Advantage™ CA-IDMS®

Release Summary

r16 SP2

Computer Associates®

B01250-3E

Third Edition, May 2005

Contents

Chapter 1. Introducing Advantage CA-IDMS 16.0

1.1 Welcome

Welcome to Advantage™ CA-IDMS® Release 16.0. This release incorporates many new features to enhance your use of Advantage CA-IDMS, including:

- Two-phase commit support
- SQL features
- Administrative and operational features
- Performance features
- Non-stop processing features
- Tool product enhancements
- TCP/IP API support
- Type 4 JDBC Driver

This chapter includes a brief overview of each of the Release 16.0 features and provides a high-level explanation of the upgrade requirements. The remaining parts of this guide describe the features in detail.

Part	Content
Chapter 2, “Upgrading to Release 16.0”	Describes actions and considerations related to upgrading to Release 16.0.
Chapter 3, “Two-Phase Commit Support”	Explains and illustrates the new Two-Phase Commit process and considerations for its use.
Chapter 4, “SQL Features”	Describes the new SQL features for improved performance, productivity, and open access.
Chapter 5, “Administrative and Operational Enhancements”	Describes: ■ Utility enhancements designed for improved DBA productivity ■ Utility and sysgen enhancements for the two-phase commit feature and TCP/IP ■ Security enhancements for the utility commands ■ Batch command facility (IDMSBCF) enhancements for the SET OPTIONS statement
Chapter 6, “Performance Enhancements”	Explains the enhancements for z/Architecture and DASD exploitation as well as journaling and recovery performance enhancements.
Chapter 7, “Non-Stop Processing Features”	Describes the new dynamic capabilities and improved messaging for enhanced system availability.

Part	Content
Chapter 8, “Tool Product Enhancements”	Describes the productivity enhancements made for: ■ Advantage™ CA-Culprit for CA-IDMS® ■ Advantage™ CA-IDMS® Database Journal Analyzer Option ■ Advantage™ CA-IDMS® Database Dictionary Module Editor (DME) Option ■ Advantage™ CA-IDMS® Database DML Online Option ■ Advantage™ CA-ADS®/Alive Option ■ Online Mapping Facility ■ Advantage CA-IDMS PL/I Compiler Enhancements ■ Support for 31-digit Packed Decimal Elements
Chapter 9, “TCP/IP API Support”	Describes the cross-platform capabilities available for applications using TCP/IP support.
Chapter 10, “Type 4 JDBC Driver”	Describes how to use the Type 4 JDBC driver supplied with Advantage CA-IDMS Server.
Appendix A, “New and Revised DCMT Commands”	Explains the DCMT commands that are new or changed with Release 16.0.
Appendix B, “New and Revised SQL Statements”	Describes the SQL statements and language elements that are new or changed with Release 16.0.
Appendix C, “SQL Functions and SQL Procedure Enhancements”	Explains when and how to use a user-defined function and provides samples for functions and procedures: ■ SQL function definition and execution for Advantage™ CA-ADS® for IDMS® dialogs ■ SQL procedure definition and execution for Advantage™ CA-ADS® for IDMS® dialogs ■ SQL function definition and execution for COBOL
Appendix D, “SQL ROWID Examples”	Provides several examples of how to use the new SQL ROWID feature.
Appendix E, “SQL Cache Tables”	Describes the tables that are involved in SQL caching and provides examples for the administrators of how to display and control the cache.

Part	Content
Appendix F, “CICS Interface Enhancements for Two-Phase Commit Support”	Describes how to: ■ Define and execute the CICS RSYN task and program for resynchronization ■ The changes for OPTIXIT to ensure resynchronization requests are routed to the correct back-end central version ■ The new and enhanced parameters added to CICSOPT and IDMSCINT to support Two-Phase Commit
Appendix G, “TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion”	Describes: ■ Each of the TCP/IP functions and associated parameters ■ The TCP/IP error codes, reason codes and return codes ■ The ASCII to EBCDIC conversion tables ■ The TCP/IP socket structures

1.2 New Features

New for SP2:

Following are listed the new features added to Advantage CA-IDMS at r16 SP2 and references to detailed descriptions about them.

Note: Be aware that updated information about Advantage CA-IDMS r16 that is not part of the new r16 SP2 features is also found in this document. Updates and new features are both marked by vertical revision bars in the margins.

New r16 SP2 Feature	Reference
SQL Enhancement	
SQL/XML functions are provided to allow Advantage CA-IDMS data to be easily published in XML documents.	See 4.6, “XML Publishing” on page 4-33.
Administrative and Operational Enhancements	
The batch command facility (IDMSBCF) has been enhanced to allow input/output reassignment to a file rather than to SYSIPT and SYSLST.	See 5.10, “IDMSBCF Input/Output Reassignment” on page 5-23.
The online compilers have been enhanced to increase the maximum data lines they can display from 20,916 lines to 41,916 lines. Also, the CV node name has been added in the command line header.	See 5.11, “Online Compiler Enhancements” on page 5-28.
The PRINT SPACE utility has been enhanced to provide space utilization reporting on a SUBAREA within an area.	See 5.12, “PRINT SPACE Utility Enhancement” on page 5-29.
The EXTRACT JOURNAL, ROLLBACK, and ROLLFORWARD utilities have been enhanced to allow processing of multiple segments.	See 5.13, “EXTRACT JOURNAL Utility Enhancement” on page 5-30, 5.14, “ROLLBACK Utility Enhancement” on page 5-31, and 5.15, “ROLLFORWARD Utility Enhancement” on page 5-32.
System setup and product maintenance in z/OS, z/VM, VSE/ESA, and BS2000/OSD have been made easier through enhancements in the way runtime options are specified.	For more information on this feature, see 5.16, “System Startup Enhancements” on page 5-33.

New r16 SP2 Feature	Reference
The IDMSLOOK utility and the LOOK system task have been enhanced to: ■ Display page reserve, SMP interval, and symbolic values when using the DMCL ALL function. ■ Display area procedures when using the SUBCHEMA and BIND SUBCHEMA functions.	These features are transparent in using the IDMSLOOK utility and the LOOK system task. To take advantage of these features, see the *Advantage CA-IDMS Utilities* and *Advantage CA-IDMS System Tasks and Operator Commands* guides.
The TUNE INDEX utility has been enhanced to allow a COMMIT ALL at the end of an area to free up shared locks.	This feature is transparent in using the TUNE INDEX utility. To take advantage of this feature, see the *Advantage CA-IDMS Utilities* guide.
The number of symbolics allowed in a DMCL has been increased from 32,768 to 2,147,483,648.	This feature is transparent in using DMCLs. To take advantage of this feature, see the *Advantage CA-IDMS Database Administration Guide*.
The DML trace facility has been enhanced to also trace the following functions: ■ COMMIT TASK ■ COMMIT TASK ALL ■ FINISH TASK ■ ROLLBACK TASK ■ ROLLBACK TASK ALL ■ ROLLBACK TASK CONTINUE	This feature is transparent in using the DML trace facility. To take advantage of this feature, see the *Advantage CA-IDMS Navigational DML Programming Guide*.
The #WTL macro has been enhanced to allow the passing of the CV node name and the Advantage CA-IDMS/DC release number.	See 5.17, “#WTL Macro Enhancements” on page 5-39.
The maximum number of pages that can be generated by line-mode terminal I/O has been increased from 999 to 32,767.	This feature is transparent to using line-mode I/O and system commands such as DCMT that rely on line-mode I/O.
The hexadecimal to character translation has been enhanced in all modules that perform memory snaps to use the common RHDCCODE translation tables. This allows the translation of additional characters, such as lowercase letters.	This feature is transparent in using the affected modules.

New r16 SP2 Feature	Reference
Performance Enhancements	
A MEMORY CACHE option has been added to the ALTER DMCL statement and DCMT VARY DMCL command to allow dynamically changing options to control where and how much memory cache storage can be allocated.	For more information on using the MEMORY CACHE option, see 6.2.3, “Altering the DMCL Definition” on page 6-5 and A.18, “DCMT VARY DMCL” on page A-29 For updated DCMT command codes, see A.32, “Command Codes” on page A-51.
A special form of storage protect is now available for the production system, which provides negligible processing overhead yet protects Advantage CA-IDMS and the operating system from user written code.	See 6.8, “High Performance Storage Protection” on page 6-18.
TCP/IP API Support Ehancement	
Additional sample TCP/IP client and generic listener server programs are provided in the following programming languages: ■ Advantage CA-ADS ■ Assembler ■ COBOL ■ PL/I	For more information, see *Advantage CA-IDMS Callable Services*.
JDBC Driver Ehancement	
A built-in TCP/IP server program is provided that enables Advantage CA-IDMS Server to function as a "Type 4" JDBC driver. This allows applications to communicate directly to the Advantage CA-IDMS address space using the native "wire" protocol, with no intervening middleware.	See Chapter 10, “Type 4 JDBC Driver.”
DCMT Command Ehancements	
The DCMT SHUTDOWN command has been enhanced to inhibit prompting for permission to proceed with system shutdown.	See A.2, “DCMT SHUTDOWN” on page A-6.

New r16 SP2 Feature	Reference
The DCMT DISPLAY AREA command has been enhanced to allow the display of all areas to be sorted alphabetically by area name or by page group and page range.	See A.3, “DCMT DISPLAY AREA” on page A-7.
The DCMT DISPLAY DEADLOCK and DCMT VARY DEADLOCK commands have been enhanced to allow control over whether additional information is generated for deadlocked tasks.	See A.5, “DCMT DISPLAY DEADLOCK” on page A-9 and A.15, “DCMT VARY DEADLOCK” on page A-24.
The DCMT DISPLAY SEGMENT command has been enhanced to optionally list all segments.	See A.9, “DCMT DISPLAY SEGMENT” on page A-18.
The DCMT VARY AREA and DCMT VARY SEGMENT commands have been enhanced to allow allocation or deallocation of all files associated with an area or segment.	See A.13, “DCMT VARY AREA” on page A-22 and A.26, “DCMT VARY SEGMENT” on page A-43.
The DCMT VARY DMCL command has been enhanced to inhibit prompting for permission to proceed with changes.	See A.18, “DCMT VARY DMCL” on page A-29.
The DCMT VARY REPORT command has been enhanced to allow varying of multiple reports.	See A.25, “DCMT VARY REPORT” on page A-42.
The DCMT DISPLAY ACTIVE PROGRAMS command output display has been enhanced to display the program type.	This feature is transparent in using the DCMT DISPLAY ACTIVE PROGRAMS command. To take advantage of this feature, see *Advantage CA-IDMS System Tasks and Operator Commands*.
The DCMT DISPLAY PROGRAM command has been enhanced to show MAINLINE as part of a mainline dialog's type information.	This feature is transparent in using the DCMT DISPLAY PROGRAM command. To take advantage of this feature, see *Advantage CA-IDMS System Tasks and Operator Commands*.

New for SP1:

Following are listed the new features added to Advantage CA-IDMS at r16 SP1 and references to detailed descriptions about them.

Note: Be aware that updated information about Advantage CA-IDMS r16 that is not part of the new r16 SP1 features is also found in this document. Updates and new features are both marked by vertical revision bars in the margins. However, only the updates and new features for the currently released service pack are marked.

New r16 SP1 Feature	Reference
The Advantage CA-IDMS socket interface has been enhanced with the FCNTL socket funtion to allow the specification or retrieval of a socket's timeout value.	See 9.5.3, “Associating Timeouts to Sockets” on page 9-25.
Support for the z/VM, VSE/ESA and BS2000/OSD operating systems was provided.	Any special requirements for use of a feature or the availability of a feature specific to each operating system is documented in place with each feature.

1.3 Two-Phase Commit Process

For enhanced open access, the new two-phase commit feature ensures that all changes made during recovery are either applied or backed out.

Advantage CA-IDMS Release 16.0 provides full two-phase commit capability with automatic resynchronization in the event that processing is interrupted during the two-phase commit operation.

- Two-phase commit support is provided between Advantage CA-IDMS systems so that an Advantage CA-IDMS batch or online application can safely update resources on multiple Advantage CA-IDMS systems. This ensures that all updates are either committed or rolled out.
- Advantage CA-IDMS is also able to participate in distributed transactions that are controlled by the CICS and RRS transaction managers. This enables a batch, TSO, or CICS application to coordinate Advantage CA-IDMS updates with those made through other resource managers, such as MQSeries and DB2, which support these same protocols.

1.4 SQL Features

Advantage CA-IDMS Release 16.0 includes many features for improved performance and ease of use of the Advantage™ CA-IDMS® Database SQL Option.

- The **dynamic SQL statement-caching** feature saves a copy of the SQL statement together with the result of the SQL compilation in a cache. The CPU cycles for parsing, reading the catalog and dictionary for metadata, and optimizing and creating an access plan are eliminated for subsequent executions of the same SQL statement. This feature provides a tremendous performance benefit for web applications using ODBC or JDBC access to Advantage CA-IDMS data since these open protocols are based on dynamic SQL.
- **User-defined functions** can now be defined for invocation within an SQL statement. The function can have one or more input parameters and must return a single value.
- The addition of several SQL **scalar functions** provides enhanced compatibility with Open Standards. Many of the scalar functions are implemented as user-defined functions and are automatically installed with Advantage CA-IDMS. The new scalar functions complement the existing scalar functions that were distributed with earlier releases of Advantage CA-IDMS.
- A **pseudo-column (ROWID)** is provided for unique access to a row in an SQL table. The pseudo-column is the db-key for the underlying database record. It is not persistent for the life of the database but can be used within a transaction. ROWID can be used instead of writing a table procedure for a searched update or delete where there is no primary key or foreign key.
- **Procedure enhancements** provide improved productivity. An SQL table procedure or SQL procedure can now inherit the DBNAME of the current transaction. An SQL procedure can be a mapless Advantage CA-ADS dialog that allows existing business logic to be reused in new web or distributed applications.
- Application programmers are now able to **use SQL to enhance existing non-SQL applications**. The transaction-sharing feature can be enabled to prevent deadlocks at runtime when the same database records are being updated using both SQL and native IDMS DML statements in a program or dialog. It can also prevent deadlocks between access performed within an SQL procedure and its invoking application.
- **Logical/physical separation techniques can be employed for SQL-defined databases** eliminating the need for separate schemas and access modules for each physical instance of an SQL-defined database. The specific instance that is accessed at runtime is determined by the database to which the SQL session is connected.
- **Cloning of an SQL database** provides improved productivity by allowing physically identical databases to be easily defined and maintained.
- A new **stamp synchronization utility** is provided to facilitate the movement of SQL data and definitions between Advantage CA-IDMS systems. The utility

allows users to manually synchronize the timestamps in the data area and the catalog for SQL-defined databases.

- **XML Publishing** is provided to allow XML data to be generated from data stored in an Advantage CA-IDMS database.

1.5 Administrative and Operational Enhancements

Advantage CA-IDMS Release 16.0 provides many features to improve DBA productivity including the following:

- Many database utilities that were previously only available for batch execution, such as PRINT PAGE and FORMAT, are now available for online execution.
- The DBA can define a DBNAME for utility use only in the DBNAME TABLE, thereby eliminating validation warnings for arbitrarily grouped segments.
- A new LOCK AREA utility command is available for locking an area in a batch job.
- A new parameter of ALREADY LOCKED is available on the FORMAT AREA and FIX PAGE utility statements to allow the operation to take place even if the area is locked.
- The recovery utilities report on distributed transactions and support the use of a manual recovery control file for use with the two-phase commit feature.
- A new clause on the sysgen PTERM statement allows you to define multiple terminals using a single statement.
- Execution of utility commands can now be secured at the user level.
- The batch command facility (IDMSBCF) now provides input/output reassignment to a file rather than to SYSIPT and SYSLST.
- Online compiler displays have been increased from 20,916 lines to 41,916 lines. Also, the CV node name now displays in the command line header.
- The PRINT SPACE utility now provides the capability to report on space utilization on a SUBAREA within an area.
- The EXTRACT JOURNAL, ROLLBACK, and ROLLFORWARD utilities now provide multiple segment processing.
- The IDMSLOOK utility has been enhanced to:
  - Display page reserve, SMP interval, and symbolic values when using the DMCL ALL function.
  - Display area procedures when using the SUBCHEMA and BIND SUBCHEMA functions.
- The TUNE INDEX utility has been enhanced to allow a COMMIT ALL at the end of an area to free up shared locks.
- The number of symbolics allowed in a DMCL has been increased from 32,768 to 2,147,483,648.

- The DML trace facility has been enhanced to also trace the following functions:
  - COMMIT TASK
  - COMMIT TASK ALL
  - FINISH TASK
  - ROLLBACK TASK
  - ROLLBACK TASK ALL
  - ROLLBACK TASK CONTINUE
- The DCMT VARY AREA and DCMT VARY SEGMENT commands have been enhanced to allow allocation or deallocation of all files associated with an area or segment.
- The DCMT DISPLAY AREA command has been enhanced to allow the display of all areas to be sorted alphabetically by area name or by page group and page range.
- The DCMT VARY REPORT command has been enhanced to allow varying of multiple reports.
- The DCMT DISPLAY DEADLOCK and DCMT VARY DEADLOCK commands have been enhanced to allow control over whether additional information is generated for deadlocked tasks.
- The DCMT DISPLAY ACTIVE PROGRAMS command output display has been enhanced to display the program type.
- The DCMT VARY DMCL command has been enhanced to inhibit prompting for permission to proceed with changes.
- The DCMT SHUTDOWN command has been enhanced to inhibit prompting for permission to proceed with system shutdown.
- The DCMT DISPLAY PROGRAM command has been enhanced to show MAINLINE as part of a mainline dialog's type information.
- The hexadecimal to character translation has been enhanced in all modules that perform memory snaps to use the common RHDCCODE translation tables. This allows the translation of additional characters, such as lowercase letters.
- The maximum number of pages that can be generated by line-mode terminal I/O has been increased from 999 to 32,767.
- The DCMT DISPLAY SEGMENT command has been enhanced to optionally list all segments.
- In z/OS, OS/390, z/VM, and VSE/ESA, system startup now supports additional runtime options in the execution parameter and allows them to be specified as keyword/value pairs.
- The #WTL macro has been enhanced to allow the passing of the CV node name and the Advantage CA-IDMS/DC release number.

- Additional sample TCP/IP client and generic listener server programs are provided in various programming languages in *Advantage CA-IDMS Callable Services*.

1.6 Performance Enhancements

Release 16.0 provides many features for improving performance. These include:

- Advantage CA-IDMS Release 16.0 exploits the 64-bit data addressing capabilities in z/OS V1R2 and above to utilize virtual storage above the 231 address logical line known as "the bar." The File Cache in Memory feature, activated through a new DMCL option, caches the contents of a database file in memory above the bar. This improves overall Advantage CA-IDMS performance by reducing the number of I/O operations.
- Advantage CA-IDMS Release 16.0 provides I/O performance improvements through exploitation of the Parallel Access Volume feature on Enterprise Storage System DASD devices such as IBM's Shark. This feature allows multiple jobs to simultaneously access the same logical volume. The parallel I/O operations allow higher I/O rates, thereby increasing overall throughput and reducing response time.
- A new sysgen option enables the sharing of Language Environment (LE) enclaves for improved performance for LE COBOL programs.
- New sysgen options controlling commit and rollback behavior provide faster recovery during warmstart and rollback operations and reduce the likelihood of a duplicate transaction ID when the local transaction ID values wrap.
- More efficient processing during journaling I/O operations provides overall improved throughput.
- Advantage CA-IDMS load modules may now be accessed from PDSE datasets without starting Advantage CA-IDMS as an authorized program. To load from a PDSE, you can specify an SVC number on the execute parameter in columns 29-31.
- A special form of storage protect is now available for the production system, which provides negligible processing overhead yet protects Advantage CA-IDMS and the operating system from user written code.

1.7 Non-Stop Processing Features

Several features are available in Release 16.0 to provide enhanced availability of Advantage CA-IDMS in a 24 x 7 environment. These include the ability to:

- Dynamically turn on/off tracing as well as the ability to vary the size of the trace tables
- Dynamically vary any program attribute using the DCMT task
- Write a message to the console when a "Short on Storage" condition occurs so that corrective action can be taken

1.8 Tool Product Enhancements

Many tool product enhancements are implemented as part of Release 16.0, including:

- Advantage™ CA-IDMS® Database DML Online Option

  Advantage CA-IDMS DMLO Release 16.0 incorporates many enhancement requests including:

  - Enhanced DMLO entry screen display to indicate which interrupt key is used to exit DMLO.
  - HLPDICT now defaults to the current working dictionary if no setting is specified in the Installation Parameter module USDTPARM when used in the Advantage CA-IDMS/DC environment. Previously this defaulted to TOOLDICT.
  - The User Exit Program USDMLXIT can dynamically pass back a message for subsequent display on the DMLO command line. Previously only a numeric return code was passed back. This provides the user with the capability to dynamically alter message text.

- Advantage™ CA-IDMS® Database Dictionary Module Editor (DME) Option

  The following enhancements are available in Advantage CA-IDMS DME Release 16.0:

  - A new "fast in" installation parameter is provided for direct invocation of the Module Edit Screen.
  - Any value (including nulls) can now be specified for the DME Print Class.
  - If any compile errors occur when using the Advantage CA-ADS ADSC compiler, the user is presented with an edit browse screen that illustrates and highlights the lines in error.

- Advantage™ CA-ADS®/Alive Option

  The following enhancements are provided with Advantage CA-ADS Alive Release 16.0:

  - The maximum number of records that can be processed per Advantage CA-ADS dialog is increased to 200.
  - An installation parameter option is available to disable the Post Abort browse screen feature. The abend details continue to be written to the DEBUGQUEUE.

- Advantage™ CA-IDMS® Database Journal Analyzer Option

  Advantage CA-IDMS Journal Analyzer Release 16.0 delivers:

  - Support for the new journal records and layouts for Advantage CA-IDMS 16.0.
  - Enhanced RECORD and DBKEY DISPLAY processing to allow for the addition of START and STOP dates when ALL=Y is indicated on the

PROCESS statement. This allows the user to create RECORD and DBKEY displays for a particular time period.

