-STATUS-OK	No error
0307	DB-END-OF-SET	End of set, area, or SPF index
0326	DB-REC-NOT-FOUND	No record found
0001 to 9999	ANY-ERROR-STATUS	Any nonzero status
0000 to 9999	ANY-STATUS	Any status
3101 3201 3401 3901	DC-DEADLOCK	Waiting will cause a deadlock
3202 3402	DC-NO-STORAGE	Insufficient space available
4303	DC-AREA-ID-UNK	ID cannot be found
4404	DC-QUEUE-ID-UNK	Queue header cannot be found
4305 4405	DC-REC-NOT-FOUND	Record cannot be found
3908	DC-RESOURCE-NOT-AVAI L	Resource not available
3909	DC-RESOURCE-AVAIL	Resource is available
3210	DC-NEW-STORAGE	New space allocated
3711	DC-MAX-TASKS	Maximum attached tasks
4317	DC-REC-REPLACED	Record has been replaced
4319 4419 4519 4719	DC-TRUNCATED-DATA	Return area too small; data has been truncated
4525 4625	DC-ATTN-INT	Attention interrupt received
4743	DC-OPER-CANCEL	Session cancelled

Autostatus Return Codes

Status Code	Meaning
0307	The end-of-set condition was encountered.

Status Code	Meaning
0326	The requested record cannot be found.
1707	The 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	A request queue record cannot be found.
5149	NOWAIT was specified in a KEEP LONGTERM request, and a wait is required.

Default Level-88 Values

Status Code	Condition Name
0000	DB-STATUS-OK
0307	DB-END-OF-SET
0326	DB-REC-NOT-FOUND
1707	DB-END-OF-INDEX
1726	DB-INDEX-NOT-FOUND
4303	SCRATCH-AREA-NOT-FOUND
4305	SCRATCH-REC-NOT-FOUND
4317	SCRATCH-REC-REPLACED
4404	QUEUE-ID-NOT-FOUND
4405	QUEUE-REC-NOT-FOUND
0001	THRU 9999 DB-ANY-ERROR

Appendix K: Online Debugger Syntax

This section contains the following topics:

[General Registers Symbols](#) (see page 751)

[DC/UCF System Symbols](#) (see page 752)

[Address Symbols and Markers](#) (see page 752)

[User Symbols](#) (see page 753)

[Program Symbols](#) (see page 753)

[Expression Operators](#) (see page 753)

[Delimiters](#) (see page 754)

[Debugger Commands](#) (see page 754)

General Registers Symbols

General registers include the registers used by the program at the time of execution and the registers used by the DC/UCF system. The program status word (PSW) and register definitions are always preceded by a colon (:) and are specified by these symbols:

- **:PSW** for the current program status word
- **:Rn** for the user program register at the time of interrupt, where *n* represents the number of the register and can have a value of 0 through 15
- **:REGS** for all user program registers at the time of interrupt
- **:SRn** for a DC/UCF system register at the time of interrupt, where *n* represents the number of the register and can have a value of 0 through 15
- **:SREGS** for all DC/UCF system registers at the time of interrupt

Important! A single debug expression can reference only one general register.

DC/UCF System Symbols

Certain DC/UCF system symbols also function as debugger entities, and you can refer to them during a debugging session. A colon (:) must precede each symbol. These are the valid symbols:

:BAT

Specifies the base address table for session.

:CSA

Specifies the DC/UCF common storage area.

:DLB

Specifies the debug local block, control block required for debugging session.

:LTE

Specifies the current logical terminal element.

:PTE

Specifies the current physical terminal element.

:TCE

Specifies the current task control element.

:VECT

Specifies the vector table for debugger.

Important! A single debug expression can reference only one system entity.

Address Symbols and Markers

Symbol	Symbol Name	Designated Location
@	At sign	Absolute address
\$	Dollar sign	Load address
¢	Cent sign	Address of current dialog process

User Symbols

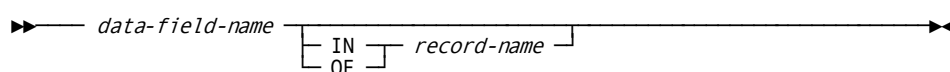
- **:DR n** for a debugger general register, where n represents the number of the register and can have a value of 0 through 15
- **:DREGS** for all debugger registers
- **:H1** and **:H2** for halfword 1 and halfword 2
- **:F1** and **:F2** for fullword 1 and fullword 2
- **:UCHR** for a 48-byte character area

You can also refer to specified sections of this area:

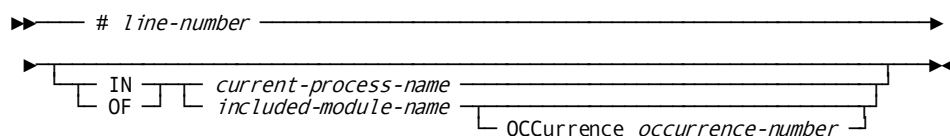
- **:UC0**, the first 16 bytes
- **:UC16**, the next 16 bytes
- **:UC32**, the last 16 bytes