- Advantage™ CA-Culprit for CA-IDMS®

  The use of Advantage CA-Culprit parameters stored in AllFusion™ CA-Librarian® or AllFusion™ CA-Panvalet® libraries no longer requires that the AllFusion CA-Librarian or AllFusion CA-Panvalet file access routines be linked with Advantage CA-Culprit routines to form the respective interfaces. These interfaces are now dynamically loaded.

1.9 TCP/IP API Support

The TCP/IP feature provides support for the development and execution of client/server Advantage™ CA-IDMS®/DC Transaction Server applications that use the industry-standard TCP/IP communications protocol. A generic listener function and callable sockets API allow client and server programs written in Advantage CA-ADS, COBOL, PL/I or Assembler to communicate through TCP/IP with programs running on the same or different platforms.

1.10 Type 4 JDBC Driver

A built-in TCP/IP server program is provided that enables the Advantage CA-IDMS Server JDBC driver to function as a "Type 4" driver. This allows client applications written in Java to communicate directly with the Advantage CA-IDMS address space using the native "wire" protocol, with no intervening middleware.

1.11 Upgrading to Release 16.0

In general, to install Release 16.0, follow the instructions documented in the Advantage CA-IDMS installation manual for your operating system.

Before starting the installation, carefully read Chapter 2, “Upgrading to Release 16.0.” This helps to ensure that you are successful in your use of Advantage CA-IDMS 16.0 and are able to fall back to a previous release of Advantage CA-IDMS, if necessary.

Chapter 2. Upgrading to Release 16.0

2.1 Overview

This chapter describes the actions that must be taken and the considerations involved in upgrading to Release 16.0 of the Advantage CA-IDMS family of products. You can upgrade to Release 16.0 from Advantage CA-IDMS Release 10.x, 12.0, 14.0, 14.1, or 15.0. The conversion utilities provided for Releases 12.0, 14.0, 14.1, and 15.0 are included on the Release 16.0 installation tape.

This is a summary of actions required to update the Advantage CA-IDMS software to Release 16.0:

- Install the software into a new environment.
- Install the new SVC delivered with Release 16.0.
- Initialize the journal files using the Release 16.0 FORMAT utility before starting a Release 16.0 system for the first time.
- Offload the log file using a pre-Release 16.0 ARCHIVE LOG utility or initialize the log file before starting a Release 16.0 system for the first time.
- For all Advantage CA-IDMS systems using the cloned system facility to share the system definition with another system, add the SYSIDMS DCNAME parameter to the startup JCL.
- Run IDMSDIRL against each dictionary containing the IDMSNTWK schema definition.
- Update the Advantage CA-IDMS task and program definitions using the source members provided on the installation tape and the sysgen compiler.
- Customers employing dynamic routing of database connections may need to define additional destinations to the system's resource table.
- In a data sharing environment, where the queue area is shared between group members, all sharing systems must be upgraded to Release 16.0 simultaneously or sharing of the queue between Release 16.0 and pre-Release 16.0 systems must be disabled.
- For clients running z/OS 1.2 or later, if your Advantage CA-IDMS database files are cached in dataspaces or you intend to exploit the 64-bit memory architecture for file caching, it may be necessary to reassign initiator classes for Advantage CA-IDMS systems and local mode batch jobs.

  Warning: Advantage CA-IDMS and CICS cannot run in the same initiator class.
- z/OS and OS/390 customers using the CMS Option no longer need to set optional APAR bit 236 to activate CMS support.
- Advantage CA-IDMS SQL customers and users of Advantage CA-IDMS Visual DBA should:
  - Update the CA-supplied SYSCA schema definition using the command facility and the source members provided on the installation tape.

  - Execute the CONVERT CATALOG command against each SQL-enabled dictionary.

- Apply APARs to all prior versions of Advantage CA-IDMS that access or are accessed by Release 16.0 software. This includes access from an external teleprocessing monitor such as CICS.
- Clients using CICS must create new IDMSINTC and IDMSINTL interface modules before using Release 16.0 runtime libraries in their CICS systems.
- Clients using the IDMSINTC CICS interface:
  - May need to change TPNAME parameters or specify a new CICS_NAME SYSIDMS parameter to ensure that every CICS system has a consistent and unique identifier.
  - If you use the auto-commit feature so your Advantage CA-IDMS database transactions can be committed through a CICS syncpoint operation, you must take additional steps to implement two-phase commit support between CICS and Advantage CA-IDMS.

    Note: This requirement also applies to Advantage CA-IDMS Transparency for VSAM users.
  - Must define a new resynchronization task and program.
- Recompile all user-written programs that reference Advantage CA-IDMS control blocks or journal files.
- Review the recovery and restart procedures for applications:
  - Issuing remote database requests between Advantage CA-IDMS systems.
  - Using IDMSINTC with the auto-commit feature enabled.
- z/OS, OS/390, z/VM, and VSE/ESA users may need to alter the way in which they create their startup modules or specify additional startup parameters in their execution JCL.

2.2 Installing the Software

Follow the instructions documented in the Advantage CA-IDMS installation manual for your operating system. Also, follow any special installation instructions outlined in the cover letter delivered with the installation tape. Be sure to install the Release 16.0 software into a new set of installation libraries. You cannot install Release 16.0 into an existing Advantage CA-IDMS environment.

2.3 Installing the SVC

A new SVC is delivered with Release 16.0. It should be used for all Release 16.0 systems. The SVC is downward compatible and can be used with Release 14.1 and Release 15.0 systems. If you are sharing an SVC with multiple releases of Advantage CA-IDMS, please refer to Chapter 8 of the *Advantage CA-IDMS Installation and Maintenance - z/OS and OS/390* manual to ensure that you are specifying the correct parameters to CAIRIM. This ensures that all releases of Advantage CA-IDMS that are using the SVC are identified.

2.4 Formatting Journal Files

Several journal records are changed in Release 16.0. It is necessary to initialize the journal files using the Release 16.0 FORMAT utility statement before the journal files are used with a Release 16.0 system. At startup, the system verifies the journal files are correctly formatted. If the files are not properly formatted a DC202037 message is issued:

```
IDMSWARM – Journals not formatted correctly for the current release of IDMS
```

This is followed by another informational message and then a 3033 ABEND occurs.

If it is necessary to fall back to an earlier release of the software, the journal files must be reinitialized using the FORMAT utility and runtime libraries from the earlier release, otherwise warmstart fails.

2.5 Offloading the Log File

The format of the log file's statistics records is unchanged in Release 16.0, although the release identifier in these records is updated and contains the string 'R160'. If Advantage CA-IDMS encounters a log record with an earlier release identifier, the ARCHIVE LOG utility issues the warning message:

```
NON 16.0 RECORD HAS BEEN ENCOUNTERED IN THE LOG, RECORD WILL BE BYPASSED
```

To avoid these messages and to separate logs from prior releases, offload the log file using the ARCHIVE LOG utility before installing Release 16.0 or initialize the log file if you do not need the log information.

If it is necessary to fall back to an earlier release of the software, any log files that are accessed by a Release 16.0 system must be offloaded or initialized prior to its use by a pre-Release 16.0 system.

2.6 Specifying a DCNAME for Cloned Systems

If an Advantage CA-IDMS system relies on the cloned system capability to share its system definition with another Advantage CA-IDMS system, the SYSIDMS file in its startup JCL must include a DCNAME parameter in order to uniquely identify the system that is being started. For more information, see 3.5, “Impact on System Operations” on page 3-18.

2.7 Updating Dictionary Descriptions

New fields were added to records in the DDLDML and DDLCAT areas using existing filler space. Although no dictionary conversion is necessary, you should update the definition of these records in every dictionary containing the IDMSNTWK schema description. To do this, use the IDMSDIRL utility. For instructions on executing this utility, refer to the *Advantage CA-IDMS Utilities* manual.

2.8 Updating Task and Program Definitions

There are new CA-supplied task and program definitions for Release 16.0. You should update the system definition using the batch system generation compiler, RHDCSGEN, and source members provided on the installation tape. This can be accomplished easily by:

1. Performing an UPGRADE install to upgrade the definitions for SYSTEM 99.
2. Copying the task and program definitions from SYSTEM 99 to your system definition.

For more information on the UPGRADE install process, see the Advantage CA-IDMS installation manual for your operating system.

If it is necessary to fall back to an earlier release of the software, you can recreate the earlier versions of the task and program definitions by reinstalling them from the installation tape provided with the earlier release or by restoring the system dictionary from a backup that was taken prior to the migration. If returning to Release 15.0 of Advantage CA-IDMS, it is not necessary to restore the earlier versions of the task and program definitions.

2.9 Defining Destination Resources

If dynamic routing is used within a parallel sysplex environment to balance a workload across multiple members of a DBGROUP, it may be necessary to add the group members as destinations within a front-end system's resource table so that startup can resynchronize with these systems. This must be done only for group members that are not defined to the system through NODE statements. For more information see 3.4, “Impact on System Definition” on page 3-16.

2.10 Disabling Queue Area Sharing

A Release 16.0 system cannot share its queue with a pre-Release 16.0 system. Clients that are sharing the queue area must either:

- Upgrade all sharing systems within a data sharing group at the same time.
- Disable sharing of the queue between Release 16.0 and pre-Release 16.0 members.

2.11 Reassigning Initiator Classes

If running a z/OS 1.2 or later operating system, it may be necessary to reassign initiator classes for central versions or local mode batch jobs that cache database files in memory or in a dataspace. Under these circumstances, Release 16.0 allocates 64-bit storage. Since 64-bit storage acquisition is incompatible with subspaces, you must ensure that the same address space is not used for both Advantage CA-IDMS and applications, such as CICS, that use subspaces. For more information, see 6.2, “File Cache in Memory” on page 6-4.

2.12 Activating the CMS Option

When running a Release16.0 system, z/OS and OS/390 customers using the CMS Option are not required to set optional APAR bit 236 to enable CMS support for the Advantage CA-IDMS system. Activation of the CMS Option is now automatic for each system using an SVC for which the CMS option is enabled

2.13 Updating Advantage CA-IDMS SQL

2.13.1 Updating SYSCA Schema Definitions

Advantage CA-IDMS Visual DBA and SQL users should update their SYSCA schema definitions in each catalog in which the SYSCA schema is defined. This process varies slightly depending on your release of Advantage CA-IDMS. Details of the required steps can be found in the installation materials received with your Release 16.0 installation tape.

The following changes have been made to the SYSCA schema for Release 16.0 and are downward compatible with prior releases of Advantage CA-IDMS:

- SYSCA.ACCESSIBLE_TABLES view is updated and excludes functions.
- SYSCA.TABLES view is updated and excludes functions.
- The following new views are defined:
  - SYSCA.ACCESSIBLE_PROCS
  - SYSCA.ACCESSIBLE_FUNCS
  - SYSCA.DSCCACHEV
- The following new table procedures are defined:
  - SYSCA.DSCCACHEOPT
  - SYSCA.DSCCACHECTRL
  - SYSCA.DSCCACHE
- The following new functions are defined:
  - SYSCA.ABS
  - SYSCA.ACOS
  - SYSCA.ASIN
  - SYSCA.ATAN
  - SYSCA.ATAN2
  - SYSCA.CEIL
  - SYSCA.CEILING
  - SYSCA.COS
  - SYSCA.COSH
  - SYSCA.COT
  - SYSCA.DEGREES
  - SYSCA.EXP

- SYSCA.FLOOR
- SYSCA.LOG
- SYSCA.LOG10
- SYSCA.MOD
- SYSCA.PI
- SYSCA.POWER
- SYSCA.RADIANS
- SYSCA.RAND
- SYSCA.ROUND
- SYSCA.SIGN
- SYSCA.SIN
- SYSCA.SINH
- SYSCA.SQRT
- SYSCA.TAN
- SYSCA.TANH
- SYSCA.TRUNCATE
- SYSCA.INSERT
- SYSCA.REPEAT
- SYSCA.REPLACE
- SYSCA.RIGHT
- SYSCA.SPACE
- SYSCA.DAYNAME
- SYSCA.DAYOFWEEK
- SYSCA.DAYOFYEAR
- SYSCA.MONTHNAME
- SYSCA.QUARTER
- SYSCA.WEEK

2.13.2 Converting SQL Catalogs

Advantage CA-IDMS Visual DBA and SQL users must use the CONVERT CATALOG command to update the definitions of system tables in each catalog in which the SYSTEM schema is defined.

The converted definitions are compatible with these Releases:

- 14.0

- 14.1
- 15.0

Warning: If you are upgrading from Release 12.0 or 12.01, you should retain backup files of the catalog.

If it is necessary to fall back to Release 14.0, 14.1, or 15.0 version of the software, no special action needs to be taken regarding the catalog; however, if falling back to Release 12.0 or 12.01, the catalog (and database areas containing tables that are created or altered using Release 16.0) must be restored.

2.13.2.1 Release 16.0 Changes

When a catalog is converted, the definitions of the following tables are upgraded to their Release 16.0 definitions and new columns in associated rows are initialized appropriately:

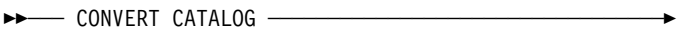
- SYSTEM.SCHEMA
- SYSTEM.TABLE
- SYSTEM.DBNAME

Changes introduced in earlier releases of the software are applied if they have not already been made. Refer to the *Advantage CA-IDMS Features Guide — Release 15.0* for a description of these prior changes.

2.13.2.2 Executing the Catalog Conversion Utility

The catalog conversion utility can be invoked using the online command facility (OCF) or the batch command facility (IDMSBCF). If running in local mode or if converting from Release 12.0 or 12.01, you should back up the target catalog before executing this utility.

To convert a catalog enter the following statement:



After successful execution, the Command Facility issues one of two informational messages to indicate the status of the conversion.

If a catalog conversion is performed, the message indicates the number of rows of each type that are changed. If a catalog conversion is not required, an appropriate message is issued.

2.14 Applying an APAR to Earlier Releases

If a Release 16.0 system is accessed by a 14.1 or 15.0 version of Advantage CA-IDMS or vice-versa, you must apply one of these APARs to the pre-Release 16.0 system:

- 14.1 — QO23507
- 15.0 — QO23506

Communications with releases earlier than 14.1 are not supported.

2.15 Updating the CICS Interfaces

2.15.1 Creating New CICS Interface Modules

Before a CICS system can use the Release 16.0 Advantage CA-IDMS runtime library, you must create new IDMSINTC and IDMSINTL interface modules and UCF front-ends (if applicable) using the Release 16.0 source and object libraries. There is no need to create new IDMSCINT or IDMSCINL modules or relink user applications when upgrading to an Advantage CA-IDMS Release 16.0 system.

2.15.2 Identifying a CICS System

Users of the IDMSINTC interface must ensure that each CICS system has a consistent and unique identifier. By default, the name of a CICS system is determined by the TPNAME parameter of the CICSOPT options table. If more than one IDMSINTC interface is used within a CICS system they must all have the same TPNAME value in their CICSOPT assembly or you must specify a new CICS_NAME parameter in the SYSIDMS file included in the CICS startup JCL. The name given to a CICS system must be unique across all CICS systems accessing any one central version.

For more information on specifying a name for a CICS system, refer to 3.8, “Two-Phase Commit Support with CICS” on page 3-30.

2.15.3 Implementing Two-Phase Commit Support in CICS

In Release 16.0, a two-phase commit protocol is used whenever AUTOCMT is enabled for an IDMSINTC CICS interface. The use of AUTOCMT causes a CICS syncpoint operation to commit changes made by database sessions that are still active when the syncpoint is taken. In order to support two-phase commit processing between CICS and Advantage CA-IDMS, additional installation steps must be taken. These steps are discussed in 3.8, “Two-Phase Commit Support with CICS” on page 3-30.

Advantage CA-IDMS Transparency for VSAM forces the use of AUTOCMT in CICS. Therefore users of this product must also perform these extra installation steps.

2.16 Recompiling User-Written Programs

Several control block formats are changed in Release 16.0. Although in most cases the changes simply entail the addition of new fields, it is recommended that all programs referencing Advantage CA-IDMS control blocks, such as user-written exits, be recompiled using the Release 16.0 library.

2.17 Creating a System Startup Module

For z/OS, OS/390, z/VM, and VSE/ESA users, the linkedit parameters used to create the startup module for an Advantage CA-IDMS system may have changed depending on your operating system and how you created these modules in the past. See *Advantage CA-IDMS System Operations* for instructions on how to create a startup module appropriate to your environment.

The IDMSDC module that is created during the installation process no longer includes an RHDCPARM module. If you use the installed IDMSDC module as your system startup module, you likely will need to specify additional startup parameters at execution time. See the cover letter for your installation tape for more details and 5.16, “System Startup Enhancements” on page 5-33 for a description of the enhanced startup parameters.

Chapter 3. Two-Phase Commit Support

3.1 Overview

This chapter discusses the two-phase commit support provided in Advantage CA-IDMS Release 16.0.

3.2 Two-Phase Commit Protocol

Two-phase commit is a protocol used to ensure that all changes made within the scope of a distributed unit of work are either applied (committed) or backed out.

As the name implies, a two-phase commit process is divided into two phases. In the first phase, resource managers participating in the unit of recovery prepare their resources to be committed. If they cannot do so, they inform the requestor of the failure. In the second phase, the resource managers either make their changes permanent or back them out based on the overall outcome of the transaction.

If a resource manager indicates that it has successfully prepared its resources to be committed, it guarantees that the resources can be committed even if some adverse condition, such as a system failure, occurs prior to completion of the commit process. It is this guarantee that ensures that all changes are either applied or backed out in their entirety.

3.2.1 Terminology

The following terms are associated with two-phase commit processing:

A **resource manager** is a software component that controls access to and the state of one or more recoverable resources such as a database. An Advantage CA-IDMS central version is an example of a resource manager.

A **transaction manager** is a software component that directs commit and backout processes. Multiple transaction managers may be involved in a single commit or backout operation. If so, their actions are coordinated to achieve transaction consistency. Every Advantage CA-IDMS system has a transaction manager as a component.

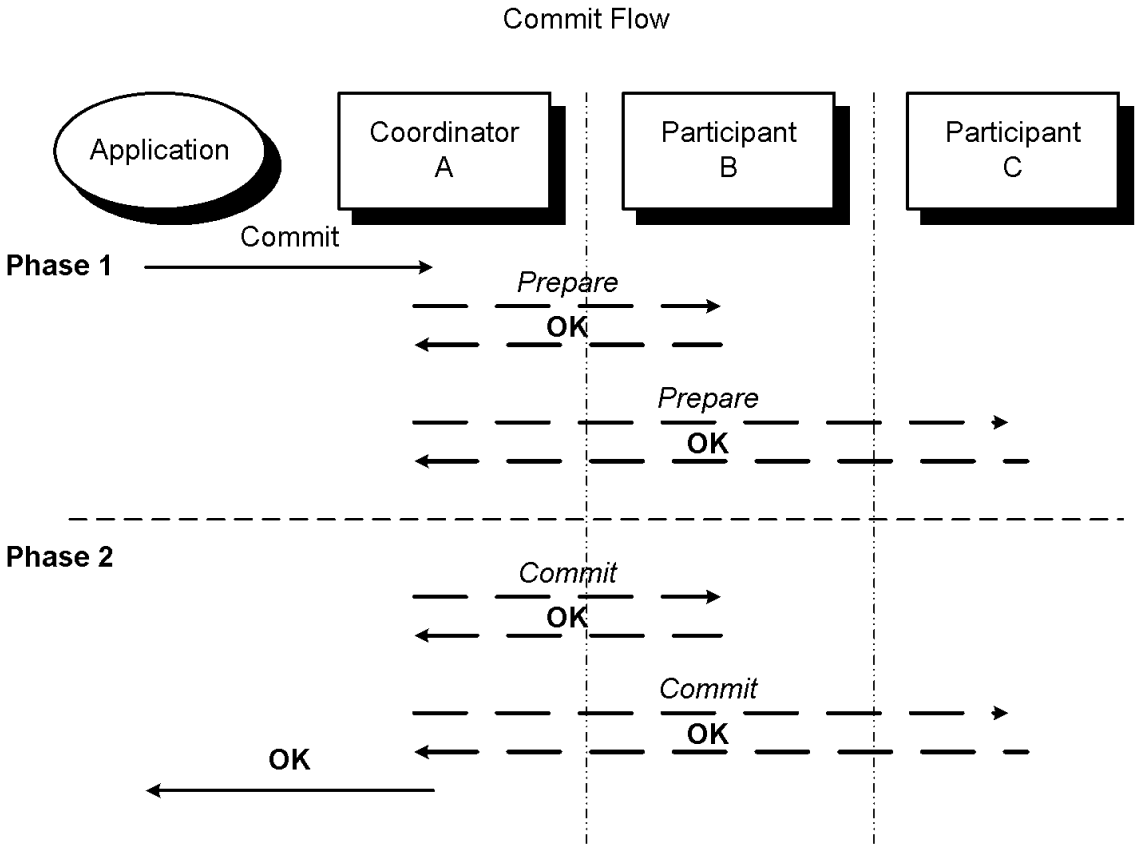
A **coordinator** is a transaction manager that initiates a two-phase commit operation and is responsible for its overall outcome. A coordinator is sometimes referred to as an initiator.

A **participant** is a resource manager or a transaction manager other than the coordinator that participates in a two-phase commit operation. A participant is sometimes referred to as an agent.

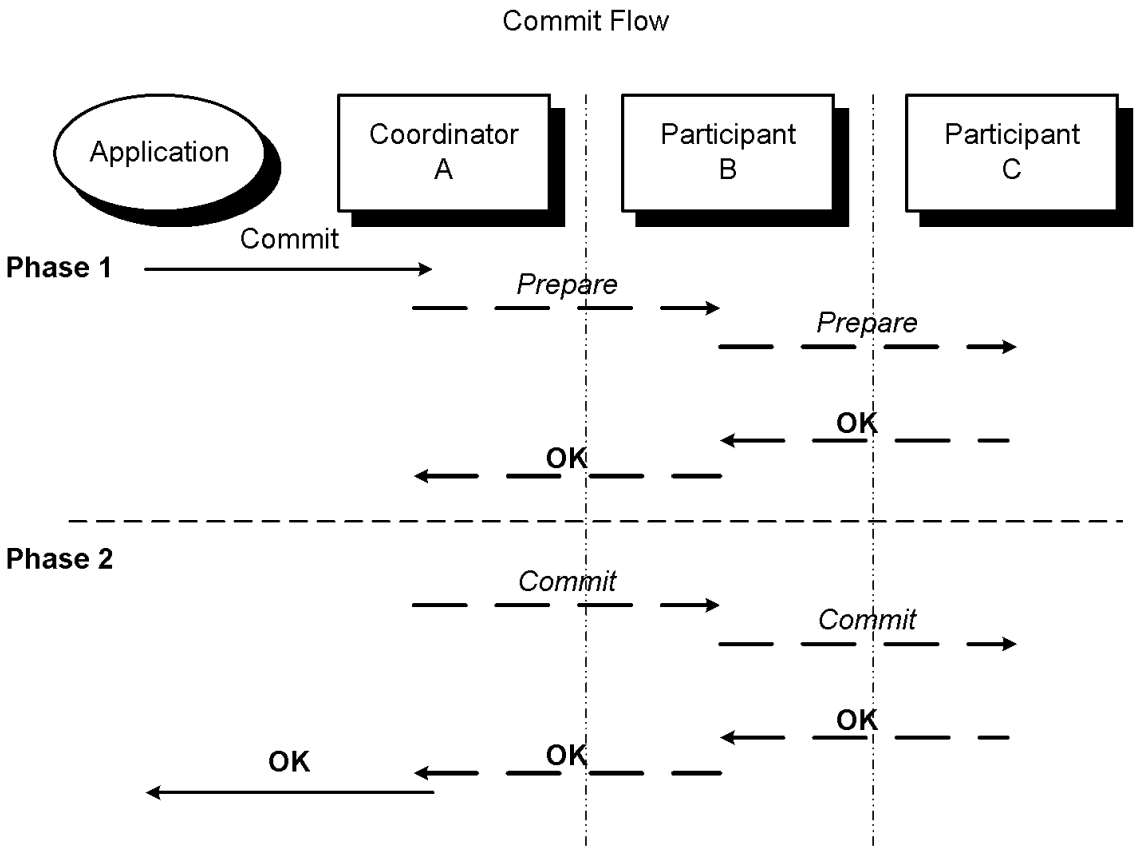
A **distributed transaction** is a unit of recovery in which more than one resource manager participates.

3.2.2 Typical Commit Flows

The following diagram illustrates the communications that take place during a typical two-phase commit operation involving three systems. In this example, A is the coordinator since it initiates the commit operation, and B and C are participants.



The following diagram illustrates another typical commit flow. In this example, A is again the overall transaction coordinator, and B and C are participants. However, in this case B plays a dual role. It is both a participant with respect to A and a coordinator with respect to C since it forwards the *Prepare* and *Commit* directives that it receives from A to C. Such a situation might arise because an application on A starts a remote SQL transaction on B that, in turn, updates resources on C through an SQL procedure.



3.2.3 Prepare and Commit Outcomes

When a participant receives a *Prepare* request, it does whatever is necessary to guarantee that a subsequent *Commit* request can be honored. This may involve such things as flushing buffers or forwarding requests to other participants. If all of these activities are completed successfully, the participant signals its willingness to commit by responding OK to the *Prepare* request. If it is unable to successfully complete its preparations, it indicates this by responding BACKOUT to the *Prepare* request.

The coordinator gathers the responses from its participants and determines the final outcome for the commit operation. If all participants indicate that they are willing to commit, then the coordinator proceeds with the second phase and the final outcome will be OK. If any participant indicates that it cannot commit, then the coordinator directs its participants to back out their changes instead of committing them. The final commit outcome in this case is BACKOUT.

A participant can respond to a *Prepare* request in ways other than OK or BACKOUT. It can respond FORGET to signal that it made no updates within the transaction being committed and does not need to participate in the second phase. This has the potential for reducing the number of communications needed to complete the commit operation.

A participant can also respond "heuristically," indicating that its resources have been committed or backed out. A transaction might be completed heuristically because it was forced to complete through some administrative action. Such heuristic actions defeat the two-phase commit process and can lead to mixed outcomes in which some changes are committed while others are backed out.

Note: While Advantage CA-IDMS does not make heuristic decisions on its own, it does allow an administrator to commit or backout a transaction using a DCMT command. Such administrator intervention might be required following an interruption in the commit process.

3.2.4 Recovery from Failure

Failures in communications, operating systems, or resource or transaction managers can interrupt the two-phase commit process. The point at which the failure occurs determines whether a transaction's changes are committed or backed out. If the failure occurs during the first phase in the process, changes are backed out. If the failure occurs during the second phase, changes are committed.

Recovery from failure during a two-phase commit involves a process called resynchronization, in which messages are exchanged between a coordinator and a participant in order to complete transactions whose commit process was interrupted. To facilitate resynchronization, both the coordinator and the participant write additional journal records at critical points during the two-phase commit process.

3.3 Two-Phase Commit Support Within Advantage CA-IDMS

With Release 16.0, a central version always uses a two-phase commit protocol to commit resources. The DML commands that an application issues to commit tasks and database transactions (for example, FINISH TASK or COMMIT WORK) now follow a two-phase commit protocol. When one of these commands is issued, the Advantage CA-IDMS system to which the command is directed becomes the coordinator. Any other Advantage CA-IDMS system involved in the transaction becomes a participant.

From a programming perspective, the semantics of these commands have not changed since prior releases always attempted to commit all updates made within a distributed transaction. However, because a two-phase commit protocol was not used in prior releases, it was possible for some changes to be committed while others were backed out. Release 16.0 eliminates this potential by always following a two-phase commit protocol.

3.3.1 Optimizations Supported

To minimize the cost of doing a two-phase commit operation, Advantage CA-IDMS supports the Read Only, Single Agent, and Presumed Abort optimizations.

The **Read Only** optimization reduces the communications needed to commit a distributed transaction. A participant that has not updated resources within the scope of the transaction can respond FORGET to a *Prepare* request. Advantage CA-IDMS does not include such read-only participants in the second phase of the commit operation, thus eliminating at least one communication. Additionally, the read-only participant writes no journal records in support of the two-phase commit operation.

Advantage CA-IDMS uses the **Single Agent** optimization to reduce the flows needed to commit a distributed transaction. At the point when a *Prepare* request is to be sent to the last remaining participant, if all other participants have responded FORGET or if this is the only participant in the transaction, then a *OnePhaseCommit* request is sent instead of a *Prepare*. This results in only a single communication with the participant to complete the commit operation. Furthermore, if there is only a single participant, the coordinator writes no journal records in support of the distributed transaction.

Advantage CA-IDMS uses a **Presumed Abort** protocol to reduce journaling overhead. Simply put, this means that while a coordinator retains knowledge of a committed transaction until all of its participants indicate that they have completed the second phase of the commit operation, the coordinator can immediately forget transactions whose outcome is BACKOUT. Consequently, no journaling activity for a distributed transaction takes place at a coordinator until all *Prepare* votes have been collected and then only if the outcome is OK. The absence of knowledge of a transaction signifies that its outcome is BACKOUT.

The alternative to Presumed Abort is **Presumed Nothing**. Under this protocol a coordinator retains knowledge of the outcome of a commit operation until all participants indicate that it can be forgotten, regardless of whether the final outcome is

OK or BACKOUT. Consequently, a coordinator must journal the existence of a transaction prior to forwarding the first *Prepare* request, and it must retain knowledge of backed out transactions longer. Advantage CA-IDMS does not support the Presumed Nothing protocol.

3.3.2 Support for External Coordinators

Release 16.0 uses a two-phase commit protocol internally to commit its own resources, and it can participate in a two-phase commit operation controlled by the following external coordinators:

- CICS Transaction Server
- RRS — IBM's system-level resource recovery platform for z/OS and OS/390
- XA transaction managers supported by future releases of Advantage™ CA-IDMS® Database Server Option

By participating in externally controlled transactions, updates to Advantage CA-IDMS resources can safely be coordinated with those of other resource managers that are supported by the above transaction managers.

For more information on:

- CICS Transaction Server — see 3.8, “Two-Phase Commit Support with CICS” on page 3-30
- RRS — see 3.9, “Two-Phase Commit Support with RRS” on page 3-36

Information on XA support will be provided in the future.

3.3.3 Support for External Resource Managers

Release 16.0 can coordinate transactions in which external resource managers are participants. It does this in one of two ways: by enlisting the services of RRS or by using a resource manager interface tailored to both the Advantage CA-IDMS environment and the external resource manager.

If the external resource manager supports RRS as a coordinator, using RRS as an intermediary is the easiest way to extend two-phase commit support to the external resource manager. In this way, any resource manager that supports RRS as a coordinator can potentially participate in an Advantage CA-IDMS-controlled transaction.

If the resource manager does not support RRS as a coordinator, then an interface that is tailored to the external resource manager and that supports the Advantage CA-IDMS transaction manager protocol can be used to enable the resource manager to be a direct participant in an Advantage CA-IDMS-controlled transaction.

Computer Associates will work with third-party vendors that provide online access from Advantage CA-IDMS DC/UCF to resource managers such as DB2 and MQ-Series to extend full two-phase commit participation to these products.

3.3.4 Support for Pre-Release 16.0 Systems

Pre-Release 16.0 systems do not support a two-phase commit protocol; however, they can participate in a two-phase commit operation.

Pre-Release 16.0 systems support a one-phase commit protocol only. Hence, when they are participants in a two-phase commit operation, they are sent *OnePhaseCommit* requests, rather than separate *Prepare* and *Commit* requests. If only a single pre-Release 16.0 system participates in the transaction, it is treated as a "last agent," meaning that all other participants are sent *Prepare* requests before the pre-Release 16.0 system is sent its *OnePhaseCommit* request. If this latter request is successful, then the commit operation proceeds to a successful conclusion; otherwise, the transaction is backed out. If more than one pre-Release 16.0 system participates in the transaction, each one is sent a *OnePhaseCommit* request during the second phase of the commit operation. Of course, this can result in some changes being committed while other changes are backed out; however, this potential is eliminated once all systems are converted to Release 16.0 and is no riskier than if all systems were pre-Release 16.0 systems.

If a Release 16.0 system participates in a transaction that is controlled by a pre-Release 16.0 system, then it is sent the same commit directives that a pre-Release 16.0 participant is sent. The Release 16.0 system treats these requests as if they came from an application and uses a two-phase commit protocol to commit both local resources and those updated by remote participants of its own.

3.3.5 Support for Batch Applications

All changes made by a batch application are committed or backed out as a single unit provided at least one of the following is true:

- All updates are made through a single transaction.
- All updates are made through transactions executing on a single central version and a task-level commit request is issued.
- Batch RRS support is enabled and all database sessions started by the batch application are routed to central versions running within the same operating system image as the batch application.

If these conditions are not present, then commit support functions effectively the same as it would in a pre-Release 16.0 batch environment.

3.3.6 Implementation Details

This section provides details on certain aspects of the Advantage CA-IDMS two-phase commit implementation. While knowledge of this material is not required to use two-phase commit, it facilitates understanding of the output from recovery utilities and DCMT commands and might prove useful in researching exceptional recovery situations.

3.3.6.1 Transaction Branches

A transaction branch represents a separately identifiable portion of a transaction within which deadlocks cannot occur. Unless transaction sharing is in effect, every database session (every run unit or SQL database session) is associated with a separate transaction branch. When transaction sharing is in effect, multiple database sessions may share a single transaction branch. In so doing, they avoid deadlocking among themselves, since deadlocks are not possible for work performed under a single transaction branch.

An application is associated with multiple transaction branches if it opens concurrent, non-sharing database sessions. Multiple branches can also result from the use of system services that access a dictionary, such as loading from a load area or accessing a queue area. If more than one transaction branch exists, they are organized hierarchically, meaning that there is a single top-level branch and one or more subordinate branches. The top-level branch represents either the work done by a database session or all work done by a task (or user session if no task is active). A subordinate branch always represents the work done by a database session. A subordinate branch may in turn have subordinate branches of its own, perhaps as a result of an SQL procedure that opens its own database session.

Every transaction branch is assigned a unique identifier that never changes. This **Branch Identifier** (**BID**) is an eight-byte hexadecimal value that is sometimes qualified by the node name of the local system to make it a globally unique value.

A commit operation is always targeted to a single transaction branch and encompasses all of that branch's subordinates. The target branch becomes the **top-level branch** of the transaction and its subordinates become the **subordinate branches** of the transaction. If a task-level commit operation is initiated, the target branch is always the top-level branch in the task's branch hierarchy. If a database session-level commit operation is initiated, the target branch is the one associated with the database session through which the commit request is issued.

3.3.6.2 Transaction Identifiers

Transactions can have multiple identifiers. Advantage CA-IDMS assigns two types of identifiers: a local transaction identifier and a distributed transaction identifier. External transaction managers may assign transaction identifiers of their own, generically referred to as external transaction identifiers.

A **Local Transaction Identifier** (**LID**) is a four-byte value that identifies the work done by a branch within a transaction. It is used to distinguish the work done by one branch from that of another and is recorded in the journal records that are used to track local database changes (for example, BGIN, BFOR, AFTR). Local transaction identifiers are unique only within a central version.

A **Distributed Transaction Identifier** (**DTRID**) is a 16-byte value that uniquely identifies a distributed transaction across all participating nodes. It is assigned by the Advantage CA-IDMS system that is acting as the coordinator for the transaction or by

a CICS interface. Every distributed transaction processed by an Advantage CA-IDMS system is assigned a DTRID, regardless of whether the transaction also has externally assigned identifiers. The DTRID is recorded in the distributed transaction journal records that are written during the two-phase commit process (for example, DIND, DCOM, DFGT).

A DTRID value is comprised of an 8-character prefix followed by an 8-byte hexadecimal value. If assigned by an Advantage CA-IDMS system, the prefix is the system's node name and the suffix is an 8-byte internal format timestamp.

If the DTRID is assigned by a CICS interface, the 8-character prefix consists of "CICS" concatenated with the 4-character CICS system identifier specified in the TPNAME parameter of the interface's CICSOPT macro. The 8-byte hexadecimal value is the UOW (Unit Of Work) identifier assigned by CICS to the work unit being committed.

External transaction managers may also assign their own identifiers to a distributed transaction in which Advantage CA-IDMS is a participant. The following types of external identifiers are recognized by Advantage CA-IDMS and are recorded in the distributed transaction journal records written by the central version that interfaces directly with the external transaction manager. These journal entries provide a cross reference between the internal and external identifiers.

- **RRS URID** — the Unit of Recovery Identifier (URID) assigned by RRS. A URID is a 16-byte hexadecimal value.
- **XA XID** — the transaction identifier assigned by an XA transaction manager. An XID is a hexadecimal value whose length can be up to 140-bytes.

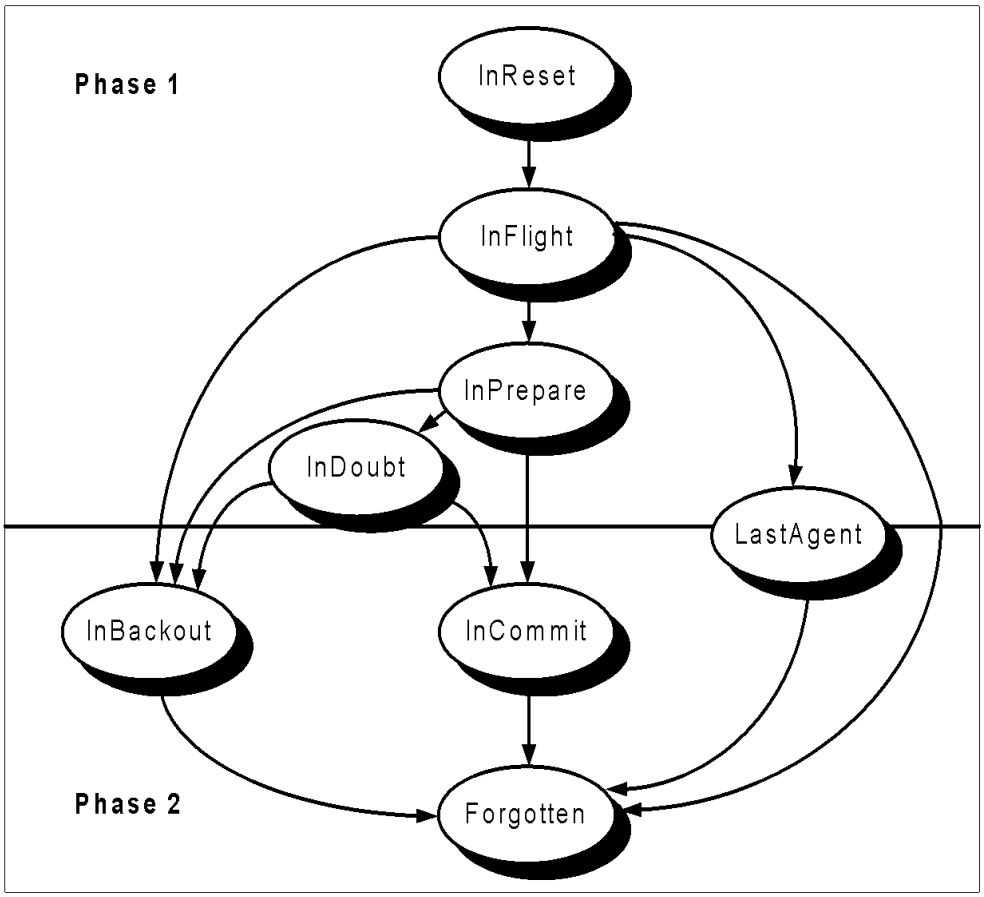
3.3.6.3 Transaction States

Transaction state is an attribute of a distributed transaction that reflects its progress through a two-phase commit operation. The Advantage CA-IDMS transaction manager assigns the following transaction states for this purpose:

- InReset — This is the initial state prior to the start of a commit or backout operation.
- InFlight — This state is assigned at the start of a two-phase commit operation and persists while the transaction manager is assessing the need for and the ability to proceed with the two-phase commit operation.
- InPrepare — This state is assigned when the transaction manager determines that a two-phase protocol is needed to guarantee the integrity of a commit operation.
- LastAgent — This state is assigned by a coordinator's transaction manager when there is only a single participant and consequently a full two-phase protocol is not needed to guarantee the integrity of a commit operation.
- InDoubt — This state is assigned by a participant's transaction manager when it writes a DIND journal record for the transaction.

- InCommit — This state is assigned when a DCOM journal record is written for the transaction.
- InBackout — This state is assigned when it is determined that the outcome of the distributed transaction is BACKOUT.
- Forgotten — This state is assigned when the two-phase commit operation is complete. The following diagram illustrates the transitions that can occur from one state to another as a transaction proceeds through a two-phase commit operation.

Transaction States



3.3.6.4 Transaction Outcomes

Fundamentally, a distributed transaction can have only one of the following three outcomes: all changes were committed, all changes were backed out, or some changes were committed while others were backed out. However, it is useful to support variations of these basic three outcomes, especially as interim results.

Advantage CA-IDMS recognizes the following transaction outcomes:

- OK — The request is complete and the transaction's changes have been committed.
- FORGET — The request is complete, but no changes were committed since none were made (that is, this is a read-only transaction).
- OK_PENDING — The request is not yet complete, but changes have been or will be committed.
- BACKOUT — The request is complete but changes have been backed out.
- BACKOUT_PENDING — The request is not yet complete, but changes have been or will be backed out.
- HC — The request is complete, and the transaction's changes have been heuristically committed.
- HR — The request is complete, but the transaction's changes have been heuristically backed out.
- HM — The request is complete, but some changes have been committed while others have been backed out.

3.3.6.5 Resource Managers, Interfaces, and Exits

When discussing commit protocols, the term "resource manager" traditionally refers to a software component that manages recoverable resources. Advantage CA-IDMS uses the term to refer to both resource and transaction managers and the specific interface that is used to communicate with them.

For example, when access is made from one Advantage CA-IDMS system to another, each system becomes a known resource manager on its partner. On the front-end, the partner system is identified by its node name and an interface name of "DSI_CLI"; on the back-end, the partner system is identified by its node name and an interface name of "DSI_SRV". Consequently, a central version may have knowledge of several resource managers whose interface name is DSI_CLI or DSI_SRV, since it may communicate with several other Advantage CA-IDMS systems. Furthermore, a central version may have knowledge of two resource managers with the same node name, one for each of the two interfaces, since a system can act as a front-end and a back-end to another system.

When a CICS system is used to access a central version, it becomes a known resource manager on that central version and is identified through a combination of its CICS system name and the name of the IDMSINTC interface module through which it is accessed.

To participate in a two-phase commit operation coordinated by Advantage CA-IDMS, a resource manager makes its existence known by registering with the local transaction manager. When registering, the resource manager interface identifies exit routines to be invoked by the transaction manager during the commit process. In this way, the resource manager interface acts as the bridge between the local transaction manager and the resource or transaction manager to which it provides access. It is the resource

manager interface's responsibility to forward prepare, commit, and backout directives and return appropriate responses to the local transaction manager.

When a resource manager's exit is invoked, it returns outcomes that are similar to the transaction outcomes outlined above. For example, a resource manager's prepare exit can return a FORGET outcome to signify that it has made no changes within the scope of the transaction and therefore need not participate in the second phase of the commit operation.

3.3.6.6 Interests and Roles

In order for a resource manager to participate in a transaction, it must register an **interest** in that transaction. The existence of an interest informs the Advantage CA-IDMS transaction manager that the resource manager's exits should be invoked during commit and backout processing.

When an interest is registered, the **role** that the resource manager is to play with respect to the transaction is specified. Advantage CA-IDMS recognizes the following roles:

- **Communications Resource Manager** (**CRM**) — indicating that the resource manager is a remote participant in the transaction.
- **Server Distributed Resource Manager (SDSRM)** — indicating that the resource manager is the coordinator for the transaction.
- **Participant (PART) —** indicating that the resource manager is a local participant in the transaction. When a central version application calls a resource manager interface to access a remote resource, the interface registers a CRM interest in the application's current transaction, since it is acting as a participant in that transaction. When remote access to a central version is provided from an environment controlled by an external transaction manager, the interface providing that access registers an SDSRM interest in the transaction since the external transaction manager is the transaction's coordinator.

As a two-phase commit operation proceeds, interests are assigned states similar to the transaction states outlined above. For example, if an interest's prepare exit returns OK, the state of the interest is set to InDoubt, reflecting the fact that the associated resource manager is waiting for the final commit or backout directive.