Program Symbols

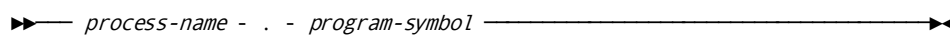
Syntax: Data Field Names



Syntax: Line Numbers



Syntax: Qualifying Program Symbols



Expression Operators

Operator	Meaning
+	Addition
-	Subtraction

Operator	Meaning
*	Multiplication
/	Division

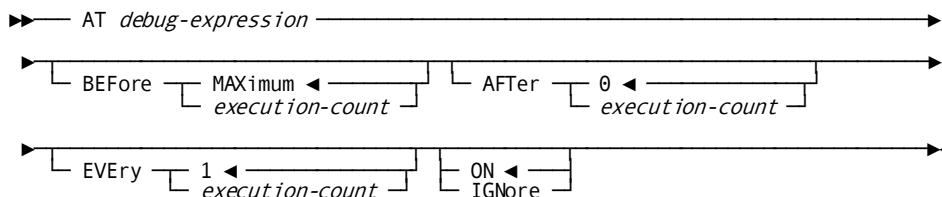
Delimiters

Delimiter	Meaning
*	Asterisk
	Blank
,	Comma
=	Equal sign
!	Exclamation point
-	Hyphen
%	Percent sign
.	Period
+	Plus sign
/	Slash

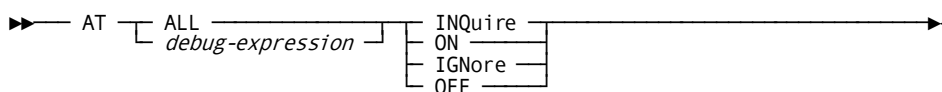
Debugger Commands

Syntax: AT

ADD Format

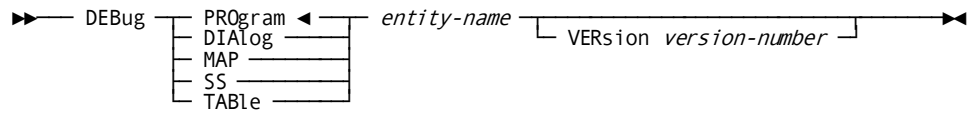


INQUIRE Format

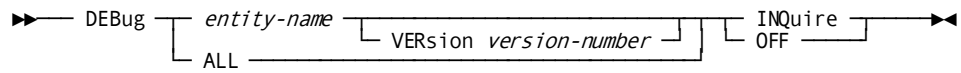


Syntax: DEBUG

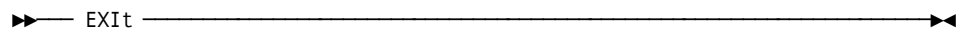
ADD format



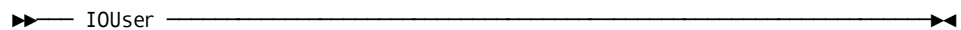
INQUIRE format



Syntax: EXIT

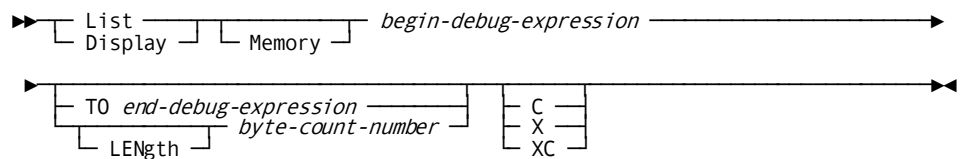


Syntax: IOUSER

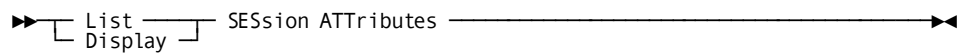


Syntax: LIST

MEMORY Format



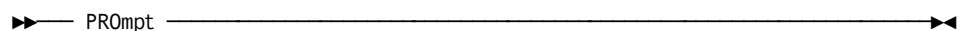
ATTRIBUTES Format



Syntax: MENU

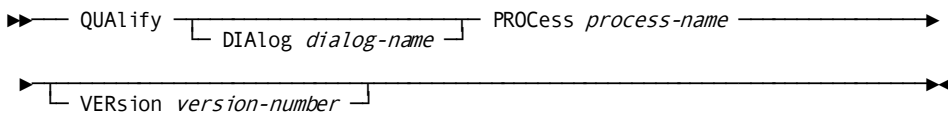


Syntax: PROMPT

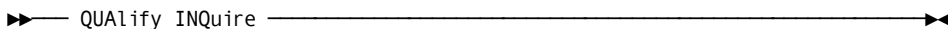


Syntax: QUALIFY

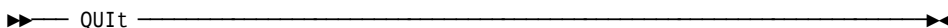
RESET Format



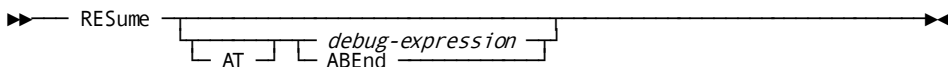
INQUIRE Format



Syntax: QUIT

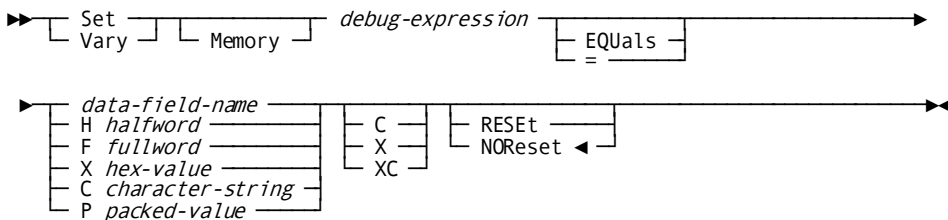


Syntax: RESUME

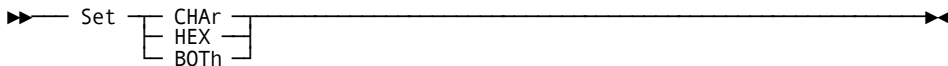


Syntax: SET

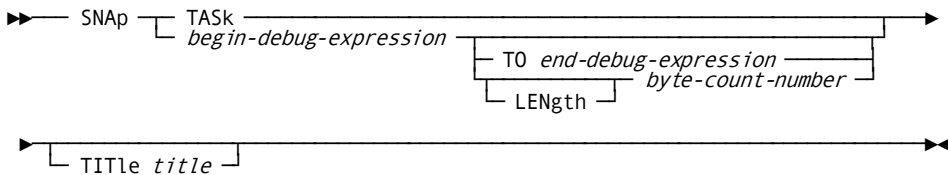
MEMORY Format



ATTRIBUTES Format



Syntax: SNAP



Syntax: WHERE

▶▶ — WHEre ————— ▶▶

Index

#

#EFMBIFS macro • 702
#EFUNMOD macro • 705
#EFUNMST macro • 701

\$

\$BACKWARD condition • 261
\$BATCH condition • 256
\$DETAIL condition • 261
\$DETAIL-NOT-FOUND condition • 261
\$END-OF-DATA condition • 261
\$END-OF-FILE condition • 248
\$ERROR-COUNT field • 287
\$FORWARD condition • 261
\$HEADER condition • 261
\$INPUT-COUNT field • 287
\$IOERROR condition • 248
\$MAXIMUM-DETAILS-PUT condition • 261, 477
\$MESSAGE field • 257, 477
\$ONLINE condition • 256
\$OUTPUT-COUNT field • 287
\$PAGE field • 257, 466
\$PAGE-READY condition • 261
\$RESPONSE field • 257

A

ABORT command • 510
ABSOLUTE-VALUE • 185
ACCEPT command • 373, 513
access module • 109
activity logging • 676
 function commands • 676
 function numbers • 676
ADD command • 306
ADSL • 47
ADSM • 48
ADSO-APPLICATION-GLOBAL-RECORD • 135, 339
 AGR-CURRENT-RESPONSE • 135, 339
ADSO-APPLICATION-MENU-RECORD • 124
 at runtime • 124
ADSOBCOM • 621, 623, 650
 control statements • 623
 JCL and command statements • 650
ADSOBSYS • 654, 656

 ADSOOPTI load module • 654
 JCL and command statements • 656
ADSOBTAT • 662, 666
 JCL and command statements • 666
 task application table (TAT) • 662
ADSOMSON menu map • 126
ADSOMUR1 menu map • 126
ADSOMUR2 menu map • 126
ADSORPTS • 584, 595, 596
 application reports • 595
 control statements • 596
 dialog debugging • 584
 dialog reports • 584
ADSO-STAT-DEF-REC record • 281
ADSOTATU • 671
AFACT-057 record • 676
ALLOCATE command • 535
ALLOWING clause • 280
APPC (Advanced Program to Program
 Communication) • 528
APPC status codes • 552
APPCCODE status code • 552, 553
APPCERC status code • 552, 553
application compiler (ADSA) • 57, 60, 63, 66, 70, 74,
 77, 80, 85, 87
 control key assignments • 57
 Function Definition (Menu) screen • 80
 Function Definition (Program) • 77
 Function Definition screen • 74
 General Options • 60
 General Options screen?Page 2 • 63
 General Options?Page 1 • 60
 Global Records screen • 85
 Response Definition screen • 70
 Response/Function List screen • 66
 Task Codes screen • 87
application compiler sequence • 53
 screen sequence • 53
application compiler session • 52, 56
 invoking • 52
 suspending • 56
application definition block (ADB) • 516
application security • 719
application structure • 328
 levels of • 328

- mainline dialogs • 328
- application thread • 327, 328
 - definition of • 327
 - menu stack • 328
 - nonoperative dialogs • 327
 - operative dialogs • 327
- applications • 57, 66, 77, 80, 85, 87
 - compiling of • 57
 - defining global records • 85
 - defining responses and functions • 66
 - defining task codes • 87
 - specifying control blocks • 77
 - specifying menus • 80
 - specifying record buffers • 77
- APPLICATIONS statement • 596
- arc cosine values • 186
- arc sine values • 187
- arc tangent values • 188
- AREPORTs • 583
- arithmetic built-in functions • 185, 207, 210, 211, 212, 213, 214, 216, 220, 223
 - ABSOLUTE-VALUE • 185
 - INVERT-SIGN • 207
 - LOG-BASE-10 • 210
 - LOG-BASE-E • 210
 - MODULO • 211
 - next integer equal or higher • 212
 - next integer equal or lower • 213
 - NUMERIC • 214
 - RANDOM-NUMBER • 216
 - sign inversion • 207
 - SIGN-VALUE • 220
 - SQUARE-ROOT • 223
- arithmetic commands • 306, 307, 309, 311, 313
 - ADD • 306
 - COMPUTE • 307
 - DIVIDE • 309
 - MULTIPLY • 311
 - SUBTRACT • 313
- arithmetic expressions • 171, 173, 174, 436
 - binary operations • 171
 - coding rules • 174
 - operands • 171
 - order of evaluation • 173
 - unary operations • 171
 - variable data fields • 173
 - WHERE clause • 436
- assigned key • 66
- assignment command • 314