3.4 Impact on System Definition

In order to be able to successfully resynchronize, a coordinator must be able to communicate with participating systems. During resynchronization, the only information that an Advantage CA-IDMS coordinator has about another Advantage CA-IDMS system is its node name. The node name is used as the resource name in opening a DTS connection and hence the coordinating system's resource table and node definitions must be capable of supporting such a connection. To this end, ensure that every partner system that can be a participant is defined to the coordinator in one of the following ways.

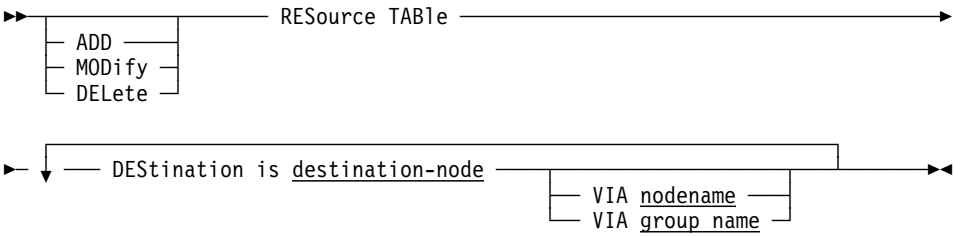
1. Define the partner system as a NODE in the coordinator's system definition. This option is appropriate if there is a direct communications path between the two systems and dynamic routing through DBGROUPs is not used or if forcing a specific access method with dynamic routing.
2. Define the partner system's node name as a destination in the coordinator's resource name table and identify the DBGROUP to which new connections should be routed by specifying a VIA parameter. This option is appropriate if dynamic routing is in use and the default access method is acceptable. Wildcarding the destination name can eliminate the need for defining a resource for every DBGROUP member and thus allows additional members to be added to the group without defining new resources.
3. Define the partner system's node name as a destination in the coordinator's resource name table and identify the intermediate node through which communication should be routed by specifying a VIA parameter. This option should be used only if there is no direct communications path between the two systems.

To facilitate earlier resynchronization when a failed participating system is restarted, it is advisable, though not required, to similarly define each potential coordinator within the participating system's definition.

3.4.1 System Generation Resource Table

Release 16.0 supports the ability to specify that database sessions targeted to a specific node be routed via a DBGROUP. This enables startup to resynchronize with participating nodes that are not defined by a NODE statement in the system definition.

3.4.1.1 Syntax



3.4.1.2 Parameters

VIA nodename

Identifies the name of the DC/UCF system where the named resource is located or the name of an intermediate node through which the request for data will be routed.

Nodename is the system name identified on the SYSTEM ID parameter of the SYSTEM statement or overridden by the DCNAME parameter in the SYSIDMS file at startup and must match a nodename defined with the NODE statement.

VIA groupname

Identifies the name of the DBGROUP to which the request for data will be dynamically routed.

Groupname is the name of a DBGROUP defined in the database name table of one or more DC/UCF systems, any of which are capable of servicing the request for data. *Groupname* must match a nodename defined by a NODE statement specifying a GROUP parameter.

3.4.1.3 Usage

Specifying Resources: The type of resource you specify is a database or a destination (nodename). In most cases, you will specify a database name. The only time you need to explicitly define a destination resource is when the target node is not defined using a NODE statement. This can occur because:

- There is no direct communications path from the system being defined to the target system, thus requiring database requests to be routed through an intermediate node. If this is the case, the target node should be defined as a destination resource and the intermediate node specified in its VIA parameter.
- Dynamic routing through DBGROUPs is used to direct database connections to any of several systems capable of processing the request and the default access method can be used to access the dynamically selected target node. If this is the case, the members of the DBGROUP should be defined as destination resources and the name of the DBGROUP specified in their VIA parameter. If the node names of all members of a DBGROUP follow a unique naming convention, then wildcarding can be used in the destination's nodename to reduce the number of required definitions and allow for the addition of new members without changing the resource table.

3.4.1.4 Example

Defining a Resource Table:

```
ADD RESOURCE TABLE
    DBNAME IS SYS104 VIA EDCQAM01
    DESTINATION IS PRODC* VIA CUSTGRP
    DBNAME IS MIS* VIA SYSTEM84.
```

3.5 Impact on System Operations

This section describes the impact that two-phase commit has on system operations.

3.5.1 Restarting a Failed System

When restarting a failed central version, it is advisable to restart it on the same logical operating system image as the one on which it abnormally terminated. This ensures that the restarted system can access (and be accessed by) the same systems with which it was able to communicate prior to the abnormal termination regardless of the intersystem access methods being used. If the restarted system cannot communicate with another system, it is not able to resynchronize with that system. This may leave incomplete transactions holding locks that prevent access to portions of the database. Resynchronization eventually completes when the necessary intersystem communications are reestablished. For more information on the resynchronization process, refer to 3.7, “Impact on Recovery” on page 3-24.

3.5.2 System Name During Warmstart

A system must be restarted using the same name that it had at the time of failure. To ensure that this is true, the name of an Advantage CA-IDMS system is recorded on its journal files. During warmstart, the system name on the journal files is used as follows:

- If no DCNAME parameter is specified in the SYSIDMS file of the system's startup JCL, the name of the system is taken from the value stored on the journal files. Any value specified in the system definition is ignored.
- If a DCNAME parameter is specified in the SYSIDMS file, it must match the value stored on the journal files, otherwise warmstart fails

The use of a DCNAME parameter is optional except for systems that are members of a data sharing group or that share a single system definition using the cloned system capability. This latter requirement is new for Release 16.0.

3.5.3 Incomplete Distributed Transactions at Startup

When restarting a failed central version, warmstart identifies incomplete distributed transactions that were active at the time of failure. Depending on where in the commit process the failure occurred, these transactions are completed by warmstart or are restarted later during the startup process. If restarted, the transactions remain active until resynchronization takes place with the other resource or transaction managers involved in the transaction or until the transactions are manually completed.

If a restarted transaction is in an InDoubt state, then any locks held by that transaction at the time of failure are reacquired and held until the transaction is completed. Since these locks prevent access to resources that were updated by the transaction, it is important to restart all failed systems as soon as possible so resynchronization can complete the transaction and free the locks.

The following sample messages might be displayed when a distributed transaction is restarted:

```
IDMS DC202038 V74 In-Doubt Transaction-ID 1416 will be added to the unrecovered transaction list
IDMS DC202051 V74 Warmstart COMPLETE, but recovery of SOME transactions have  been DEFERRED until later in startup.
IDMS DB342017 V74 T1 Will lock Transaction-ID 1416
IDMS DB342019 V74 T1 DTRID SYSTEM74::01650C90A708A9B2-01650C8C4207D9FF active at startup
IDMS DB342020 V74 T1 DTRID SYSTEM74::01650C90A708A9B2-01650C8C4207D9FF has been restarted
IDMS DB342022 V74 T1 In-Doubt Transaction 1416 has been restarted
```

3.5.4 Incomplete Distributed Transactions at Shutdown

Distributed transactions whose commit process was interrupted will remain active until resynchronization has completed successfully with all participants affected by the failure. Such transactions are said to be "pending resynchronization." If transactions are still pending resynchronization at the time a shutdown request is issued, the system will not shutdown successfully. Instead, it displays the following message and terminates abnormally with abend code 3937.

```
IDMS DC200241 V74 T1 Active transactions exist. Abending.
```

When the system is next restarted, the incomplete distributed transactions that were pending resynchronization are restarted and their locks reacquired.

To avoid abnormal terminations at shutdown, you should ensure that no distributed transactions are pending resynchronization before issuing the shutdown command. You can determine whether such transactions exist by issuing a DCMT DISPLAY DISTRIBUTED RESOURCE MANAGER command. To complete these transactions, you must restart the affected system and either allow it to resynchronize automatically or force it to resynchronize by issuing a DCMT VARY DISTRIBUTED RESOURCE MANAGER RESYNC command.

For more information on:

- Resynchronization — refer to 3.7, “Impact on Recovery” on page 3-24.
- DCMT DISPLAY DISTRIBUTED RESOURCE MANAGER command — refer to 3.5.5, “Monitoring Distributed Commit Operations.”
- Issuing the DCMT VARY DISTRIBUTED RESOURCE MANAGER RESYNC command — refer to A.16, “DCMT VARY DISTRIBUTED RESOURCE MANAGER.”

3.5.5 Monitoring Distributed Commit Operations

Recovery utilities such as PRINT JOURNAL and FIX ARCHIVE have been enhanced to report on distributed transactions encountered during their processing. Journal report 8 reports any distributed transaction journal records that it encounters. For details on these enhancements, see Chapter 5, “Administrative and Operational Enhancements.”

In addition, the following DCMT commands provide the ability to monitor various aspects of distributed commit operations:

- DCMT DISPLAY DISTRIBUTED RESOURCE MANAGER

- DCMT DISPLAY DISTRIBUTED TRANSACTION

3.5.5.1 DCMT DISPLAY DISTRIBUTED RESOURCE MANAGER

This command displays information about resource and transaction managers that are known to a system.

You can display summary information about all resource managers or details about an individual resource manager.

The summary display is useful in determining whether there are resource managers for which resynchronization has not completed and whether incomplete distributed transactions are pending completion of that resynchronization.

The detailed display can be used to determine which transactions are pending resynchronization with a particular resource manager.

For more information on this command, see A.6, “DCMT DISPLAY DISTRIBUTED RESOURCE MANAGER.”

3.5.5.2 DCMT DISPLAY DISTRIBUTED TRANSACTION

This command displays information about distributed transactions.

You can display summary information about all active distributed transactions or those pending resynchronization or details about an individual distributed transaction.

The summary display is useful in determining if any distributed transactions exist and whether any are pending resynchronization.

The detailed display can be used to determine the coordinator and participants in a distributed transaction. It also shows an external identifier if an external transaction manager such as RRS has assigned one. This information might be useful in determining how to complete a distributed transaction in the event of resynchronization failure.

For more information on this command, see A.7, “DCMT DISPLAY DISTRIBUTED TRANSACTION.”

3.6 Impact on Journaling

This section describes the new journal records that are written in support of a two-phase commit operation and the changes to existing journal records. It also describes a new journal format option that can be specified.

3.6.1 New Journal Records and Formats

Distributed transaction journal records are written during a two-phase commit operation to record the various states of a distributed transaction and to facilitate resynchronization in the event of failure. Each of them contains a Distributed Transaction Identifier (DTRID), a 16-byte value that distinguishes one distributed transaction from another.

The following new journal record types can be written in support of a two-phase commit operation:

- DIND (In doubt) — Written by a participant after it has successfully prepared its resources for commit and prior to returning an OK response to its coordinator.
- DCOM (Commit) — Written by a coordinator to signify that a transaction's changes will be committed. Its existence separates the first and second phases of the commit process. A participant also writes a DCOM immediately upon receiving a *Commit* request from its coordinator.
- DBAK (Backout) — Written by a coordinator to signify that a transaction's changes will be backed out. Its existence separates the first and second phases of the commit process. A participant also writes a DBAK immediately upon receiving a *Backout* request from its coordinator but only if a DIND had previously been written.
- DPND (Pending) — Written by a coordinator during the second phase of a commit operation if some participant is unable to complete its commit processing due to a failure. By writing this record, the coordinator is able to forget some participants while remembering others. A DPND can also be written by either a coordinator or a participant to record heuristic outcomes.
- DFGT (Forget) — Written by coordinators and participants when they have completed their two-phase commit processing for a transaction. A DFGT record is written only if some other Dxxx record was previously written.

DIND, DCOM, and DBAK records contain the Local Transaction Identifiers (LIDs) identifying the work done by local transaction branches participating in a distributed transaction. A LID is the 4-byte transaction identifier that is carried in the local transaction-related journal records (for example, BFOR, AFTR, COMT). The journal records for a distributed transaction are interspersed with associated local transaction records as follows:

- BGIN — indicating the start of a local recovery unit
- BFOR/AFTR — one or more pairs

- **DIND — on a participant only**
- **DCOM or DBAK — on a participant and a coordinator**
- COMT or ENDJ — if a DCOM was written
- ABRT — if a DBAK was written
- **DPND — on a coordinator if the commit operation was interrupted**
- **DFGT — on a participant and a coordinator if any other Dxxx record was written**

DIND, DCOM, DBAK, and DPND records contain information about a participant's coordinator and about a coordinator's participants. The specific information that is recorded varies depending on the type of the coordinator or participant. For example, the node name, resource name, and remote transaction branch identifier are recorded for Advantage CA-IDMS participants. The RRS URID (Unit of Recovery Identifier) is recorded for an RRS coordinator or participant.

Dxxx records can be larger than a single disk journal block. If this is the case, they are split into as many journal blocks as are necessary to hold the entire Dxxx record. It is also possible for a Dxxx record to be split across disk journal files and, hence, across archive files. The manual recovery utilities reassemble the record, provided that all necessary archive files are processed in a single execution of the utility. They ignore partial Dxxx records in which not all segments are present in the input file.

In addition to the above new journal records, the following journal records have been expanded to support two-phase commit processing:

- JHD1 (Journal Header 1) — A record that occupies the first block of each disk journal file. It was expanded to record information about other systems with which this system communicates. This information includes the journal stamp of the other system and how to communicate with it.
- JHD2 (Journal Header 2) — An overflow block for a JHD1; it is optionally allocated when a disk journal file is formatted.
- JSEG/DSEG/JSGX — Records that track active transactions across a journal swap and other key points. They are used by warmstart during the automatic recovery process and have been expanded to track active distributed transactions in addition to active local transactions.

3.6.2 Journal File Formatting Considerations

When formatting journal files, it might be necessary to specify a size for the amount of space to be reserved for recording information about other systems with which a system communicates. In most cases, the default size is sufficient and no explicit size parameter is needed; however, if a system's journal block size is very small or it communicates with many other Advantage CA-IDMS or CICS systems, it may be necessary to reserve additional space. For information on how to specify a storage size, see Chapter 5, “Administrative and Operational Enhancements.”

If a journal's available space is exhausted, it is necessary to shut down the system, offload and format its journal files, and restart the system before communications with new systems can take place.

3.7 Impact on Recovery

This section describes the impact that two-phase commit has on recovery operations. It describes the automatic recovery that is provided through resynchronization as well as considerations for manual recovery operations.

3.7.1 System Recovery Interdependence

In general, recovery from failures that occur during two-phase commit processing is accomplished automatically through the resynchronization process, just as warmstart automatically recovers from failures during local transaction processing. The one important consideration when dealing with distributed transaction recovery is that systems are no longer independent with respect to recovery. Information on a coordinator's journal files might be needed to complete recovery for one or more of its participants. If either system's journal files are formatted before resynchronization between the two systems has completed after a failure, then manual intervention might be needed to complete the recovery process.

3.7.2 Resynchronization Between Advantage CA-IDMS Systems

Resynchronization is a process in which information is exchanged between a two-phase commit coordinator and a participant to establish attributes relevant to the two-phase commit process and complete outstanding distributed transactions following a failure.

Depending on the systems involved and the nature of the failure, resynchronization can occur automatically or can require explicit action to be triggered. This chapter focuses on resynchronization between Advantage CA-IDMS systems.

For information on resynchronization between CICS and Advantage CA-IDMS, see 3.8, “Two-Phase Commit Support with CICS” on page 3-30.

For information on resynchronization between RRS and Advantage CA-IDMS, see 3.9, “Two-Phase Commit Support with RRS” on page 3-36.

3.7.2.1 When Does It Occur?

Resynchronization between Advantage CA-IDMS systems occurs as follows:

- When a central version is started, resynchronization is initiated with each *known* back-end system. A back-end system is *known* if it was accessed since the last time the journal files were formatted. Information about other systems is recorded in a system's journal files (in the JHD1 record). If the started system cannot communicate with one or more of its back-end systems, resynchronization is retried on a periodic basis until communication is reestablished.
- When a remote database session is started, resynchronization is initiated if the back-end system was previously unknown (that is, if this is the first time the back-end system has been accessed since the journal files were formatted) or if the

back-end system has been recycled since resynchronization previously took place between the two systems.

Note: A remote database session is started when an application binds a run unit or connects an SQL session to a remote database. It is also started when a DCUF task is executed to establish a remote default dictionary.

- When resynchronization is manually driven through a DCMT VARY DISTRIBUTED RESOURCE MANAGER command. For more information, see A.16, “DCMT VARY DISTRIBUTED RESOURCE MANAGER.”

3.7.2.2 What Does It Entail?

Resynchronization begins with an exchange of startup times and journal timestamps between the two systems.

As the name implies, the startup time is the time at which a system was started and is used to detect when a partner system is recycled.

The journal timestamp is assigned by a central version the first time it opens a set of journal files after they have been formatted. It is subsequently used to detect when a partner's journal files have been reformatted since the last time the two systems resynchronized with each other.

If no distributed transactions involving the two systems exist at the time that resynchronization takes place, the two systems simply exchange the above information, update their journal files with new or changed partner information, and record each other as open resource managers.

If distributed transactions involving the two systems do exist at the time of resynchronization, each system compares its partner's current journal timestamp with the one that it had saved previously. If the timestamps are the same, resynchronization proceeds by exchanging information about the incomplete distributed transactions that are pending resynchronization. If the timestamps are not the same, it is an indication that one of the following has occurred:

- The partner system's journal files have been prematurely formatted.
- The partner system has been started with incorrect journal files.
- The partner system has been started with an incorrect DCNAME parameter.

Any of these conditions result in a resynchronization failure.

3.7.2.3 Responding to Resynchronization Failures

If resynchronization detects a journal stamp mismatch with a system for which incomplete distributed transactions exist, resynchronization cannot complete. When this occurs, messages are displayed that show the old and new journal stamps and the incomplete distributed transactions that are impacted by the mismatch. The operator is prompted as to what action should be taken. The following example shows the

messages that are displayed as a result of a mismatch in SYSTEM74's journal stamps as they are known to SYSTEM73.

```
DC329021 V73 T23 Journal stamp mismatch for SYSTEM74::DSI_SRV *OLD yyyy-mm-dd-hh.mm.ss.ssssss
DC329021 V73 T23 Journal stamp mismatch for SYSTEM74::DSI_SRV *NEW yyyy-mm-dd-hh.mm.ss.ssssss
DC329022 V73 T23   RM Name      Dtrid                          Branch          State
DC329023 V73 T23   SYSTEM74::DSI_SRV  SYSTEM74::01650D6EDFB1AB93-01650D6A79F31E50 InDoubt
DC329024 V73 REPLY 01 T23 Reply with resynchronization action for SYSTEM74::DSI_SRV (Ignore,Defer):
```

Before replying to message DC329024, the cause of the mismatch should be determined. The appropriate response should then be made as outlined in the following table. Until a response is made to the DC329024 message, no database access is permitted with the identified resource manager. Any task attempting such access waits until a response has been made or its wait time is exceeded.

Reply	Meaning and Considerations
IGNORE	This reply specifies that resynchronization with the resource manager should continue. The distributed transactions listed in the preceding DC329023 messages require manual completion. IGNORE is appropriate if the partner system's journal files have been prematurely formatted. In this case, the only way to complete the affected transactions is to do so manually, since the journal entries required to complete the transactions automatically are no longer available on the partner system's journal files. For guidance on how to manually complete the transactions, see 3.7.3, “Completing Transactions Manually” on page 3-27.
DEFER	This reply specifies that resynchronization with the resource manager should be postponed until a later time. Database access with the identified resource manager is disallowed until resynchronization has completed successfully. DEFER is appropriate if the mismatch can be corrected by recycling one or the other system. Perhaps one of the systems was started with incorrect journal files or the partner system was started with an incorrect DCNAME parameter. After replying DEFER, the system in error should be shutdown and restarted correctly. It may then be necessary to initiate resynchronization using a DCMT VARY DISTRIBUTED RESOURCE MANAGER command.

3.7.3 Completing Transactions Manually

In certain circumstances, it may be necessary to complete a distributed transaction manually. The need for this should be extremely rare and is a consequence of a failure in resynchronization. The ability to manually complete distributed transactions is provided for situations such as the permanent inaccessibility of a partner system or the premature formatting of journal or non-Advantage CA-IDMS log files during a recovery operation.

When manually completing a transaction whose state is InDoubt, you must specify whether to commit or back out the transaction's changes. You should research the situation carefully before taking any action. If you make the wrong decision, the distributed transaction will have a mixed outcome, meaning that some of its changes are committed while others are backed out. The following sources of information may be helpful in determining the correct action to take:

- The output from a DCMT DISPLAY DISTRIBUTED TRANSACTION command indicates what system is acting as the coordinator for the transaction.
- Use the facilities provided by the coordinator to determine the outcome of the transaction.
  - If the coordinator is an Advantage CA-IDMS system, its journal files contain a DCOM record for the transaction if its changes should be committed. **The PRINT JOURNAL summary report lists all incomplete distributed transactions.** If there is no entry for the transaction and no journal information is missing, then the transaction's changes should be backed out.
  - If the coordinator is RRS, the RRS ISPF panels can be used to determine the outcome of the transaction. For more information on RRS panels, refer to the IBM guide *MVS Programming: Resource Recovery*.
  - If the coordinator is CICS, examine its log file or use CEMT commands to determine the outcome of the transaction.

Once you have determined whether a transaction's changes should be committed or backed out, issue a DCMT VARY DISTRIBUTED TRANSACTION command to complete it specifying COMMIT or BACKOUT. Doing so marks the transaction as heuristically committed or backed out accordingly. The transaction remains active, holding no locks, until resynchronization is completed with the coordinator or a DCMT VARY DISTRIBUTED TRANSACTION command is issued that specifies FORGET. Waiting for resynchronization is recommended since the overall status of the transaction can be checked for consistency...meaning that all changes are committed or backed out. If a mixed outcome is detected, this is noted on the log and the transaction remains active until a DCMT VARY DISTRIBUTED TRANSACTION is issued specifying FORGET.

For more information on this command, see A.17, “DCMT VARY DISTRIBUTED TRANSACTION.”

3.7.4 Manual Recovery Considerations

If manual recovery becomes necessary, the process is generally the same regardless of whether the archive journal files contain distributed transaction journal records (Dxxx records) or not.