- MOVE • 314
- automatic editing • 139, 257
- autostatus facility • 278

B

- BACKWARD function • 25
- batch control event conditions • 248
 - \$END-OF-FILE (\$EOF) • 248
 - \$IOERROR (\$IOERR) • 248
- batch processing • 287
 - \$ERROR-COUNT • 287
 - \$INPUT-COUNT • 287
 - \$OUTPUT-COUNT • 287
- binary data • 161
- BIND PROCEDURE command • 383
- BS2000/OSD JCL • 605
 - ADSORPTS • 605
- built-in functions • 150, 178, 179, 180, 681, 699, 713, 714, 715
 - calling user-defined functions • 715
 - changing invocation names • 681
 - coding user-defined functions • 715
 - creating user-defined functions • 714
 - data type conversion • 178
 - date formats • 180
 - error processing • 178
 - internal structure • 681
 - invocation name • 178, 713
 - omitted optional parameter • 179
 - parameters • 178
 - runtime processing • 699
 - user-defined • 179, 714
 - with LRF • 715

C

- CA ADS • 314
 - conversion rules • 314
- CA ADS comment character • 160
- CA IDMS statistics block • 380
- CALL command • 505
- CHANGED condition • 257
- characteristic • 273
- checkouts • 45, 47, 48
 - explicit • 45
 - implicit • 45
 - listing • 47
 - modifying with ADSM • 48
 - releasing with ADSM • 48

checkpoint • 143, 428
 database • 428
 queue • 428
 scratch • 428
CLOSE command • 454
COBOL • 314
 conversion rules • 314
COBOL moves • 101, 167, 306, 307, 309, 311, 313,
 314, 315
coding • 160, 174
 arithmetic expressions • 174
 CA ADS comment character • 160
 general rules • 160
 SQL comment character • 160
command status condition • 249
 ERROR-STATUS • 249
Commands • 300, 303, 306, 307, 309, 311, 313, 314,
 317, 318, 319, 321, 322, 325, 332, 334, 339, 341,
 343, 345, 353, 356, 358, 365, 373, 383, 384, 386,
 389, 391, 394, 411, 412, 413, 420, 426, 428, 429,
 433, 438, 439, 440, 445, 449, 454, 472, 474, 483,
 486, 488, 491, 496, 498, 502, 505, 506, 507, 509,
 510, 513, 516, 518, 519, 523
 ABORT • 510
 ACCEPT • 373, 513
 ADD • 306
 arithmetic • 303
 assignment • 314
 BIND PROCEDURE • 383
 CALL • 505
 CLOSE • 454
 COMMIT • 384
 COMPUTE • 307
 conditional • 317
 CONNECT • 386
 CONTINUE • 332
 control • 325
 DEFINE • 506
 DELETE QUEUE • 486
 DELETE SCRATCH • 496
 DISCONNECT • 389
 DISPLAY • 334
 DIVIDE • 309
 ERASE • 391, 438
 EXECUTE NEXT FUNCTION • 339
 EXIT • 318
 FIND/OBTAIN • 394
 GET • 411
 GET DETAIL • 472
 GET QUEUE • 488
 GET SCRATCH • 498
 GOBACK • 507
 IF • 319
 INCLUDE • 300
 INVOKE • 341
 KEEP • 412
 KEEP LONGTERM • 413
 LEAVE • 343
 LINK • 345
 LRF • 433
 map modification • 449
 MODIFY • 420, 439
 MULTIPLY • 311
 navigational database access • 365
 NEXT • 321
 OBTAIN • 440
 pageable map • 449
 PUT DETAIL • 474
 PUT QUEUE • 491
 PUT SCRATCH • 502
 queue management • 483
 READ TRANSACTION • 353
 RETURN • 353
 RETURN DB-KEY • 426
 ROLLBACK • 428
 scratch management • 483
 SNAP • 516
 STORE • 429, 445
 SUBTRACT • 313
 TRACE • 518
 TRANSFER • 356
 utility • 509
 WHILE • 322
 WRITE PRINTER • 519
 WRITE TO LOG/OPERATOR • 523
 WRITE TRANSACTION • 358
command-statements • 319, 322
 DO • 319
 ELSE • 319
 END • 319, 322
 REPEAT • 322
 THEN • 319
COMMIT command • 384
comparison conditions • 251
 CONTAINS • 251
 mask characters • 251
 MATCHES • 251
 operators • 251

compiler (ADSA) • 718
 security • 718
 compiler (ADSC) • 718
 security • 718
 COMPUTE command • 307
 CONCATENATE built-in function • 189
 conditional commands • 318, 319, 321, 322
 EXIT • 318
 IF • 319
 NEXT • 321
 WHILE • 322
 conditional expressions • 246, 248, 249, 251, 253,
 254, 256, 257, 261, 265, 450
 \$MESSAGE field • 257
 \$PAGE field • 257
 \$RESPONSE field • 257
 batch control event condition • 248
 command status condition • 249
 comparison condition • 251
 cursor position condition • 253
 dialog execution status condition • 254
 environment status condition • 256
 for maps • 450
 level-88 condition • 257
 map field status condition • 257
 map paging status conditions • 261
 operators in • 246
 order of precedence • 246
 set status condition • 265
 summary • 248
 CONFIRM command • 538
 CONFIRMED command • 539
 CONNECT command • 386
 constants • 269, 271, 272, 273
 figurative constants • 269
 fixed-point numeric literals • 273
 floating-point numeric literals • 273
 graphic literals • 271
 multibit binary • 272
 nonnumeric literals • 273
 numeric literals • 273
 CONTAINS condition • 251
 CONTINUE command • 332
 control commands • 332, 334, 339, 341, 343, 345,
 353, 356, 358, 383, 454, 523
 BIND PROCEDURE • 383
 CLOSE • 454
 CONTINUE • 332
 DISPLAY • 334
 EXECUTE NEXT FUNCTION • 339
 INVOKE • 341
 LEAVE • 343
 LEAVE APPLICATION • 343
 LINK • 345
 READ TRANSACTION • 353
 RETURN • 353
 TRANSFER • 356
 WRITE TO LOG/OPERATOR • 523
 WRITE TRANSACTION • 358
 control event • 66, 248
 control keys • 135
 default assignments • 135
 CONTROL SESSION command • 540
 control statements • 624, 626, 655, 664
 ADSOBSYS • 655
 ADSOBTAT • 664
 COMPILE • 624
 DECOMPILE • 626
 conversation, ending • 547
 conversion • 314
 CA ADS rules • 314
 COBOL rules • 314
 cosine values • 190
 currency • 329, 365, 397, 426, 485, 494
 database • 329, 365
 index • 426
 of run unit • 397
 queue • 485
 scratch requests • 494
 cursor position condition • 253
 CURSOR-COLUMN • 253
 CURSOR-ROW • 253
 cursor position data field • 287
 CURSOR-COLUMN • 287
 CURSOR-ROW • 287
 CURSOR-COLUMN condition • 253
 CURSOR-ROW condition • 253

D

data dictionary • 676
 AFACT-057 • 676
 LR-190 • 676
 LRACT-193 • 676
 organization • 676
 RCDACT-059 • 676
 SETACT-061 • 676
 SSA-024 • 676

SSOR-034 • 676
 SSR-032 • 676
 data types • 161, 167, 314
 available to CA ADS • 161
 binary • 161
 conversion of • 167
 conversion rules • 314
 definition of • 161
 examples of • 167
 floating point • 161
 group • 161
 multibit binary • 161
 packed decimal • 161
 zoned decimal • 161
 database • 142, 365, 374, 376, 380, 384, 386, 394,
 399, 413, 420, 423, 426, 429
 access of • 142
 CA IDMS statistics block • 380
 CALC key • 394
 checkpoint • 384
 currency • 365
 db-key • 374, 376, 399, 426
 location modes • 429
 modification of CALC elements • 420
 modification of sort-control elements • 420
 monitoring activity • 413
 set membership options • 386
 statistics • 380
 usage modes • 423
 database access • 141, 434
 Logical Record Facility (LRF) • 434
 date built-in functions • 191, 194, 196, 197, 199,
 200, 203, 230, 231, 233, 238, 241
 DATECHG • 191
 DATEDIF • 194
 DATEEXT • 196
 DATEINT • 196
 DATEOFF • 197
 DATETIMX • 199
 DISPDT • 199
 DTINT • 200
 GOODDATE • 203
 TIMEINT • 230
 TODAY • 231
 TOMORROW • 233
 WEEKDAY • 238
 YESTERDAY • 241
 date formats • 180, 287
 calendar • 180
 European • 180
 Gregorian • 180
 Julian • 180, 287
 date offset • 197
 DATECHG built-in function • 191
 DBCS data • 271, 273, 304
 as a graphic literal • 271
 as a nonnumeric literal • 273
 storage of • 304
 deadlock • 143
 DEALLOCATE command • 541
 debugging • 148
 DEFINE command • 506
 DELETE QUEUE command • 486
 DELETE SCRATCH command • 496
 Design guidelines • 547
 detail area • 465
 diagnostic table • 101
 dialog • 584
 FDB • 584
 Dialog Abort Information screen • 145, 510, 584
 enabling of • 510
 dialog compiler (ADSC) • 35, 92, 94, 98, 101
 control keys • 94
 Dialog Summary Report screen • 35
 Map Image screen • 35
 Options and Directives • 101
 screens • 98
 session • 92
 dialog compiler session • 92, 97
 invoking • 92
 suspending • 97
 dialog execution status • 254
 FIRST-TIME • 254
 dialog expression (ADSOBCOM) • 628
 dialog function • 25
 Dialog Selection screen • 120
 dialog statistics • 611, 615, 616, 617, 618
 CA ADS statistics block • 611
 checkpoint interval • 617
 enabling of • 615
 runtime collection and writing • 617
 selecting • 616
 statistics block identifiers • 617
 statistics reporting • 618
 Dialog Summary screen • 35
 dialog, location of allocated • 547
 dialogs • 119
 mainline • 119

DIALOGS statement • 599
DISCONNECT command • 389
DISPLAY command • 334
 mapout rules • 334
DIVIDE command • 309
DO command-statement • 319
dumps • 510
 snapdumps • 510

E

EBCDIC data type • 304
EDIT IS ERROR/CORRECT condition • 257
ELSE command-statement • 319
END command-statement • 319, 322
environment status condition • 256
 \$BATCH • 256
 \$ONLINE • 256
ERASE command • 391, 438
ERASED condition • 257
error expressions • 279
error handling • 278, 279, 280, 281, 699
 ADSO-STAT-DEF-REC • 281
 ALLOWING clause • 280
 autostatus • 278
 built-in functions • 699
 error expressions • 279
 level-88 condition names • 281
 site defined status definition record • 281
 STATUS clause • 281
 status definition record • 281
 system-defined status definition record • 281
error messages • 455
 suppression of • 455
ERROR-STATUS condition • 249
exclusive usage mode • 423
EXECUTE NEXT FUNCTION command • 339
 mapless dialog • 339
EXECUTE ON EDIT ERRORS command • 257
execution modes • 60, 91
EXIT command • 318
explicit checkouts • 45
explicit releases • 45
extended run units • 143
 checkpoint • 143
 deadlock • 143
EXTRACT built-in function • 201