However, special action may be needed when a ROLLFORWARD operation terminates or a ROLLBACK operation begins at a point in time where a distributed transaction is active and in an InDoubt state. The problem that arises in such a situation is that the recovery utility does not know whether to commit the local changes made by the InDoubt transaction or back them out. Since the utility has no way of communicating with a coordinator to determine what action to take, it may be necessary for the DBA to explicitly specify the final outcome for the transaction.

3.7.4.1 InDoubt Transactions During Manual Recovery

A distributed transaction is in an InDoubt state when the last journal record written for that transaction is a DIND. Normally, a DCOM or a DBAK record follows a DIND, and its presence determines whether a transaction's changes should be committed or backed out. The absence of a DCOM or DBAK record may be because:

- It has not been written because resynchronization with the transaction's coordinator has not completed.
- It exists but on a later archive journal file that is not being processed in the current execution of the recovery utility.
- It exists but is split between two archive journal files, only the first of which is being processed in the current execution of the recovery utility.

By default, the recovery utilities leave an InDoubt transaction in its InDoubt state, meaning that its changes are not rolled out. A DBA can override this default behavior by adding an entry to a manual recovery control file to explicitly specify the action to be taken for an InDoubt transaction.

Explicitly overriding the default action should normally not be necessary. In fact, the presence of InDoubt transactions at the end of a ROLLFORWARD or ROLLBACK operation should be researched to determine the reason for their existence and to ensure that the recovery procedure being followed is valid and includes all necessary journal input.

An InDoubt transaction might validly be encountered when recovering a damaged database file. In this case, the transaction should be allowed to remain InDoubt. When the recovered file is subsequently varied active to the central version, the transaction is completed (backed out or committed) automatically.

Generally, the only time that an InDoubt transaction should be explicitly completed is in exceptional situations such as:

- When a coordinator is permanently inaccessible

- When a coordinator's journal files have been prematurely formatted
- When a participant's journal files have been damaged.

Even in the first two situations, if the transaction is still active within the participant central version it should be completed using a DCMT VARY DISTRIBUTED TRANSACTION command rather than using a manual recovery control file override.

For more information on the format and use of the manual recovery control file, see Chapter 5, “Administrative and Operational Enhancements.”

3.7.5 Deleting Resource Managers

If a resource manager becomes inaccessible or is removed from the network, it can be deleted by issuing a DCMT command. Even in these cases, there is often no need to explicitly delete a resource manager, since it disappears when the journal files are next formatted. However, if incomplete distributed transactions exist that involve an inaccessible resource manager as a participant, then you may want to explicitly delete the resource manager in order to enable the transactions to be completed.

You can delete a resource manager by issuing a DCMT VARY DISTRIBUTED RESOURCE MANAGER command, specifying DELETE. Doing so removes the resource manager from the system, purges it from the journal files and deletes all associated transaction interests. Clearly, this command should be used with care.

The following procedure should be followed to delete a resource manager:

1. Obtain a list of transactions in which the resource manager has an interest by issuing a DCMT DISPLAY DISTRIBUTED RESOURCE MANAGER command for the target resource manager.
2. For each listed transaction determine whether the resource manager is a coordinator or a participant by displaying its detail using a DCMT DISPLAY DISTRIBUTED TRANSACTION command.
3. Complete each transaction for which the resource manager is a coordinator by issuing one or more DCMT VARY DISTRIBUTED TRANSACTION commands.
4. Issue a DCMT VARY RESOURCE MANAGER ... DELETE to delete the resource manager.
5. Complete each transaction for which the resource manager was a participant by issuing a DCMT VARY DISTRIBUTED TRANSACTION command.

   For more information on this command, see A.17, “DCMT VARY DISTRIBUTED TRANSACTION” on page A-27.

3.8 Two-Phase Commit Support with CICS

A two-phase commit protocol can optionally be used when accessing an Advantage CA-IDMS database from a CICS application. Using a two-phase commit protocol ensures that updates made to Advantage CA-IDMS data are coordinated with those made to other recoverable resources that the application accesses within the same CICS UOW (Unit Of Work). Two-phase commit support is provided only when using a Release 16.0 CICS interface (IDMSINTC) to access a Release 16.0 back-end CV.

Note: Two-phase commit is not supported through the IDMSINTL CICS interface.

3.8.1 Implementation Requirements

In order for successful two-phase commit operations between CICS and Advantage CA-IDMS, the following steps must be taken:

- Review the new IDMSCINT and CICSOPT parameters. While there is no need to create a new IDMSCINT and relink your application programs, you may want to in order to take advantage of some of the new IDMSCINT options.
- Assemble a new CICSOPT options table and link an IDMSINTC interface module for each interface for which you wish to enable two-phase commit processing.

  Note: This must be done for all IDMSINTC interface modules when upgrading to Release 16.0 even if two-phase commit processing is not enabled.
- Ensure that each CICS system is uniquely identified through the TPNAME parameter of the CICSOPT macro or the new CICS_NAME SYSIDMS parameter.
- Ensure that the CICS system is logging transaction information. This requires the use of a CICS log file. For more information, refer to the appropriate CICS documentation.
- Create a CICS RSYN transaction and program for each CICS interface module used within the CICS system.
- If using an OPTIXIT or OPTIQXIT to route requests to different back-end central versions, modify the OPTIXIT to recognize and correctly route resynchronization requests.

The remainder of this chapter discusses these requirements and other aspects of two-phase commit support between CICS and Advantage CA-IDMS.

3.8.2 Programming Interface

A CICS commit operation is initiated through an explicit CICS SYNCPOINT command or at normal CICS task termination. Regardless of how it is initiated, CICS becomes the coordinator and the back-end Advantage CA-IDMS system(s) become participants.

A CICS backout operation is initiated when one of the following occurs:

- An explicit CICS BACKOUT command is issued.
- A CICS task terminates abnormally.
- An Advantage CA-IDMS database session, for which the parameter AUTONLY is enabled, is rolled back. See 3.8.4, “Requesting the Use of Two-Phase Commit” for information about the AUTONLY parameter.

3.8.3 Optimizations Supported

In order to minimize the cost of doing a CICS syncpoint operation, the Advantage CA-IDMS CICS interface supports the CICS single-update and read-only optimizations.

The CICS **single-update** optimization permits CICS to make a single phase commit request to the Advantage CA-IDMS CICS interface rather than separate Prepare and Commit requests, if it is the only updating resource manager participating in the UOW.

The CICS **read-only** optimization permits CICS to make a single phase commit request to the Advantage CA-IDMS CICS interface if it has made no updates within the CICS transaction. Furthermore, if all resource managers but one are read-only, CICS can avoid the overhead of a two-phase operation by directing the sole updater to do a single phase commit.

These optimizations not only reduce communications with participating resource managers, but also reduce log and journal overhead. For more information on the single-update and read-only optimizations, refer to the appropriate CICS documentation.

3.8.4 Requesting the Use of Two-Phase Commit

Whether the work done by a database session is to be included in a CICS UOW is determined at the time a database session is opened. A database session is opened when a bind run unit or the first SQL statement is executed. When a session's work is included in a CICS UOW, its changes are committed or backed out as directed by CICS and the Advantage CA-IDMS interface uses a two-phase commit protocol to achieve the desired outcome.

Several new and enhanced IDMSCINT and CICSOPT parameters control whether a database session is included in a CICS UOW and therefore if a two-phase commit protocol is used.

- AUTOCMT — Enabling this option makes the work done by the database session eligible for inclusion in a CICS UOW. The following determine if it is actually included:
  1. The AUTONLY setting
  2. Whether the application issues its own commit or rollback DML requests before the CICS syncpoint operation

- AUTONLY — Enabling this option forces the work done by the database session to be included in the CICS UOW. DML statements that would typically commit work (such as FINISH or COMMIT WORK) do not cause changes to be committed even if the session itself is terminated. The session's changes are committed only when the CICS syncpoint occurs. On the other hand, if the changes made by a session for which AUTONLY is enabled are backed out, either as the result of a DML ROLLBACK request or because of some environmental condition such as a deadlock, the entire CICS UOW is immediately backed out. This ensures consistent behavior across all resources updated by the application.

  If AUTONLY is not enabled and AUTOCMT is enabled, the work done by the database session is included in the CICS UOW provided that the application does not issue commit or rollback DML requests prior to the CICS syncpoint operation.

  AUTONLY is ignored if AUTOCMT is not enabled.

  Note: If transaction sharing is enabled, AUTONLY and AUTOCMT are automatically enabled.

- ONCOMT — This option specifies the effect that a CICS syncpoint operation has on a database session whose work is included in the CICS UOW. The session can optionally be treated as if a FINISH, COMMIT ALL or COMMIT CONTINUE were issued, meaning that it can be terminated, remain active but have currencies cleared or remain active with currencies left in-tact.

- ONBACK — This option specifies the effect that a CICS backout operation has on a database session whose work is included in the CICS UOW. The session can optionally be treated as if a ROLLBACK or a ROLLBACK CONTINUE were issued, meaning that it can be terminated or remain active but have its currencies cleared.

All of these options can be specified through both IDMSCINT and CICSOPT parameters. The CICSOPT parameters can override their IDMSCINT counterparts or be used as defaults. For more information on these new parameters, refer to Appendix F, “CICS Interface Enhancements for Two-Phase Commit Support.”

3.8.5 Additional Two-Phase Commit Parameters

In addition to the parameters that control whether a two-phase commit protocol is used, four new CICSOPT parameters affect two-phase commit processing.

- TRUE — Specifies a 5-character prefix used in forming TRUE (Task Related User Exit) entry names. The prefix must be unique across all IDMSINTC interface modules in use within a single CICS system.

- MAXCON — Specifies the maximum number of Advantage CA-IDMS systems that can be concurrently accessed by an application using an IDMSINTC interface. This limit applies only to systems accessed through database sessions for which AUTOCMT is enabled.

- MAXIDMS — Specifies the maximum number of Advantage CA-IDMS systems that an IDMSINTC interface can access during the life of a CICS system. This

limit applies only to systems accessed through database sessions for which AUTOCMT is enabled.

- RSYNTXN — Specifies the name of the resynchronization transaction defined to CICS for this interface. A separate CICS transaction must be defined for each interface in use within a CICS system. For more information on the CICS resynchronization transaction, refer to The RSYN Transaction and Program Appendix F, "CICS Interface Enhancements for Two-Phase Commit Support" on page F-1.

For more information on these new parameters, refer to Appendix F, "CICS Interface Enhancements for Two-Phase Commit Support."

3.8.6 CICS System Name Requirements

An important consideration for successful two-phase commit operations between CICS and Advantage CA-IDMS is that the name of every CICS system have a consistent name that is unique across all CICS systems accessing a central version.

The name of the CICS system is established as:

- The value in the new CICS_NAME parameter specified in the SYSIDMS file included in the CICS startup JCL.

Or, if the CICS_NAME parameter is not specified:

- The value of the TPNAME parameter associated with the first IDMSINTC interface within a CICS system.

If the CICS name is allowed to default to the TPNAME of the first CICS interface, all other IDMSINTC interface modules started within the CICS system must have the same TPNAME value, otherwise they will fail with a K213 abend code.

When restarting a CICS system, its name must remain unchanged if it is involved in incomplete distributed transactions that are still active on a central version. Changing the name while incomplete transactions exist may make it necessary to complete those transactions manually.

The description of the new SYSIDMS parameter for specifying a CICS system name is as follows:

```
CICS_NAME=<CICS-name>
```

Where:

<CICS-name> — specifies a 1-4 character value that identifies the CICS system being started. It must be unique across all CICS systems that access the same central version.

3.8.7 Resynchronization between CICS and Advantage CA-IDMS

Resynchronization is part of the recovery process that takes place following a failure during a two-phase commit operation. It involves the exchange of information between a coordinator and a participant in order to resolve incomplete units of work. Release 16.0 provides a mechanism to resynchronize a CICS system (the coordinator) and an Advantage CA-IDMS central version (the participant) following abnormal terminations of either system.

Resynchronization between CICS and Advantage CA-IDMS is undertaken in the context of a specific interface module (IDMSINTC). This means that if multiple interface modules are used within a single CICS system to access a given back-end CV, a separate resynchronization process takes place for each one. Consequently, resynchronization actually takes place between a CICS interface running on a given CICS system and a back-end CV rather than between a CICS system and a back-end CV.

3.8.7.1 The Resynchronization Transaction and Program

Resynchronization between a CICS interface and an Advantage CA-IDMS central version is done through execution of a resynchronization transaction defined to CICS. The Advantage CA-IDMS installation default name for this transaction is RSYN. The resynchronization transaction is associated with a resynchronization program whose installation default name is IDMSCSYN. A separate resynchronization transaction and program must be created for each CICS interface module (IDMSINTC) that is used within a CICS system and the name of the transaction must be specified in the RSYNTXN parameter of the interface's CICSOPT macro. Failure to define the CICS resynchronization transaction causes any task attempting to open a database session for which AUTOCMT is enabled to fail with an abend code of K209.

For details on defining the transaction and creating the resynchronization program, refer to Appendix F, “CICS Interface Enhancements for Two-Phase Commit Support.”

3.8.7.2 How is Resynchronization Initiated?

Resynchronization between a CICS interface and an Advantage CA-IDMS central version is initiated in the following ways:

- In a CICS Transaction Server V1R1 (or later release) for z/OS or OS/390, resynchronization takes place automatically when the interface is started. It resynchronizes with all central versions accessed through the interface and known to CICS as participants in incomplete UOWs.
- When the first database session is connected through the interface to a back-end central version after either system is started, resynchronization takes place automatically for the central version being accessed.
- When the CICS resynchronization transaction (RSYN) is invoked manually, resynchronization takes place for the central version identified in the transaction invocation. For information about invoking a resynchronization transaction

manually, refer to Appendix F, “CICS Interface Enhancements for Two-Phase Commit Support.”

3.8.7.3 When Should You Manually Resynchronize?

Normally there is no need to manually initiate resynchronization since it occurs automatically when the first connection is made following restart of the CICS or Advantage CA-IDMS systems. However, if a particular back-end system is accessed infrequently through a given interface and incomplete transactions on the back-end system require resynchronization, you can invoke the RSYN task manually to force resynchronization to occur immediately.

In non-CICS Transaction Server V1R1 (or later release) for z/OS or OS/390 environments, manually initiated resynchronization can only take place when no user applications are accessing the back-end CV through the interface being resynchronized. Manual resynchronization terminates if such active connections exist. This restriction does not apply to CICS Transaction Server V1R1 (or later release) for z/OS or OS/390.

3.8.7.4 The Resynchronization Process

When the resynchronization task is executed (either automatically or manually), it retrieves a list of incomplete distributed transactions that are known to the central version with which it is resynchronizing and that are pending resynchronization with the associated CICS interface. It then issues a CICS RESYNC command to inform CICS of the Units of Work (UOWs) that are pending completion. CICS, in turn, initiates a CRSY task for each affected UOW. The CRSY task drives the TRUE syncpoint exit to inform the back-end central version as to whether to commit or back out the distributed transaction.

If the resynchronization task is initiated automatically, back-end tasks that are still awaiting communications from the CICS system are canceled with an abend code of RSYN. During automatic resynchronization, such tasks can only exist following an abnormal termination of the CICS system. While they eventually time out, the resynchronization process cannot proceed until they have terminated; therefore, it cancels them. Back-end tasks are not canceled if resynchronization is driven manually since there is no guarantee that activity between the two systems has been quiesced.

3.8.7.5 OPTIXIT Considerations

If an OPTIXIT or an OPTIQXIT program is used to route requests to different back-end central versions, the OPTIXIT must be enhanced to recognize and correctly route resynchronization requests. A resynchronization request is identified by a program name of INTCRSYN and the OPTI block that is passed to the exit contains the node name of the target system. The exit must use the node name to select an OPTI (if multiple SYSCTL support is enabled) or modify the OPTI passed on the request so that the resynchronization request is routed to the correct back-end system. To see an example of the type of processing needed, refer to Appendix F, “CICS Interface Enhancements for Two-Phase Commit Support.”

3.9 Two-Phase Commit Support with RRS

RRS is IBM's resource recovery platform for z/OS and OS/390. Release 16.0 of Advantage CA-IDMS can exploit RRS services in the following ways:

- A batch application can use RRS as a coordinator to ensure that the updates made through one or more central versions are coordinated with those of other resource managers such as MQSeries.
- An online application can update external resources through an RRS-enabled interface to ensure that those updates are coordinated with those made to Advantage CA-IDMS resources.

This section discusses how RRS support is enabled and describes considerations associated with its use.

3.9.1 Enabling RRS Support Within an Advantage CA-IDMS System

To exploit RRS functionality through batch or online applications, you must enable RRS support in one or more central versions. To do this, you specify parameters in specific columns of the EXEC statement's PARM field in the system's startup JCL. If the PARM field specifies the Advantage CA-IDMS DC/UCF system version number, column numbering starts in the column after the system number.

You use the PARM field to:

- Enable RRS support by specifying a T or an R in column 21. Specify a T if the system is to support multitasking; specify an R otherwise.
- Optionally specify one plus the number of subtasks that are capable of accessing RRS in columns 22-23. The value specified must be between 2 and 99. If no value is specified, the number of subtasks is determined as one plus the number of processors. If multitasking is also enabled, the value specified also represents the number of subtasks that perform Advantage CA-IDMS work.

  For example, the following PARM specification enables RRS support in a uni-tasking system and specifies that two subtasks should support access to RRS.

```
                                        Column   Column     Column
                                        0        1          2
                                        1        0          1
                                        ---+----+----+----+---
//STARTUP  EXEC PGM=DCUCFSYS,PARM='S=91                     R03'      .
```

The DCMT DISPLAY SUBTASK command has been enhanced to show what type of work a subtask can perform. A new DCMT VARY SUBTASK command can be used to alter the type of work that a subtask can perform. For more information on these commands, see A.27, “DCMT VARY SUBTASK.”

3.9.2 Impact on System Startup

If RRS support is enabled, a central version registers with RRS during startup. In so doing, it identifies itself as a resource manager with the following name:

```
IDMS.RM.nodename.CA
```

Nodename is the node name of the central version, padded with underscores ("_") if it is less than eight characters in length. The node name is specified in the SYSTEM ID parameter of the system definition's SYSTEM statement and can be overridden by a DCNAME parameter in the SYSIDMS file in the system's startup JCL. In order to use RRS support, the node name must be unique within the sysplex in which the system is executing.

The following message is displayed after a successful registration with RRS:

```
DC224001 V73 T23 Registered with RRS services as IDMS.RM.nodename.CA
```

Once registered, an Advantage CA-IDMS system typically remains so until shutdown. The following message is displayed when a central version deregisters with RRS:

```
DC221001 V73 T1 IDMS.RM.nodename.CA Unregistered from RRS; return code = 00000000
```

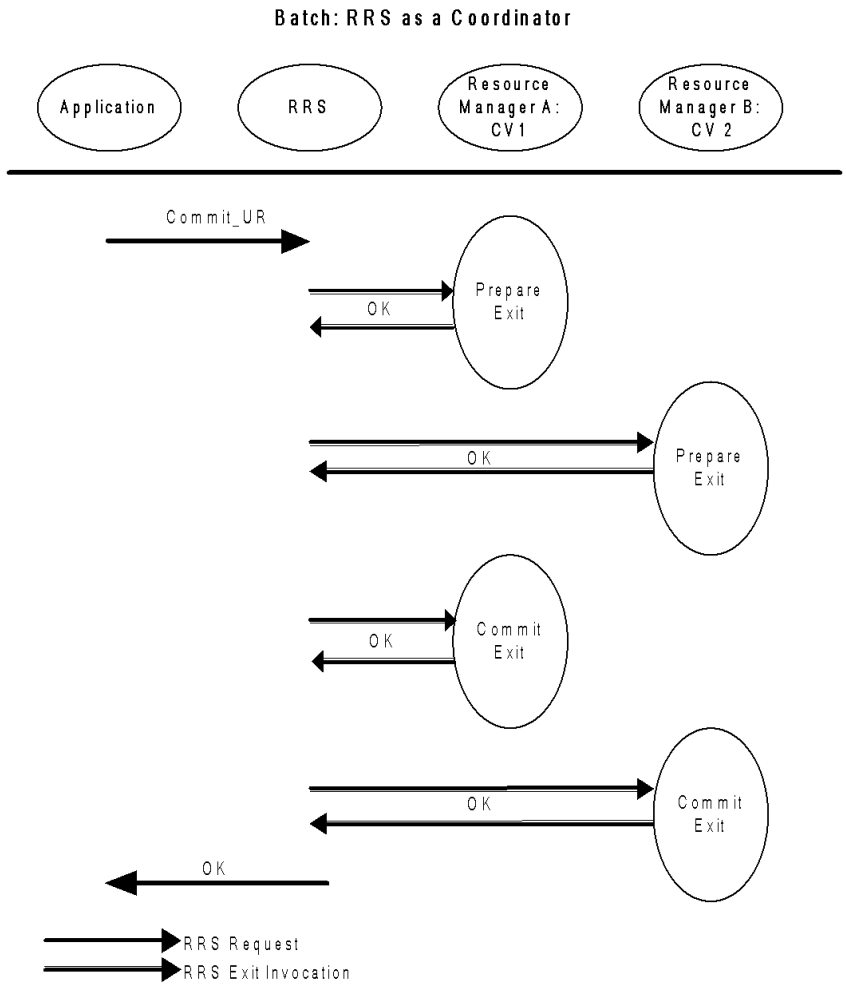
After successful registration with RRS, a resynchronization process is started in order to exchange information and complete recovery following a failure. Refer to 3.9.6, “Resynchronization Between RRS and Advantage CA-IDMS” on page 3-44 for more information.

The operating system image on which a failed system is restarted can be significant. For more details, see the IBM manual *MVS Programming: Resource Recovery* and the specific topic "Resource Manager Environments."

3.9.3 RRS Support for Batch Applications

A batch application updating resources controlled by multiple resource managers can make use of RRS services to guarantee atomicity of the updates. Advantage CA-IDMS supports RRS for batch applications that make their database updates through one or more central versions running on the same operating system image as the batch job.