F

fast mode • 60
figurative constants • 269
FIND/OBTAIN • 394
 FIND • 394
FIRST-TIME condition • 254, 356
FIX built-in function • 202
fixed dialog block (FDB) • 516, 584
 contents of • 584
fixed-point numeric literals • 273
floating-point numeric literals • 273
flow of control • 135, 139
 automatic editing • 139
 default control key assignments • 135
footer area • 465
FORWARD function • 25
Function Definition (Dialog) screen • 74
Function Definition (Menu) screen • 80
Function Definition (Program) screen • 77

G

General Options screen?Page 2 • 63
General Options?Page 1 • 60
GET command • 411
GET DETAIL command • 261, 472
GET QUEUE command • 488
GET SCRATCH command • 498
Global Records screen (ADSA) • 85
GOBACK command • 507
GOODDATE built-in function • 203
GOODTRAILING built-in function • 204
graphic literals • 271
group data type • 161

H

header area • 465
HELP function • 25
help screen • 133

I

ICTL statement • 622, 655
 ADSOBCOM • 622
 ADSOBSYS • 655
IDENTICAL condition • 257
IF command • 319
implicit checkouts • 45
implicit releases • 45

IN ERROR condition • 257
INCLUDE command • 300
INDEX built-in function • 223
INITCAP built-in function • 205
INITIALIZE RECORDS command • 515
INSERT built-in function • 206
INSERT directive • 300
intermediate result area (IRA) • 699
INVERT-SIGN built-in function • 207
INVOKE command • 341
ISEQ statement • 622, 655
 ADSOBCOM • 622
 ADSOBSYS • 655

K

KEEP commands • 412

L

LEAVE APPLICATION command • 343
LEAVE command • 343
LEFT-JUSTIFY built-in function • 208
level-88 condition • 249, 257
LIKE built-in function • 209
LINK command • 149, 345
 linking to OLQ • 149
 nesting • 345
 with a user program • 345
LIST statement • 603
local mode processing • 604
 SYSIDMS parameter file • 604
location mode • 429
 CALC • 429
 DIRECT • 429
 VIA • 429
LOG-BASE-10 built-in function • 210
LOG-BASE-E built-in function • 210
Logical Record Facility • 345
 linking to dialog with LRF subschema • 345
logical records • 433
 in database access • 433
 path • 433
LR-190 record • 676
LRACT-193 record • 676
LRF • 442
 path status • 442
LRF commands • 434, 438, 439, 440, 442, 445
 ERASE • 438
 MODIFY • 439

OBTAIN • 440
ON command • 442
STORE • 445
WHERE clause • 434
LU 6.2 • 528

M

mainline dialogs • 328
mantissa values • 273
map field status conditions • 257
 ALL BUT • 257
 CHANGED • 257
 ERASED • 257
 EXCEPT • 257
 IDENTICAL • 257
 IN ERROR • 257
 pageable map considerations • 257
 TRUNCATED • 257
Map Image screen • 35
map modification commands • 450, 455
 ATTRIBUTES • 450
 MODIFY MAP • 455
map paging session • 466
map paging status condition • 261
maps • 450, 455
 conditional expressions • 450
 output data options • 455
 permanent modifications • 450, 455
 suppressing error message • 455
 temporary modifications • 450, 455
mask character • 209, 251, 436
master function table • 681, 682
MATCHES condition • 251
matching string • 209
menu definition • 124
menu function • 25
menu maps • 126
 ADSOMSON • 126
 ADSOMUR1 • 126
 ADSOMUR2 • 126
 signon • 126
 site-defined • 126
 system-defined • 126
menu stack • 328
menu/dialog function • 25
message codes • 334
messages • 334
modified data tags (MDTs) • 455

- resetting • 455
- setting for map fields • 455
- MODIFY command • 420, 439
- MODIFY MAP command • 257, 455
- MODULO built-in function • 211
- MOVE command • 314
- multibit binary constants • 272
- multibit binary data type • 161
- multiple databases • 287
 - accessing • 287
- MULTIPLY command • 311

N

- native VSAM data sets • 368, 376, 386, 389, 391, 402, 420
 - currency requests • 376
 - set status condition • 368
 - with CONNECT • 386
 - with DISCONNECT • 389
 - with ERASE • 391
 - with FIND/OBTAIN OWNER • 402
 - with MODIFY • 420
- natural logarithm • 210
- navigational DML commands • 373, 384, 386, 389, 391, 394, 411, 412, 420, 423, 426, 428, 429
 - ACCEPT • 373
 - COMMIT • 384
 - CONNECT • 386
 - DISCONNECT • 389
 - ERASE • 391
 - FIND/OBTAIN • 394
 - GET • 411
 - KEEP • 412
 - MODIFY • 420
 - READY • 423
 - RETURN DB-KEY • 426
 - ROLLBACK • 428
 - STORE • 429
- nesting • 345
- NEXT command • 321
- NEXT-INTEGGER-EQUAL-HIGHER built-in function • 212
- NEXT-INTEGGER-EQUAL-OR-LOWER built-in function • 213
- nonnumeric literals • 273
- NUMERIC built-in function • 214
- numeric fields • 304
- numeric literals • 273

O

- OBTAIN command • 440
- OCTL statement • 622, 655
 - ADSOBCOM • 622
 - ADSOBSYS • 655
- ON command • 442
- online control block (OCB) • 516
- online help • 48, 124
 - in CA ADS applications • 124
 - in CA ADS compilers • 48
- online terminal block (OTB) • 516
- online terminal block extension (OTBX) • 516
- online work area (OWA) • 516
- Options and Directives • 101