When RRS is used as the coordinator, each resource manager (RM) that is accessed to perform work on behalf of a UR expresses an interest in it. To commit all changes as a unit, the application issues a Commit_UR (or an HLL Application_Commit_UR) request to RRS. The following diagram illustrates the flow of control that occurs:



3.9.3.1 Example

Consider a batch application that accesses Advantage CA-IDMS and MQSeries and wishes to coordinate the work done on each. To do this the central version must be accessed through an RRS-enabled batch interface. The interface passes a context token to the central version so that it can express an interest in the UR associated with the context. At commit time, RRS invokes the central version's prepare and commit exits so that its work is coordinated with that of MQSeries.

3.9.3.2 Enabling RRS for Batch Applications

A batch application tells Advantage CA-IDMS that it wants to use RRS as a coordinator by specifying a new SYSIDMS parameter:

```
ENABLE_RRS=ON
```

Advantage CA-IDMS then extracts the current context token and passes it on to the central version, which expresses interest in it.

If ENABLE_RRS=ON is established as a default in a SYSIDMS load module, it can be overridden at runtime by specifying:

```
ENABLE_RRS=OFF
```

Notes:

- The central version(s) to which the batch application's database sessions are directed must be started with RRS support and must be running on the same operating system image.
- It is not possible to access a pre-Release 16.0 central version if the batch job runs with RRS enabled. Local access is supported but is not part of the RRS UR.
- The 10.2 services batch interface (also known as IDML) does not support RRS.

3.9.3.3 Batch RRS Transaction Boundaries and Application Design Considerations

Batch applications that use RRS as a coordinator have to be carefully designed. The usage of RRS implies these rules:

1. The application verbs that mark a transaction boundary are the RRS verbs: Commit_UR or Backout_UR.
2. Prior to issuing a Commit_UR, all database sessions whose transaction is under the control of RRS must be completed. This can be accomplished by:
   - Issuing a FINISH TASK DML command
   - Explicitly finishing all active database sessions by issuing a FINISH or COMMIT RELEASE DML command for each one

   Note: A FINISH TASK must be issued if a BIND TASK was issued.

   Finishing a database session does not terminate its associated transaction when it is under the control of RRS; instead, the database session is closed and currency locks are released, but the transaction remains active and update locks are maintained until the RRS UR is committed or backed out.

   It is possible to serially create and finish database sessions within a single RRS UR; however, unless transaction sharing is in effect, a deadlock may occur if a latter session attempts to access a record that was updated by a previous session.
3. When a ROLLBACK TASK DML command is issued, it ultimately results in the back out of the entire RRS UR, even if the application subsequently issues a Commit_UR request. At the time the ROLLBACK command is issued, the changes made to the Advantage CA-IDMS database are backed out and the associated locks are released. However, the RRS UR is not backed out until an RRS commit or backout operation is initiated. If necessary, Advantage CA-IDMS will vote "BACKOUT" during the first phase of commit processing to cause the RRS UR to be backed out.

4. When an application program ends (normally or abnormally), the associated RRS context is terminated by the operating system. RRS default actions are to commit on normal context termination and backout on abnormal context termination.

3.9.3.4 Example of a COBOL Batch Program

The following extracts from a COBOL program show how to invoke the RRS Commit_UR and Backout_UR services. The COBOL program is a subroutine that is called to perform a certain action as defined in ACTION-CD. Only the Advantage CA-IDMS task level and RRS actions are shown.

```
      *RETRIEVAL
      *NO-ACTIVITY-LOG
      *DMLIST
       IDENTIFICATION DIVISION.
       PROGRAM-ID.           MBINDSUB.
      ****************************************************************
      *  SUBSCHEMA CONTROL IS PASSED FROM MAINLINE PROGRAM.
      ****************************************************************
       ENVIRONMENT DIVISION.
       IDMS-CONTROL SECTION.
       PROTOCOL.                MODE IS BATCH DEBUG
                                IDMS-RECORDS MANUAL.
       DATA DIVISION.
       SCHEMA SECTION.
           DB EMPSS01 WITHIN EMPSCHM VERSION 100.
       WORKING-STORAGE SECTION.
       01  WK-DATA.
           02  I            PIC S9(4) COMP.
       01  COPY IDMS SUBSCHEMA-NAMES.
       01  COPY IDMS SUBSCHEMA-RECORDS.
       LINKAGE SECTION.
       01  DB-PARM.
           02  DBNAME-IN    PIC X(8).
           02  FILLER       PIC X.
           02  DBNODE-IN    PIC X(8).
           02  FILLER       PIC X.
           02  ACTION-CD    PIC X.
               88    ACT-BIND   VALUE 'R'.
               88    ACT-BINDU  VALUE 'U'.
               88    ACT-DML1   VALUE '1'.
               88    ACT-DML2   VALUE '2'.
               88    ACT-DML3   VALUE '3'.
               88    ACT-UPDT   VALUE '4'.
               88    ACT-FIN    VALUE 'F'.
               88    ACT-TCOM   VALUE 'C'.
               88    ACT-RCOM   VALUE 'D'.
               88    ACT-TFIN   VALUE 'X'.
               88    ACT-TBAK   VALUE 'B'.
               88    ACT-RBAK   VALUE 'Y'.
           02  RETURN-CD    PIC S9(8) COMP.
           ...
01  COPY IDMS SUBSCHEMA-CTRL.
       PROCEDURE DIVISION USING DB-PARM, SUBSCHEMA-CTRL.
       MAINLN SECTION.
           MOVE 0 TO RETURN-CD.
           ...
```

```
           IF ACT-BINDU
              PERFORM BIND-IT
           ELSE IF ACT-RCOM
              PERFORM RCOM-IT
           ELSE IF ACT-TFIN
              PERFORM TFIN-IT
           ELSE IF ACT-TBAK
              PERFORM TBAK-IT
           ELSE IF ACT-RBAK
              PERFORM RBAK-IT
           ELSE IF ...
           ...
           ELSE
              MOVE 32 TO RETURN-CD.
           GOBACK.
       BIND-IT SECTION.
           MOVE SPACES TO SUBSCHEMA-CTRL.
           MOVE 'MBINDSUB' TO PROGRAM-NAME.
           BIND RUN-UNIT DBNODE DBNODE-IN
                         DBNAME DBNAME-IN.
           READY USAGE-MODE UPDATE.
           PERFORM CHECK-STAT.
           BIND EMPLOYEE.
           PERFORM CHECK-STAT.
           BIND DEPARTMENT.
           PERFORM CHECK-STAT.
       ...
       TCOM-IT SECTION.
           COMMIT TASK.
           PERFORM CHECK-STAT.
       RCOM-IT SECTION.
      *  Issue RRS Commit_UR
           CALL 'SRRCMIT' USING RETURN-CD.
           PERFORM CHECK-RRS.
       TFIN-IT SECTION.
           FINISH TASK.
           PERFORM CHECK-STAT.
       RBAK-IT SECTION.
      *  Issue RRS Backout_UR
           CALL 'SRRBACK' USING RETURN-CD.
           PERFORM CHECK-RRS.
       ...
```

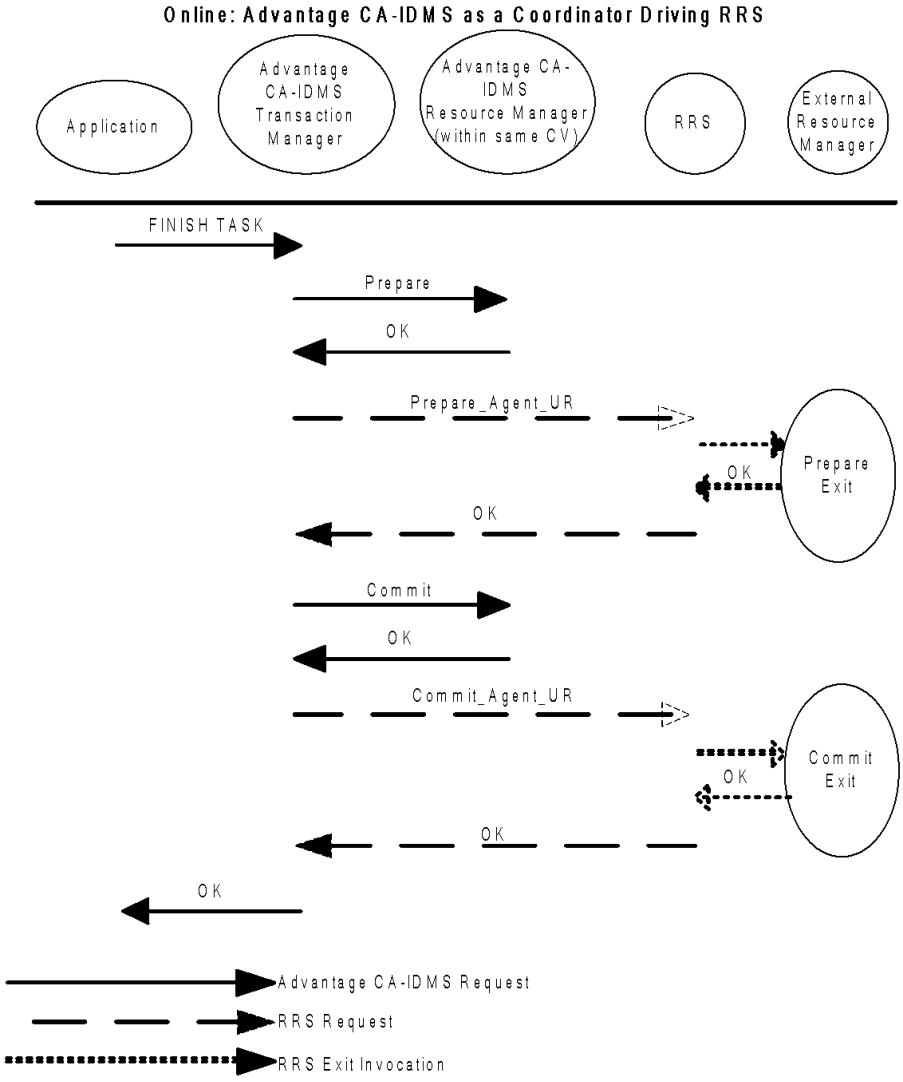
3.9.4 RRS Support for Online Applications

RRS can be used by an online application to ensure that updates made through external resource managers such as MQSeries are coordinated with those of Advantage CA-IDMS. In order to exploit this functionality, the external resource manager must be accessed through its RRS-enabled interface.

Before accessing the external resource manager, the online task must establish a private RRS context. This context can then be passed to any external resource manager that wants to participate in the Advantage CA-IDMS controlled transaction. Typically, online support for accessing external resources is provided by a third party vendor and, consequently, it is the vendor's responsibility to establish the private context and ensure that it is available to the external resource manager's RRS-enabled interface.

The RRS-enabled interface passes the context to its resource manager so that it can register an interest in the context's UR.

To initiate a commit operation involving all interested resource managers, the online application issues an Advantage CA-IDMS commit DML command (such as a FINISH TASK or a COMMIT WORK). The local transaction manager then uses RRS as an agent to coordinate its updates with those of the external resource managers.



3.9.4.1 Example

Consider an online application that accesses Advantage CA-IDMS and MQSeries and wishes to coordinate the work done on each. To do this, a private context (referred to as CTXPRIV) is first created by calling IDMSIN01. MQSeries is then accessed through its RRS-enabled interface, specifying CTXPRIV. When the transaction is committed through a DML command such as FINISH TASK, the Advantage CA-IDMS transaction manager becomes the coordinator and drives RRS as a participant. RRS in turn directs the actions of MQSeries in support of the commit operation.

3.9.4.2 Programming Interface

The following IDMSIN01 function allows private context manipulation. It is designed for third party vendors who want to exploit the two-phase commit functionality.

```
label    IDMSIN01 RRSCTX,                                          X
               RRSFUNA=rrs-function-address,RRSCTXA=rrs-context-address
```

3.9.4.3 Parameters

RRSCTXA=rrs-context-address

Specifies the address of a 16-byte field for the RRS context token. Depending upon the function, this field is input, output, or both.

RRSFUNA=rrs-function-address

Specifies the address of a 1-byte field that contains the function to execute. Valid function values and their return codes are:

- X'01': Get RRS context.

  Return codes:

  - 00 — An RRS context exists; the field pointed to by RRSCTXA contains the current RRS context.
  - 04 — No RRS context exists; the field pointed to by RRSCTXA is cleared.
  - Any other return code — An internal error occurred. The content of the field pointed to by RRSCTXA is undefined.

- X'02': Set RRS context. If the field pointed to by RRSCTXA contains binary zeros, a new RRS context is created and returned; if the field is not binary zeros, it must contain an RRS context token which is saved by the Advantage CA-IDMS transaction manager. No attempt is made to validate the RRS context token.

  Return codes:

  - 00 — The RRS context token was successfully saved by the Advantage CA-IDMS transaction manager.
  - Any other return code — An error occurred. Return codes 103-107, 301, 701, 756, F00, and FFF are from context services. Their description can be found in the IBM guide *MVS Programming: Resource Recovery* in the specific topic "Begin_Context."

- X'03': End RRS context. The field pointed to by RRSCTXA must contain the token of the RRS context to be ended.

  Return codes:

  - 00 — The RRS context was successfully terminated. The field pointed to by RRSCTXA is set to binary zeros.
  - Any other return code — An error occurred. Return codes 103-107, 360-369, 703, 756, and FFF are from context services. Their description can be found in the IBM guide *MVS Programming: Resource Recovery* in the specific topic "End_Context."

3.9.4.4 Application Design Considerations

The private context created by a call to IDMSIN01 is terminated when the transaction is ended. Therefore, after a commit or rollback operation, another context must be created through a call to IDMSIN01 before another request can be made of the external resource manager.

3.9.5 Optimizations Supported

To decrease the cost of a syncpoint operation using RRS, Advantage CA-IDMS supports the RRS only-agent and read-only exit minimization optimizations. For more information on RRS optimizations, see the IBM manual, *MVS Programming: Resource Recovery*.

The RRS **only-agent** optimization permits RRS to make a single phase commit request rather than separate Prepare and Commit requests, provided there is only one resource manager participating in the transaction at the time that the syncpoint operation is initiated. This optimization not only reduces communications between RRS and a central version, but also reduces both log and journal overhead.

The RRS **read-only exit minimization** optimization reduces the number of communications with a central version provided that it performed no updates within the RRS Unit of Recovery (UR) being committed.

3.9.6 Resynchronization Between RRS and Advantage CA-IDMS

Resynchronization is a process in which information is exchanged between a two-phase commit coordinator and a participant to establish attributes relevant to the two-phase commit process and complete outstanding distributed transactions following a failure.

Depending on the nature of the failure, resynchronization may occur automatically or may require explicit action to be triggered. This section focuses on resynchronization between RRS and an Advantage CA-IDMS system.

3.9.6.1 When Does It Occur?

Resynchronization between RRS and an Advantage CA-IDMS system occurs:

- When a central version is started, as part of registering with RRS.
- When resynchronization is manually driven through a DCMT VARY DISTRIBUTED RESOURCE MANAGER command. See A.16, “DCMT VARY DISTRIBUTED RESOURCE MANAGER” on page A-25.

3.9.6.2 What Does It Entail?

Resynchronization begins with validation of the LOG names: both the name with which RRS knows the Advantage CA-IDMS system (the Advantage CA-IDMS log name) and the RRS log name as known to the Advantage CA-IDMS system (the RRS log name).

The Advantage CA-IDMS log name has the following format:

```
IDMS.RM.jrnlstamp.nodename.CA
```

Where:

- *Jrnlstamp* is the central version's 26-character journal timestamp with dashes ("-") replaced by underscores ("_"). This value is assigned by a central version the first time it opens a set of journal files after they have been formatted.
- *Nodename* is the central version's node name, padded with underscores ("_") if it is less than eight characters in length.

The following messages are displayed during the resynchronization process:

```
DC224002 V73 T23 RRS log name ATR.B8909786A2A8AA40.IBM
DC224002 V73 T23 Resource Manager log name IDMS.LOG.yyyy_mm_dd_hh.mm.ss.ssssss.nodename.CA
DC224006 V73 T23 Resynchronization with RRS complete
```

If no distributed transactions involving the two systems exist at the time that resynchronization takes place, then the two systems simply accept each other's LOG names.

If distributed transactions involving the two systems do exist at the time of resynchronization, then the LOG names are compared. If they are the same, resynchronization proceeds by exchanging information about the incomplete distributed transactions that are pending resynchronization. If the LOG names are not the same, it indicates that one of the following has occurred:

- The RRS LOG has been prematurely formatted.
- RRS has been started with incorrect LOG files.
- The Advantage CA-IDMS system's journal files have been prematurely formatted.
- The Advantage CA-IDMS system was started with incorrect journal files.

Any of these conditions result in a resynchronization failure.

3.9.6.3 Responding to Resynchronization Failures

If resynchronization detects a LOG name mismatch and incomplete distributed transactions exist, resynchronization cannot complete. When this occurs, check whether RRS and the Advantage CA-IDMS system were started with correct log and journal files. If they were not, correct the situation. If premature formatting is the cause of the resynchronization failure, the incomplete transactions must be manually completed:

- If the RRS LOG was formatted, complete the transactions once the central version is up and running. For more information on how to do this, see 3.7.3, “Completing Transactions Manually” on page 3-27.
- If the Advantage CA-IDMS journal files were formatted, use the RRS ISPF panels to complete the transactions. For more information on RRS panels, see the IBM manual *MVS Programming: Resource Recovery*.

Chapter 4. SQL Features

4.1 Overview

Release 16.0 provides the following new SQL features that are described in this chapter:

- Dynamic SQL caching
- SQL-defined database enhancements
- SQL productivity enhancements
- Enhanced compatibility with Open Standards
- XML Publishing

4.2 Dynamic SQL Caching

Release 16.0 provides a dynamic SQL caching feature that dramatically improves runtime performance when you are executing a dynamic SQL statement. This benefits web access to data using the ODBC and JDBC drivers in the Advantage CA-IDMS Server and the Advantage™ EDBC products.

Dynamic SQL caching is a common technique used to improve performance in an SQL environment. Caching works in the following manner: when a dynamic SQL statement is compiled, a copy of the SQL statement and the result of the SQL compilation are saved in a cache. For each subsequent SQL compilation request, the cache is searched. If the statement is found, the matching compiled structures are used instead of recompiling the statement. This improves performance by eliminating the I/O requests to read the catalog and the CPU usage required to invoke the SQL optimizer for subsequent executions of the same dynamic SQL statement.

In most cases, the savings in resource consumption due to bypassing the SQL compilation are significantly greater than the extra cost associated with caching the SQL source, access plans, and related structures.

4.2.1 Searching the Cache

When a search is made in the cache for a matching SQL statement, a *cache hit* occurs when a matching entry is found. The following factors are considered in determining whether an SQL statement matches a cache entry:

- The text of the statement
- The default schema in effect for the SQL session
- The dictionary to which the SQL session is connected
- The presence of temporary table references within the statement

A literal comparison of the statement's text is made against each cache entry until a match is found. A literal comparison avoids the overhead of parsing but has the consequence that an entry may not match because of differences in such things as case and spacing. For example, the following three statements are different if using a literal comparison:

```
Select * from EMPLOYEE

Select * from   EMPLOYEE

select * from employee
```

Specifying values as literals instead of as dynamic parameters can also result in unequal comparisons. The following two statements would be textually identical if a dynamic parameter had been used in place of the numeric values 100 and 101:

```
select * from DEMOEMPL.EMPLOYEE where EMP_ID = 100

select * from DEMOEMPL.EMPLOYEE where EMP_ID = 101
```

Note: While the use of dynamic parameters can increase the frequency of finding a matching cache entry, it may occasionally result in a less efficient access strategy than one chosen for a specific value.

When a dynamic statement that relies on a default schema is cached, both the statement text and the default schema are saved. When the cache is searched for a statement that relies on a default schema, both the statement's text and the session's default schema must be equal to their cached equivalents in order for the entry to match. Consider the following two statements. The first matches a cached entry regardless of the default schema in effect for the SQL session. The second matches only if the default schema in effect for the SQL session is the same as that in the cache:

```
select * from DEMOEMPL.EMPLOYEE
select * from EMPLOYEE
```

The name of the dictionary to which an SQL session is connected is always saved in the cache and compared to the session's dictionary during a search of the cache. If the two are not the same, then the cache entry does not match.

If an SQL statement references a temporary table, it is not cached since each temporary table instance can be structurally different from others of the same name. Therefore, no statement that references a temporary table will match a cache entry.

4.2.2 Impact of Database Definition Changes

Database definition changes may or may not be detected automatically based on whether the database is SQL-defined or non-SQL defined. This has consequences for dynamic SQL caching as explained below.

4.2.2.1 SQL-Defined Databases and Caching

Because SQL-defined databases have an associated catalog and because areas for SQL-defined databases have timestamps, Advantage CA-IDMS is able to automatically detect definition-based changes that impact cached SQL statements. Whenever a statement needs recompilation, Advantage CA-IDMS automatically detects this condition and recompiles the affected statement dynamically.

4.2.2.2 Non-SQL Defined Databases and Caching

Non-SQL defined databases do not have timestamps for automatically determining whether a database's definition accurately describes the underlying data. Consequently, when changing the structure of a non-SQL defined database, it is the administrator's responsibility to ensure that all SQL statements impacted by the change are recompiled. If dynamic SQL caching is not used, then this entails recompiling access modules that reference the affected database. If dynamic SQL caching is used, then it

also entails purging the cache of statements that reference the affected database. This can be done by deleting rows from the SYSCA.DSCCACHE or SYSCA.DSCCACHEV tables. For more information on these tables, see Appendix E, "SQL Cache Tables."

It is also recommended that dynamic SQL caching be disabled during the transition period in which the definition-based changes are being implemented. For information on how to do this, see 4.2.3, "Controlling the Cache."

Advantage CA-IDMS detects the need to recompile cached SQL statements if a change is made to the referencing SQL schema through which a non-SQL defined schema is referenced. It does this by comparing the update stamp of the referencing SQL schema to the compile stamp of the cached statement.

4.2.3 Controlling the Cache

There are various ways that an individual user and a DBA can control dynamic SQL caching. Three ways are discussed below:

- Establishing caching attributes for an individual SQL session by issuing a SET SESSION statement
- Establishing default caching attributes for a central version through a system generation SQL CACHE statement
- Establishing default caching attributes for a local mode job by specifying a SYSIDMS SQL_CACHE_ENTRIES parameter

For information on the various tables that control caching and examples of ways to display and control the cache using SQL, see Appendix E, "SQL Cache Tables."