P

- packed decimal data type • 161
- page=end CA OLQ • 150
- page=start CA OLQ • 149
- pageable map commands • 472, 474
 - GET DETAIL • 472
 - PUT DETAIL • 474
- pageable maps • 105, 465, 466, 471
 - \$PAGE field • 466
 - areas of • 465
 - Auto display specification • 105
 - Backpage specification • 105
 - flow of control • 466
 - map paging dialog options • 105, 471
 - map paging session • 466
 - UPDATE specification • 105
- parameters for process commands • 159
 - keywords • 159
 - variable terms • 159
- path status (LRF) • 442
- POP function • 25
- POPTOP function • 25
- PREPARE-TO-RECEIVE command • 543
- printer output • 455
- process commands • 160
 - coding of • 160
 - comment character • 160
 - quoted strings • 160
- process modules • 157
 - premap • 157
 - response • 157
- processing • 527
 - cooperative • 527

program function • 25
protected usage mode • 423
PUT DETAIL command • 257, 474
PUT QUEUE command • 491
PUT SCRATCH command • 502

Q

queue management commands • 486, 488, 491
 DELETE QUEUE • 486
 GET QUEUE • 488
 PUT QUEUE • 491
queue records • 485
QUIT function • 25

R

RANDOM-NUMBER built-in function • 216
RCDACT-059 record • 676
READ TRANSACTION command • 353
READY command • 423
RECEIVE-AND-WAIT command • 543
record buffer block (RBB) • 516
record locking • 485
 queue • 485
record locks • 369, 371, 384
 deadlock conditions • 369
 exclusive • 369
 explicit • 369
 implicit • 369
 long-term explicit • 369
 release of • 384
 retrieval locks • 371
 shared • 369
records • 485, 494
 queue • 485
 scratch • 494
recovery • 428
releases • 45
 explicit • 45
 implicit • 45
releasing entities • 45
remainder values • 211
REPEAT command-statement • 322
repeat string • 225
REPLACE built-in function • 218
replace string • 218
REQUEST-TO-SEND command • 545
REQUEST-TO-SEND-RECEIVED system field • 552,
 557

reset keyboard • 455
response • 25, 124
 runtime selection • 124
Response Definition screen (ADSA) • 70
response process • 114
 security • 114
Response/Function List screen • 66
Response/Function List screen (ADSA) • 66
Response/Function search • 66
responses • 133, 135
 runtime selection • 133
 security • 135
RETURN command • 353
RETURN DB-KEY command • 426
RETURN function • 25
RHDCEVBF source module • 702
RIGHT-JUSTIFY built-in function • 219
ROLLBACK command • 428
run units • 142, 143
 extended • 143
runtime system • 119
 ADSOMAIN • 119
 ADSORUN1 • 119
 initiation of • 119

S

scratch area • 494
scratch management commands • 496, 498, 502
 DELETE SCRATCH • 496
 GET SCRATCH • 498
 PUT SCRATCH • 502
scratch records • 494
screens • 35, 57, 60, 63, 66, 70, 74, 77, 80, 85, 87,
 98, 101, 120, 124, 133, 145, 672
 application compiler • 57
 Dialog Abort Information screen • 145
 Dialog Selection screen • 120
 Dialog Summary Report screen • 35
 Function Definition (Dialog) screen • 74
 Function Definition (Menu) screen • 80
 Function Definition (Program) • 77
 General Options?Page 1 (ADSA) • 60
 General Options?Page 2 (ADSA) • 63
 Global Records • 85
 help • 133
 Menu definition • 124
 Options and Directives (ADSC) • 101
 Response Definition • 70

Response/Function List • 66
Response/Function Listscreen • 66
screens • 98
task codes • 87
TAT Update Utility • 672
SEARCH Statement • 603
search, Response/Function • 66
security • 63, 114, 621, 719, 720
 ADAPGOP2 • 63
 ADSOBCOM • 621
 General Options screen?Page 2 • 63
 response • 719
 response process • 114
 security-tailored menus • 719
 signon • 720
SEND/RECEIVE commands • 527, 534
 summary • 534
SEND-DATA command • 545
SEND-ERROR command • 546
SET condition • 265
set membership options • 386
SETACT-061 record • 676
sign inversion • 207
SIGNOFF function • 25, 720
SIGNON function • 25, 720
SIGNON statement (ADSOBCOM) • 623
SIGN-VALUE built-in function • 220
sine values • 220
SNA (Systems Network Architecture) • 528
SNAP command • 516
sort key • 408
sorted set • 408
SQL • 109
 access module • 109
 compliance • 109
SQL comment character • 160
SQUARE-ROOT • 223
SREPORTs • 618
SSA-024 area • 676
SSOR-034 set • 676
SSR-032 record • 676
status codes • 553
status definition record • 281
 ADSO-STAT-DEF-REC • 281
 level-88 condition names • 281
 site defined • 281
 STATUS clause • 281
 system defined • 281
step mode • 60
storage • 151
 management • 151
 XA • 151
STORE command • 429, 445
string built-in functions • 189, 201, 202, 205, 206,
208, 209, 218, 219, 223, 224, 225, 226, 232, 234,
236, 237, 240
 CONCATENATE • 189
 EXTRACT • 201
 FIX • 202
 INDEX • 223
 INITCAP • 205
 INSERT • 206
 LEFT-JUSTIFY • 208
 LIKE • 209
 REPLACE • 218
 RIGHT-JUSTIFY • 219
 STRING-INDEX • 223
 STRING-LENGTH • 224
 STRING-REPEAT • 225
 SUBSTRING • 226
 TOLOWER • 232
 TOUPPER • 234
 TRANSLATE • 236
 VERIFY • 237
 WORDCAP • 240
string verification • 237
STRING-LENGTH built-in function • 224
STRING-REPEAT built-in function • 225
subroutine control commands • 505, 506, 507
 CALL • 505
 DEFINE • 506
 GOBACK • 507
Subschema Control Block • 384
SUBSCHEMA-CONTROL • 345
SUBSTRING built-in function • 226
SUBTRACT command • 313
suspense file • 358
symbol table • 101
SYSIDMS parameters • 604
 for physical requirements • 604
system fields • 552, 557
system functions • 25, 133, 720
 BACKWARD • 25
 FORWARD • 25
 HELP • 25, 133
 POP • 25
 POPTOP • 25
 QUIT • 25

- RETURN • 25
- SIGNOFF • 25, 720
- SIGNON • 25, 720
- TOP • 25
- SYSTEM statement (ADSOBSYS) • 655
- system-supplied data fields • 287
 - \$ERROR-COUNT • 287
 - \$INPUT-COUNT • 287
 - \$OUTPUT-COUNT • 287
 - CURSOR-COLUMN • 287
 - CURSOR-ROW • 287
 - DATE • 287
 - DB-NAME • 287
 - DIRECT-DBKEY • 287
 - ERROR-STATUS • 287
 - LENGTH • 287
 - NODE-NAME • 287
 - TIME • 287

T

- tables • 101
 - diagnostic • 101
 - symbol • 101
- task • 141
- task application table (TAT) • 87, 120, 516
- taskcode, for TCF • 52
- Task Codes screen (ADSA) • 87
- TAT Update Utility • 672
- test conditions • 434
 - WHERE clause • 434
- THEN command-statement • 319
- time built-in functions • 229
 - TIMEEXT • 229
- TODAY built-in function • 231
- TOLOWER built-in function • 232
- TOMORROW built-in function • 233
- TOP function • 25
- TOUPPER built-in function • 234
- TRACE • 518, 725
- trace facility • 725
- trailing sign built-in functions • 204, 235, 242
 - GOODTRAILING • 204
 - TRAILING-TO-ZONED • 235
 - ZONED-TO-TRAILING • 242
- TRAILING-TO-ZONED built-in function • 235
- TRANSFER command • 356
- transfer control facility • 52, 92
- TRANSLATE built-in function • 236

- trigonometric built-in functions • 186, 187, 188, 190, 220
 - ARCCOSINE-DEGREES • 186
 - ARCCOSINE-RADIANS • 186
 - ARCSINE-DEGREES • 187
 - ARCSINE-RADIANS • 187
 - ARCTAN-DEGREES • 188
 - ARCTAN-RADIANS • 188
 - COSINE-DEGREES • 190
 - COSINE-RADIANS • 190
 - SINE-DEGREES • 220
 - SINE-RADIANS • 220
- TRUNCATED condition • 257

U

- usage modes • 423
 - exclusive • 423
 - protected • 423
 - retrieval • 423
 - shared • 423
 - update • 423
- user program • 345
 - DC RETURN statement • 345
 - linking • 345
- user-defined • 285
 - user-defined • 285
- utilities • 583, 621, 654, 662, 671
 - ADSOBCOM • 621
 - ADSOBSYS • 654
 - ADSOBTAT • 662
 - ADSORPTS • 583
 - ADSOTATU • 671
- utility commands • 510, 513, 515, 516, 518, 519
 - ABORT • 510
 - ACCEPT • 513
 - INITIALIZE RECORDS • 515
 - SNAP • 516
 - TRACE • 518
 - WRITE PRINTER command • 519

V

- variable dialog block (VDB) • 516
- variables • 171, 245, 269, 279, 285, 287, 293
 - arithmetic expressions • 171
 - conditional expressions • 245
 - constants • 269
 - data fields • 285
 - entity names • 293

- error expressions • 279
- system-supplied • 287
- target fields • 287
- variable target fields • 287
- VDE module • 693
 - processing of • 693
- vector call codes • 584
- VERIFY built-in function • 237
- VXDE module • 681, 685, 693
 - processing of • 693

W

- WEEKDAY built-in function • 238
- WHAT-RECEIVED system field • 552, 557
- WHERE clause • 434, 436
 - comparison expression • 436
 - conditional expression • 434
 - test condition • 434
- WHILE command • 322
 - with FIND/OBTAIN DB-KEY • 399
- WORDCAP built-in function • 240
- WRITE TO LOG/OPERATOR command • 523
- WRITE TRANSACTION command • 358

X

- XDE module • 681, 683, 685

Y

- YESTERDAY built-in function • 241

Z

- z/OS and PS/390 JCL • 605
 - ADSORPTS • 605
- z/OS JCL • 650, 656
 - ADSOBCOM • 650
 - ADSOBSYS • 656
- z/VM commands • 605, 653, 660
 - ADSOBCOM • 653
 - ADSOBSYS • 660
 - ADSORPTS • 605
- z/VSE JCL • 605, 651, 658
 - ADSOBCOM • 651
 - ADSOBSYS • 658
 - ADSORPTS • 605
- zoned decimal data type • 161
- ZONED-TO-TRAILING built-in function • 242