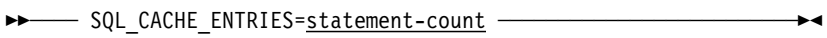
4.2.3.1 SET SESSION Statement

Since there may be occasions when the cost of dynamic SQL caching outweighs its benefit, the SET SESSION statement has been enhanced to allow control over caching within an individual SQL session. For a description of the new syntax, see Appendix B, "New and Revised SQL Statements."

4.2.3.2 SYSIDMS SQL_CACHE_ENTRIES Parameter

Administrators and batch users can control SQL caching in local mode with the following new SYSIDMS parameter.

Syntax



Parameters

SQL_CACHE_ENTRIES

Specifies the maximum number of SQL statements that can be placed in the SQL statement cache. Specify 0 to disable caching.

statement-count

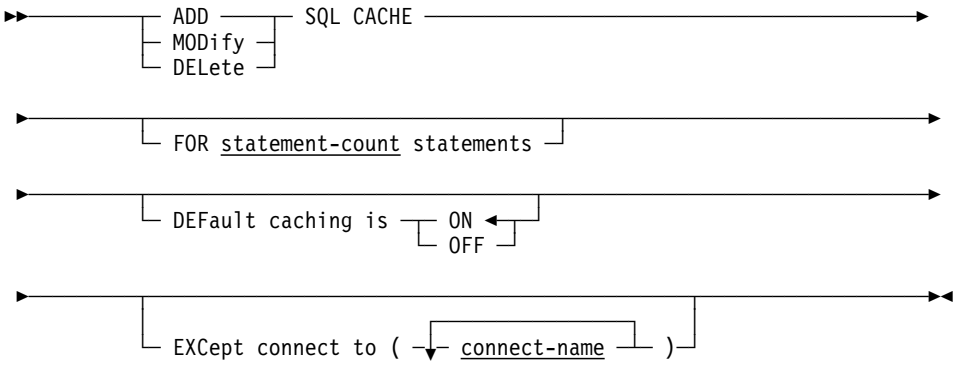
A numeric value ranging from 0 to 2,147,483,647. The default value is 200. The maximum value depends on available memory.

A site can establish a different default value for *statement-count* by creating a SYSIDMS load module and using an SQL_CACHE_ENTRIES parameter to specify the desired default value.

4.2.3.3 System Generation SQL CACHE Statement

In a central version, SQL caching is controlled through a new system generation SQL CACHE statement.

Syntax



Parameters

statement-count

Specifies the maximum number of SQL statements that can be placed in the SQL statement cache. The default value is 100. The maximum theoretical value is 2,147,483,647. The actual maximum depends on available memory.

DEFault caching

ON

Specifies that caching of dynamic SQL statements is enabled. ON is the default.

OFF

Specifies that caching of dynamic SQL statements is disabled.

connect-name

Specifies the name of a dictionary or catalog to which a user of the CV can connect. You can specify multiple *connect-names* to form an exception list to the default caching specification.

Usage:

Dynamically changing caching attributes: All of the options that can be specified in an SQL CACHE statement can be changed dynamically by issuing SQL DML statements against CA-supplied tables. For more information, refer to Appendix E, "SQL Cache Tables."

Default caching status: If an SQL CACHE statement is not specified for a system, dynamic SQL caching is disabled at system startup. SQL caching can be dynamically enabled by inserting a row into the SYSCA.SQLCACHEOPT table. For more information, refer to Appendix E, "SQL Cache Tables."

Specifying an exception list: You can specify an exception list of *connect-names* for which caching is enabled or disabled depending on what was implicitly or explicitly specified in the DEFAULT CACHING clause. If default caching is enabled, caching is disabled for any session connected to a dictionary or database whose name appears in the exception list. Conversely, if default caching is disabled, caching is enabled for any such session.

System currency: Before issuing an SQL CACHE statement, you must establish currency on the target system to be modified.

4.3 SQL-Defined Database Enhancements

Release 16.0 includes the following SQL-defined database enhancements:

- Logical/physical separation
- Database cloning
- Stamp synchronization

These database enhancements are described in the sections that follow.

4.3.1 Logical/Physical Separation

Release 16.0 provides logical/physical separation for SQL-defined databases. Logical/physical separation enables multiple instances of identically defined databases to be represented by a single schema and accessed through a single set of access modules. In so doing, it can significantly reduce the effort involved in administering certain environments, such as the following:

- A development environment in which several copies of a test database need to be maintained (possibly hundreds of copies, one for each developer). Without logical/physical separation, the database administrator must maintain multiple schema definitions as well as multiple sets of access modules, one for each instance of the database.
- A staged implementation environment, in which access modules that have already passed quality assurance testing can be moved into production without recompilation.
- A production environment, in which multiple segments of a production database can be accessed through a single set of access modules, with the target segment determined by the database to which the SQL session connects.

The remainder of this section describes how to implement logical/physical separation for SQL-defined databases and considerations associated with its use.

4.3.1.1 Implementing Logical/Physical Separation

In order to implement logical/physical separation for SQL-defined databases, you must create a referencing schema. Release 16.0 extends the CREATE SCHEMA command so that it can reference another SQL schema in the same way that it can reference a non-SQL schema. Any SQL-defined schema can be referenced by another schema except for schemas that:

- Are themselves referencing schemas
- Include constraints that reference tables in another schema
- Include tables that are referenced by constraints in another schema

Once a referencing schema is defined, any base table or routine (procedure, table procedure, or function) defined in the referenced schema is automatically accessible as

an entity in the referencing schema. Views, however, are not. For more information, see 4.3.1.3, “Views and Logical/Physical Separation” on page 4-10.

The referencing schema can be bound to a specific database instance or unbound by not specifying a DBNAME as part of the referencing schema definition. Accessing tables through an unbound referencing schema allows runtime determination of the database instance to be accessed based on the database to which an SQL session connects. Therefore, the same table names (and access modules) can be used to access different database instances simply by connecting to different DBNAMEs, provided those DBNAMEs include the appropriate database segment to be accessed.

For more information on referencing an SQL schema, see CREATE SCHEMA and ALTER SCHEMA statements in Appendix B, “New and Revised SQL Statements.”

4.3.1.2 Changing a Referenced or Referencing Schema

If a change is made to one or more tables in the referenced SQL schema or the referencing schema is changed to refer to a different SQL schema or DBNAME, affected access modules are recompiled automatically when they are next used. Manual recompilation is not necessary as is the case if reference is made to a non-SQL schema.

However, views that reference tables through a referencing schema require manual redefinition if changes are made to the referenced or referencing schema. In order to determine which views are affected, you can use the DISPLAY ALL VIEW statement with the REFERENCED option. For example, the following statement displays all views that access a table in schema FIN:

```
DISPLAY ALL VIEWS WHERE REFERENCED TABLE SCHEMA NAME = 'FIN'
```

In order to redefine these views, you must drop and recreate them. Before dropping them, you can use the DISPLAY/PUNCH VIEW statement to generate the necessary syntax to recreate them.

4.3.1.3 Views and Logical/Physical Separation

Logical/physical separation impacts the use of views. If you wish to define a view that can be used to access different database instances, then it must be defined in a schema that is separate from both the referencing and referenced schemas. The view should access tables through the referencing schema and may join the results with tables in other schemas.

Suppose that there is a referencing schema called FINANCE that references an SQL schema called FINBASE. In order to define a view that is independent of a specific financial database instance, it must be defined in a third schema (CORP) and reference financial base tables only through the FINANCE referencing schema as illustrated below:

```
CREATE VIEW CORP.BUDGET AS SELECT ... FROM FINANCE.BUDGET ...
```

4.3.2 Database Cloning

Release 16.0 allows you to explicitly specify physical attributes whose values would otherwise be automatically assigned when creating or altering SQL-defined entities such as tables, procedures, and indexes. The ability to specify such physical attributes, including the synchronization stamp that is used to detect definitional changes, enables a DBA to create and maintain identically defined databases. This can be useful in situations such as the following:

- Taking a snapshot copy of a production database for testing purposes
- Implementing database segmentation so that multiple segments can be accessed through a single referencing schema and set of access modules
- Restoring a back-version of a database and its definition

The following DDL statements have been enhanced to enable the specification of these physical attributes:

- CREATE AREA
- ALTER AREA
- CREATE TABLE
- ALTER TABLE
- CREATE VIEW
- CREATE TABLE PROCEDURE
- ALTER TABLE PROCEDURE
- CREATE PROCEDURE
- ALTER PROCEDURE
- CREATE FUNCTION
- ALTER FUNCTION
- CREATE INDEX New DISPLAY and PUNCH options allow you to generate syntax for these physical attributes. The new FULL PHYSICAL option generates syntax for all attributes of an entity including physical attributes such as table IDs and synchronization stamps. The new WITH TIMESTAMP option generates only the syntax for specifying a synchronization timestamp.

The following statements have been modified to support these new options:

- DISPLAY SCHEMA
- PUNCH SCHEMA
- DISPLAY TABLE
- PUNCH TABLE
- DISPLAY VIEW

- PUNCH VIEW
- DISPLAY TABLE PROCEDURE
- PUNCH TABLE PROCEDURE
- DISPLAY PROCEDURE
- PUNCH PROCEDURE
- DISPLAY FUNCTION
- PUNCH FUNCTION
- DISPLAY INDEX
- PUNCH INDEX

The enhanced CREATE and ALTER AREA statements are described below. For more information on the other statements, see Appendix B, “New and Revised SQL Statements.”

4.3.2.1 Specifying Synchronization Timestamps

While the ability to specify physical attributes can be useful in certain situations, it should be used with care. If you change the value of a synchronization timestamp, you can disable the ability for the database engine to detect definitional changes. This could result in data corruption because an out-of-date access module updates the database.

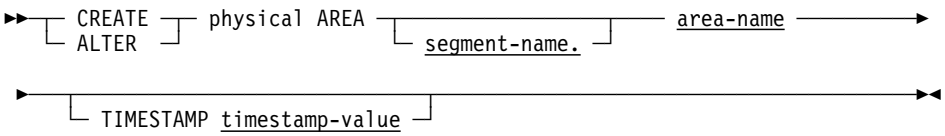
At a minimum, you should ensure that every version of an entity's definition has a unique synchronization timestamp associated with it. You should also be aware that while some entities, such as indexes and constraints, do not have an associated timestamp, changing their definition is, in effect, changing the definition of their associated table(s) and must also result in a unique synchronization stamp value.

If a table resides in an area that is controlled by area-level synchronization stamps, you must update the area's synchronization timestamp. Updating the table's synchronization stamp is optional but recommended. If a table resides in an area that is controlled by table-level synchronization stamps, you must update the table's stamp and cannot update that of the area.

4.3.2.2 Specifying Table and Index IDs

It is not always possible to create a table with a specific table ID or an index with a specific index ID. You are able to do so only if the value specified is not assigned to another table or index in the same area. Consequently, manipulation of physical attributes is generally only appropriate for schemas that define the entire contents of a database area or segment.

4.3.2.3 CREATE/ALTER AREA Statement Syntax



4.3.2.4 Parameters

TIMESTAMP
Specifies the value of the synchronization stamp to be assigned to the area.

timestamp-value
A valid external representation of a timestamp. This clause is valid only for areas for which area-level stamping is in effect.

4.3.3 Stamp Synchronization

Release 16.0 provides two new utility functions to allow a DBA to manipulate synchronization timestamps. The new SYNCHRONIZE STAMPS utility lets you compare stamps in the data area and the catalog and to update one from the other. A new option of the INSTALL STAMPS utility allows you to replace existing synchronization stamps in an area with the values from the catalog.

These new facilities are provided as an alternative mechanism for taking snapshot copies of identically defined databases and also as an aid in recovery situations in which either the catalog or a data area must be restored independently of the other. Each of these utility enhancements is described below.

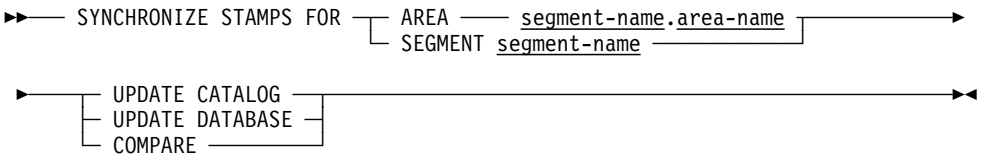
4.3.3.1 SYNCHRONIZE STAMPS Utility

Purpose: This new utility statement manipulates SQL synchronization stamps in the following ways:

- Displays and compares the stamps in the catalog and the data area(s), issuing a warning if stamps are inconsistent
- Updates the catalog with the stamps from the data area(s)
- Updates the data area(s) with the stamps from the catalog

Authorization: You must hold DBAWRITE authority on every area to be processed by the SYNCHRONIZE STAMPS utility.

Syntax



Parameters

AREA
Specifies the area in which to synchronize stamps.

segment-name
Specifies the name of the segment associated with the area.

area-name
Specifies the name of an area included in the DMCL module.

SEGMENT
Specifies the segment whose areas will have their stamps synchronized.

segment-name
Specifies the name of a segment included in the DMCL module.

UPDATE CATALOG
Specifies that the catalog is to be updated with the stamps from the data area(s).

UPDATE DATABASE
Specifies that the data area(s) are to be updated with stamps from the catalog.

COMPARE
Displays and compares the stamps in the catalog and the data area(s) and issues a warning if the stamps are inconsistent.

Usage:

How to submit a SYNCHRONIZE STAMPS statement: You submit a SYNCHRONIZE STAMPS statement through OCF or IDMSBCF.

Use Caution When Updating Stamps: By using the SYNCHRONIZE STAMPS utility to update stamps in a catalog or data area, you are asserting that the definition in the catalog accurately describes data in the area. You should be sure that this is true before updating stamp values. No data validation is performed by the utility.

Example: The following statement compares the synchronization stamps in the USERDB.EMP_AREA with those in the catalog:

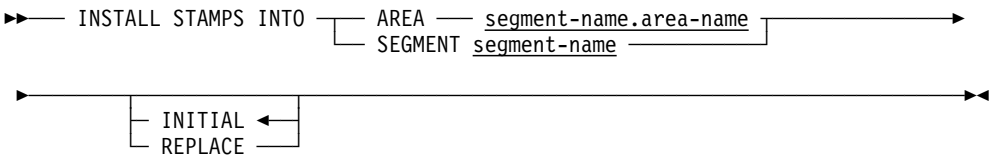
```
SYNCHRONIZE STAMPS FOR AREA USERDB.EMP_AREA COMPARE;
*+ Status = 0        SQLSTATE = 00000
                        *** Current Stamps Report ***
  Area:USERDB.EMP_AREA                  Table Stamping
      Catalog Stamp:  <null>
      Database Stamp: <null>

         Table ID:1024    Table:DEMO.EMPL
             Catalog Stamp:  yyyy-mm-dd-hh.mm.ss.ssssss
             Database Stamp: yyyy-mm-dd-hh.mm.ss.ssssss

         Table ID:1025    Table:DEMO.POSITION
             Catalog Stamp:  yyyy-mm-dd-hh.mm.ss.ssssss
             Database Stamp: yyyy-mm-dd-hh.mm.ss.ssssss

         Table ID:1026    Table:DEMO.MANAGERS
             Catalog Stamp:  yyyy-mm-dd-hh.mm.ss.ssssss
             Database Stamp: yyyy-mm-dd-hh.mm.ss.ssssss

         Table ID:1027    Table:INV.PART
             Catalog Stamp:  yyyy-mm-dd-hh.mm.ss.ssssss
             Database Stamp: yyyy-mm-dd-hh.mm.ss.ssssss

         Table ID:1028    Table:INV.COMPONENT
             Catalog Stamp:  yyyy-mm-dd-hh.mm.ss.ssssss
             Database Stamp: yyyy-mm-dd-hh.mm.ss.ssssss

         Table ID:1029    Table:EMP.T5
             Catalog Stamp:  yyyy-mm-dd-hh.mm.ss.ssssss
             Database Stamp: yyyy-mm-dd-hh.mm.ss.ssssss

         Table ID:1030    Table:LRD.EMPL
             Catalog Stamp:  yyyy-mm-dd-hh.mm.ss.ssssss
             Database Stamp: yyyy-mm-dd-hh.mm.ss.ssssss

         Table ID:1031    Table:JPD.T5
             Catalog Stamp:  yyyy-mm-dd-hh.mm.ss.ssssss
             Database Stamp: yyyy-mm-dd-hh.mm.ss.ssssss
```

4.3.3.2 INSTALL STAMPS Utility

Purpose: This utility stores synchronization stamps in an SQL-defined area. In Release 16.0, this utility has been enhanced to allow replacing stamps if they already exist.

This section describes only the new parameter of this statement. For more information, see the *Advantage CA-IDMS Utilities*.

Syntax



Parameters

INITIAL

Specifies that the area(s) contain no synchronization stamps because they were formatted using the file or segment option of the FORMAT utility statement executing in local mode. INITIAL is the default.

REPLACE

Specifies that the area(s) contain synchronization stamps that should be replaced with those from the catalog.

Usage:

Use Caution When Replacing Stamps: By replacing stamps in an area, you are asserting that the catalog's definition accurately describes data in the area. You should be sure that this is true or that the area contains no data before replacing stamp values. No data validation is performed by the utility.

4.4 SQL Productivity Enhancements

Release 16.0 includes the following SQL productivity enhancements:

- User-defined SQL functions
- SQL procedures written as Advantage CA-ADS mapless dialogs
- Database name inheritance
- ROWID pseudo-column
- Transaction sharing

These SQL productivity enhancements are described in the sections that follow.

4.4.1 User-Defined SQL Functions

Release 16.0 provides the ability for sites to define and invoke their own scalar SQL functions. This new support is a subset of the SQL standard specification for external functions. The function can have zero or more input parameters and must return a single value. Such user-defined functions allow encapsulation and reuse of business logic from within the SQL language, thus enabling the use of that logic from any environment in which SQL can be issued.

To take advantage of this feature, follow the steps below:

- Define the function using the new CREATE FUNCTION statement.
- Write the function in COBOL, PL/I, Assembler, or Advantage CA-ADS following the guidelines given in Appendix C, "SQL Functions and SQL Procedure Enhancements." You may be able to use an existing program as a template for a function.
- If necessary, define the function to an Advantage CA-IDMS system.
- Invoke the function as needed by specifying it anywhere that a *value-expression* can be specified in an SQL statement, except in the check constraint of a table definition.

Note: The number of user-defined function invocations and subqueries in a statement, including those in referenced views, must not exceed 1024. The maximum number of arguments in a user-defined function invocation is around 620 and depends on the datatypes and the complexity of the expressions used in the function invocation.

For the new CREATE FUNCTION DDL command syntax, see Appendix B, "New and Revised SQL Statements." For a comprehensive discussion and examples on defining, using, and writing functions, see Appendix C, "SQL Functions and SQL Procedure Enhancements."

4.4.2 Procedures and Functions Written as Advantage CA-ADS Mapless Dialogs

In Release 16.0, you can code an SQL procedure or an SQL function as an Advantage CA-ADS mapless dialog. Use the protocol clause on the following SQL statements to specify that the procedure or function is coded in Advantage CA-ADS.

- CREATE PROCEDURE
- CREATE FUNCTION

4.4.2.1 Protocol Clause

The syntax for the protocol clause in the CREATE statements is PROTOCOL IDMS/ADS. There is no default and the protocol is required.

You must specify IDMS for SQL procedures or functions that are written in COBOL, PL/I, or Assembler, and ADS for SQL procedures or functions that are written in Advantage CA-ADS. The name of the dialog that is loaded and run when the SQL procedure or function is invoked is specified in the EXTERNAL NAME clause of the CREATE/ALTER PROCEDURE or CREATE/ALTER FUNCTION statements. If the protocol is set to ADS, you *must* set the mode clause to SYSTEM. (See the examples provided in Appendix C, “SQL Functions and SQL Procedure Enhancements.”)

The value of the protocol for procedures and functions is stored in the SQL catalog in the COMPRESS column of SYSTEM.TABLE. The IDMS protocol is encoded as I; the ADS protocol as A.

For more information on the CREATE FUNCTION and CREATE PROCEDURE statements, see Appendix B, “New and Revised SQL Statements.”

4.4.2.2 Mapless Dialog

The Advantage CA-ADS dialog that implements the SQL procedure or function must be mapless. To return to the SQL engine, the Advantage CA-ADS premap process must issue a LEAVE ADS command.

4.4.2.3 Work Records

To access the procedure or function parameters, the dialog must include a work record whose name is <schema>.<procedure_name> or <schema>.<function_name>. This record is not read from the dictionary but instead is automatically constructed by the Advantage CA-ADS dialog compiler (ADSC or ADSOBCOM) when it compiles the dialog. You can refer to the procedure and function parameters and the corresponding null indicators in the Advantage CA-ADS process code in the same way as you refer to columns in any SQL table.

Within the function, the value to be returned must be moved to USER_FUNC data element. The datatype of this data element is automatically defined in accordance with the RETURNS <datatype> clause of the SQL function definition.

When parameters of a procedure or function are dropped, added, or altered, the dialog that implements the procedure or function must be recompiled. Failure to do so may result in a DC171066 error message when the procedure is next executed. The runtime validation producing this message is based solely on the size of the record.

4.4.2.4 Additional Records

Besides the above pseudo-work record, other records related to the procedure or function can be included.

ADSO-SQLPROC-COM-AREA is a system-supplied record. The record layout is given below:

```
ADD RECORD NAME ADS0-SQLPROC-COM-AREA.
  03 FILLER            PIC S9(8) COMP SYNC.
  03 FILLER    PIC X(3).
  03 SQLPROC-SQLSTATE  PIC X(5).
  03 SQLPROC-NAME      PIC X(18).
  03 SQLPROC-SPECIFIC-NAME     PIC X(8).
  03 SQLPROC-MSG-TEXT  PIC X(80).
  03 SQLPROC-COMMAND-CODE      PIC S9(8) COMP SYNC.
  03 SQLPROC-OPERATION-CODE    PIC S9(8) COMP SYNC.
  03 SQLPROC-INSTANCE-ID       PIC S9(8) COMP SYNC.
  03 FILLER            OCCURS 2.
```

The non-FILLER elements of the ADSO-SQLPROC-COM-AREA record are the parameters that are common to all SQL procedures and functions. For a description of these parameters, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

If the procedure or function definition contains a LOCAL or GLOBAL WORKAREA clause, you can define corresponding records in the dictionary. While the layout of these records is application dependent, the name must comply with the following rules in order for the Advantage CA-ADS runtime to properly initialize these records:

- <dialog-name>-SQLPROC-GLOBAL-AREA
- <dialog-name>-SQLPROC-LOCAL-AREA

<dialog-name> is the name of the dialog as specified in the name clause of the procedure or function definition. For more information and examples, see Appendix C, “SQL Functions and SQL Procedure Enhancements.”

4.4.3 Database Name Inheritance for Table Procedures, Procedures, and Functions

In Release 16.0, an SQL routine (a table procedure, procedure, or function) can inherit the current database name of the encompassing SQL session as a default. To control the inheritance, a new clause, DEFAULT DATABASE NULL/CURRENT, has been added to the following SQL statements:

- ALTER PROCEDURE
- ALTER FUNCTION

- ALTER TABLE PROCEDURE
- CREATE PROCEDURE
- CREATE FUNCTION
- CREATE TABLE PROCEDURE

In the new clause (DEFAULT DATABASE NULL/CURRENT), NULL is the default for the CREATE statements and guarantees compatibility with previous releases of Advantage CA-IDMS. CURRENT makes the CURRENT DATABASE the default database name for any subordinate database session started by the SQL routine.

For more information on using these SQL statements, see Appendix B, “New and Revised SQL Statements.”

4.4.4 ROWID Pseudo-Column

The ROWID pseudo-column feature provides unique access to a row in an SQL table or view. ROWID contains the db-key for an underlying database record. It is not persistent for the life of the database, but it can be used within a transaction or other controlled processes.

Although ROWID can be used for SQL-defined tables, it is most useful for updating non-SQL defined databases. Since such databases tend to have record types with no primary or foreign keys, identifying a specific row to be updated or deleted is often difficult. For such record types, it was often necessary to implement a table procedure to perform the update or deletion. The use of ROWID may eliminate the need for the table procedure, since it uniquely identifies each row of the non-SQL defined table.

ROWID pseudo-column has the following properties:

- Every base table has a ROWID pseudo-column associated with it. ROWID is defined automatically. Pseudo-columns are similar to, but not the same as, normal columns.
- The value of ROWID is unique for each row of a base table; however, you cannot consider it to be a table's primary key because its value can change over the lifetime of the database. This could happen after unloading and reloading the data.
- The value of ROWID does not change during an SQL transaction as long as the row is not deleted and reinserted.
- ROWID provides the fastest access to a row.
- The datatype of ROWID is TID (tuple id); it has a length of 8 bytes. The first 4 bytes are the db-key. The last 4 bytes are reserved for future use and are currently ignored.
- The value of ROWID can be null (for example, as the result of an outer join operation).
- You can select ROWID values, but you cannot insert or update them.
- The ROWID column is not defined in the catalog.

- Views also have a ROWID. The value of a view's ROWID is the ROWID of the first base table in the decomposition of the view from left to right. The ROWID of a view is not necessarily unique.

  For examples of the use of ROWID, see Appendix D, “SQL ROWID Examples.”

4.4.5 Transaction Sharing

Release 16.0 provides a new facility called transaction sharing, which allows multiple database sessions within a user session to share a single locking structure and recovery unit, thereby eliminating inter-session deadlocks. While not strictly related to SQL, it is primarily intended to facilitate the use of SQL to extend existing applications either by adding SQL to traditional applications or by using SQL routines (table procedures, procedures, and functions) to encapsulate business logic.

To illustrate how transaction sharing can assist in extending existing applications, consider an Advantage CA-ADS application that uses navigational DML to access data. An enhancement is planned in which the database is accessed using SQL instead of navigational DML. If the SQL statements access different portions of the database from that of the navigational requests, then intra-task deadlock is not an issue. If however, both types of DML access the same data and update it, there is a strong possibility of deadlock between the navigational and SQL database sessions. Transaction sharing can eliminate this deadlock potential by enabling the two sessions to share a single transaction.

Another area in which transaction sharing can benefit SQL users is in the development of SQL routines. Table procedures, in particular, are used extensively to overcome some of the limitations that SQL has in accessing non-SQL defined databases. They are also used to encapsulate and reuse business logic, making it accessible from many platforms. However, any access to a database from within a table procedure (or other SQL routine) brings with it the potential for deadlocking if the same data is directly accessed from within the encompassing SQL session. By having the routine and the encompassing SQL session all share a single transaction, the deadlock potential is eliminated.

4.4.5.1 Enabling Transaction Sharing

Transaction sharing can be enabled in the following ways:

- For an entire central version, through a new parameter on the system generation SYSTEM statement or through the new DCMT VARY TRANSACTION SHARING command.
- For all executions of a specific task, through a new parameter on the system generation TASK statement or a new option on the DCMT VARY TASK statement.
- For a batch job step, through a new SYSIDMS TRANSACTION_SHARING parameter.
- For an SQL routine, through a new parameter on a corresponding CREATE or ALTER DDL statement.

- Dynamically from within an application, through a call to IDMSIN01.

If transaction sharing is enabled for a system, it applies to all online tasks executing within that system unless overridden for an individual task. If transaction sharing is enabled for a task, it is initially enabled for all tasks of that type. If transaction sharing is enabled for an executing task or batch job step, it applies to all database sessions started by that task or job step unless dynamically overridden by a call to IDMSIN01 or by a procedure or function specification. Whether transaction sharing is enabled for a remote database session is determined by the front-end task or job step, not by the back-end task.

Regardless of how transaction sharing is enabled, if it is in effect at the time a new database session is started, then that database session is eligible to share its transaction with other database sessions started by the same task or user session. The following rules determine whether two sessions will share a transaction:

- A top-level database session will share its transaction with another top-level session if they are both eligible for transaction sharing. A *top-level database session* is one that is started by an application program rather than an SQL routine.
- A subordinate database session that is eligible for transaction sharing will share its parent's transaction even if the parent session is not eligible for transaction sharing. A subordinate database session is one that is started by an SQL routine.
- A system run unit will never share its transaction with another session.

Refer to the following for more information on enabling transaction sharing:

- For the new system generation SYSTEM parameter, see 4.4.5.3, “System Generation SYSTEM Statement” on page 4-24.
- For the new system generation TASK parameter, see 4.4.5.4, “System Generation TASK Statement” on page 4-24.
- For the new SYSIDMS parameter, see 4.4.5.5, “SYSIDMS TRANSACTION_SHARING Parameter” on page 4-25.
- For the new call to IDMSIN01, see 4.4.5.6, “IDMSIN01 Call” on page 4-25.
- For the new DCMT command and option, see Appendix A, “New and Revised DCMT Commands.”
- For the new SQL routine DDL parameter, see Appendix B, “New and Revised SQL Statements.”

4.4.5.2 Application Programming Considerations

Transaction sharing affects applications in the following ways:

- An update made through a database session can impact other database sessions sharing the same transaction.
- A rollback issued within one database session affects all sessions that share the same transaction.

- A commit issued by a database session whose transaction is shared has no effect on the transaction unless all other sharing sessions have also been committed.

Database sessions that share a transaction can impact each other in ways that would not be possible if transaction sharing were not in effect, since locking would otherwise prevent such interactions. For example, a record can be deleted by one database session while it is current of another database session that is sharing the same transaction. This can result in new and possibly unexpected error conditions. If a database session's currency is impacted by an update made through another database session, that currency is invalidated. If a subsequent DML request is issued that relies on the invalidated currency, an error is returned:

- For navigational DML, an error status of xx03 is returned to the application.
- For SQL, the application receives an SQLCODE of -4 (statement failure) and an SQLRSN of 1087 (conflicting activity within a shared transaction).

Before enabling transaction sharing for an application, you should ensure that affected programs handle these errors appropriately.

If multiple database sessions share a transaction and one of those sessions issues a rollback request, all changes made within the transaction are immediately rolled out. Other sessions sharing the transaction must issue their own rollback requests before issuing any other DML requests. Failure to do so results in an error:

- For navigational DML, the run unit is terminated and an error status of *xx*19 is returned to the application.
- For SQL, the application receives an SQLCODE of -5 (transaction failure) and an SQLRSN of 1088 (transaction forced to back out).

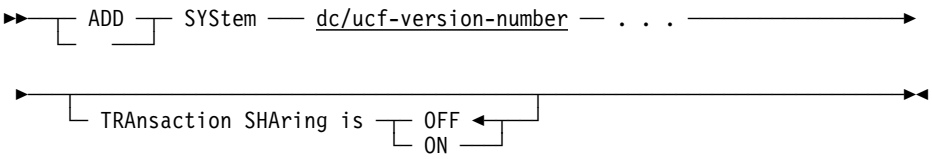
A task-level rollback is equivalent to issuing individual rollback requests for each shared or non-suspended database session associated with the task.

If multiple database sessions share a transaction, and one of those sessions issues a commit request, no changes are committed until all top-level sharing sessions that have had activity since the last commit, rollback, or start of transaction have issued a commit or until a task-level commit is issued. The term "commit" refers to any DML command that would normally result in committing changes (COMMIT CONTINUE, FINISH, COMMIT WORK RELEASE, and so on). Unless a commit continue request is issued (for which currency locks are retained), all currencies owned by the database session are immediately released; however, update and kept locks acquired by the database session remain until the transaction is committed, even if the request terminates the database session. A task-level commit has no effect on non-shared transactions if all associated top-level database sessions are suspended.

4.4.5.3 System Generation SYSTEM Statement

Use the system generation SYSTEM statement to establish the default transaction sharing option for all tasks within the system.

Syntax



Parameters

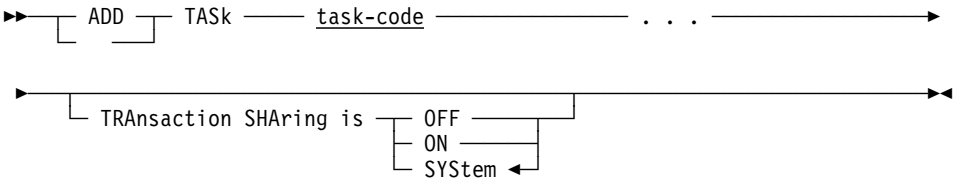
OFF
Specifies transaction sharing is disabled for all tasks in the system.

ON
Specifies transaction sharing is enabled for all tasks in the system.

4.4.5.4 System Generation TASK Statement

Use the system generation TASK statement to establish the initial transaction sharing option for all tasks of the given type.

Syntax



Parameters

OFF
Specifies that transaction sharing is initially disabled when a task of this type is initiated.

ON
Specifies that the transaction sharing is initially enabled when a task of this type is initiated.

SYStem
Specifies that the initial transaction sharing setting for a task of this type is the current system default setting.

4.4.5.5 SYSIDMS TRANSACTION_SHARING Parameter

A batch application tells Advantage CA-IDMS that it wants to use transaction sharing for all of its database sessions by specifying a new SYSIDMS parameter:

```
TRANSACTION_SHARING=ON
```

If this is specified, Advantage CA-IDMS enables transaction sharing for every database session started by the application unless a call to IDMSIN01 changes the transaction sharing option.

If TRANSACTION_SHARING=ON is established as a default in a SYSIDMS load module, it can be overridden at runtime by specifying:

```
TRANSACTION_SHARING=OFF
```

Notes:

1. Sharing transactions in local mode enables concurrent sharing sessions to ready the same area in update mode.
2. Whether transaction sharing is in effect for a batch/CV database session is determined from the front-end (that is, from the batch address space), regardless of whether transaction sharing is enabled for the back-end (RHDCNP3S) task.

4.4.5.6 IDMSIN01 Call

An application program calls IDMSIN01 to override the current transaction sharing setting for the task or job step. New IDMSIN01 functions enable or disable transaction sharing as illustrated below. If a call is made from within an SQL routine, the transaction sharing setting that was current on entry to the routine is reestablished on exit. This means that the IDMSIN01 call affects only the current routine and any subordinate routines that it might invoke as a result of SQL commands that it issues.

```
ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.


*****************************************************************
*     The following is the 1st parameter on all IDMSIN01 calls
*****************************************************************
 01  RPB.
     02  FILLER             PIC X(36).

*****************************************************************
*     The following is the 2nd parameter on all IDMSIN01 calls
*****************************************************************
 01  REQ-WK.
     02  REQUEST-CODE      PIC S9(8) COMP.
         88  IN01-FN-TRACE            VALUE 00.
         88  IN01-FN-NOTRACE          VALUE 01.
         88  IN01-FN-GETPROF          VALUE 02.
         88  IN01-FN-SETPROF          VALUE 03.
```

```
           88  IN01-FN-GETMSG           VALUE 04.
           88  IN01-FN-GETDATE          VALUE 05.
           88  IN01-FN-GETUSER          VALUE 08.
           88  IN01-FN-SYSCTL           VALUE 10.
           88  IN01-FN-TRINFO           VALUE 16.
           88  IN01-FN-TXNSON           VALUE 28.
           88  IN01-FN-TXNSOFF          VALUE 29.
           88  IN01-FN-RRSCTX           VALUE 30.
           88  IN01-FN-STRCONV          VALUE 34.
       02  REQUEST-RETURN     PIC S9(8) COMP.

*****************************************************************
*     The following work fields are used by a variety of
*     IDMSIN01 calls
*****************************************************************
 01  WORK-FIELDS.
     02  WK-DTS-FORMAT          PIC S9(8) COMP VALUE 0.
     02  LINE-CNT               PIC S9(4) COMP.
     02  WK-DTS                 PIC X(8).
     02  WK-CDTS                PIC X(26).
     02  WK-KEYWD               PIC X(8).
     02  WK-VALUE               PIC X(32).
     02  WK-DBNAME              PIC X(8).
     02  WK-USERID              PIC X(32).
     02  WK-SYSCTL              PIC X(8).
     02  WK-TIME-INTERNAL       PIC X(8).
     02  WK-TIME-EXTERNAL       PIC X(8).
     02  WK-DATE-INTERNAL       PIC X(8).
     02  WK-DATE-EXTERNAL       PIC X(10).
     02  WK-RRS-FAKE-FUNCTION   PIC S9(4) COMP.
         88  IN01-FN-RRSCTX-GET       VALUE 01.
         88  IN01-FN-RRSCTX-SET       VALUE 02.
     02  WK-RRS-FUNCTION-REDEF    REDEFINES WK-RRS-FAKE-FUNCTION.
         03  WK-RRS-FAKE-FILLER PIC X.
         03  WK-RRS-FUNCTION    PIC X.
     02  WK-RRS-CONTEXT         PIC X(16).
     02  WK-STRING-FUNCTION     PIC X(4).
         88  CONVERT-EBCDIC-TO-ASCII  VALUE 'ETOA'.
         88  CONVERT-ASCII-TO-EBCDIC  VALUE 'ATOE'.
     02  WK-STRING              PIC X(17)
                                      VALUE 'String to convert'.
     02  WK-STRING-LENGTH       PIC S9(8) COMP VALUE 17.

*****************************************************************
*     The following group item is only used by the call that
*     retrieves SQL error messages
*****************************************************************
 01  SQLMSGB.
     02  SQLMMAX            PIC S9(8) COMP VALUE +6.
     02  SQLMSIZE           PIC S9(8) COMP VALUE +80.
     02  SQLMCNT            PIC S9(8) COMP.
     02  SQLMLINE           OCCURS 6 TIMES PIC X(80).

*****************************************************************
*     The following SQL include statement is needed only for
*     the call that retrieves SQL error messages, and is only
*     required if the program contains no other SQL statements.
*****************************************************************
 EXEC SQL
```

```
     INCLUDE SQLCA
 END-EXEC.


*************************************************************
 PROCEDURE DIVISION.
*************************************************************


*************************************************************
*  Call IDMSIN01 to activate Transaction Sharing for this
*  task.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*************************************************************

     SET IN01-FN-TXNSON  TO TRUE.
     CALL 'IDMSIN01' USING RPB REQ-WK.


*************************************************************
*  Call IDMSIN01 to deactivate Transaction Sharing for this
*  task.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*************************************************************
```

4.5 Enhanced Compatibility with Open Standards

Advantage CA-IDMS Release 16.0 provides enhanced compatibility with Open Standards with the addition of several SQL scalar functions. The scalar functions are automatically installed with Advantage CA-IDMS and many are implemented as user-defined functions. The new scalar functions complement existing scalar functions that were distributed with earlier releases of Advantage CA-IDMS.

The following tables list the scalar functions included with Advantage CA-IDMS/DB and that are defined in the JDBC specification or are commonly used in the industry. For a complete description of the functions, refer to Appendix B, "New and Revised SQL Statements."

The notations are coded as follows:

- B — Function is implemented as a true built-in function.
- U — Function is implemented as a user-defined function in the SYSCA schema.
- UN — Function is implemented as a user-defined function in the SYSCA schema, but it is not included in the JDBC specification.
- X — Function was implemented in an earlier release of Advantage CA-IDMS. There might be semantic differences between the Advantage CA-IDMS implementation and the Open Standards definitions.

For complete descriptions of the functions marked *U* and *B*, see Appendix B, "New and Revised SQL Statements."

4.5.1 Numeric Functions

Notation	Function	Return Value
U	ABS(*number*)	Absolute value of *number*
U	ACOS(*float*)	Arccosine, in radians, of *float*
U	ASIN(*float*)	Arcsine, in radians, of *float*
U	ATAN(*float*)	Arctangent, in radians, of *float*
U	ATAN2(*float1*, *float2*)	Arctangent, in radians, of *float2*/*float1*
UN	CEIL(*number*)	Smallest integer greater than or equal to *number*
U	CEILING(*number*)	Smallest integer greater than or equal to *number*
U	COS(*float*)	Cosine of *float* radians

Notation	Function	Return Value
UN	COSH(*float*)	Hyperbolic cosine of *float* radians
U	COT(*float*)	Cotangent of *float* radians
U	DEGREES(*number*)	Degrees in *number* radians
U	EXP(*float*)	Exponential function of *float*
U	FLOOR(*number*)	Largest integer less than or equal to *number*
U	LOG(*float*)	Base e logarithm of *float*
U	LOG10(*float*)	Base 10 logarithm of *float*
U	MOD(*integer1*, *integer2*)	Remainder for *integer1*/*integer2*
U	PI()	The constant pi
U	POWER(*number*, *power*)	*Number* raised to (integer) *power*
U	RADIANS(*number*)	Radians in *number* degrees
U	RAND(*integer*)	Random floating point for seed *integer*
U	ROUND(*number*, *places*)	*Number* rounded to *places*
U	SIGN(*number*)	-1 to indicate number is less than 0 0 to indicate number is equal to 0 1 to indicate number is greater than 0
UN	SINH(*float*)	Hyperbolic sine of *float* radians
U	SIN(*float*)	Sine of *float* radians
U	SQRT(*float*)	Square root of *float*
U	TAN(*float*)	Tangent of *float* radians
UN	TANH(*float*)	Hyperbolic tangent of *float* radians
U	TRUNCATE(*number*, *places*)	*Number* truncated to *places*

4.5.2 String Functions

Notation	Function	Return Value
B	CONCAT(*string1*, *string2*)	Character string formed by appending *string2* to *string1*; if a string is null, the result is DBMS-dependent

Notation	Function	Return Value
U	INSERT(*string1*, *start*, *length*, *string2*)	A character string formed by deleting *length* characters from *string1* beginning at *start*, and inserting *string2* into *string1* at *start*
B	LCASE(*string*)	A character string equal to *string* in which all uppercase characters are converted to lowercase
X	LEFT(*string*, *count*)	A character string equal to the *count* leftmost characters from *string*
X	LENGTH(*string*)	Integer representing the number of characters in *string*, excluding trailing blanks
X	LOCATE(*string1*, *string2*, *start*)	Position in *string2* of the first occurrence of *string1*, searching from the beginning of *string2*; if *start* is specified, the search begins from position *start*. 0 is returned if *string2* does not contain *string1* Position 1 is the first character in *string2*.
X	LTRIM(*string*)	A character string equal to *string* with leading blank spaces removed
U	REPEAT(*string*, *count*)	A character string formed by repeating *string count* times
U	REPLACE(*string1*, *string2*, *string3*)	A character sting equal to *string1* in which all occurrences of *string2* are replaced with *string3*
U	RIGHT(*string*, *count*)	The *count* rightmost characters in *string*
X	RTRIM(*string*)	The characters of *string* with no trailing blanks
U	SPACE(*count*)	A character string consisting of *count* spaces
X	SUBSTRING(*string*, *start*, *length*)	A character string formed by extracting *length* characters from *string* beginning at *start*

Notation	Function	Return Value
X	UCASE(*string*)	A character string equal to *string* in which all lowercase characters are converted to uppercase

4.5.3 Time and Date Functions

Notation	Function	Return Value
B	CURDATE()	The current date as a date value
B	CURTIME()	The current local time as a time value
U	DAYNAME(*date*)	A character string representing the day component of *date*; the name for the day is specific to the data source
B	DAYOFMONTH(*date*)	An integer from 1 to 41 representing the day of the month in *date*
U	DAYOFWEEK(*date*)	An integer from 1 to 7 representing the day of the week in *date*; 1 represents Sunday
U	DAYOFYEAR(*date*)	An integer from 1 to 366 representing the day of the year in *date*
X	HOUR(*time*)	An integer from 0 to 23 representing the hour component of *time*
X	MINUTE(*time*)	An integer from 0 to 59 representing the minute component of *time*
X	MONTH(*date*)	An integer from 1 to 12 representing the month component of *date*
U	MONTHNAME(*date*)	A character string representing the month component of *date*; the name for the month is specific to the data source
B	NOW()	A timestamp value representing the current date and time

Notation	Function	Return Value
U	QUARTER(*date*)	An integer from 1 to 4 representing the *quarter* in date; 1 represents January 1 through March 31
X	SECOND(*time*)	An integer from 0 to 59 representing the second component of *time*
U	WEEK(*date*)	An integer from 1 to 53 representing the week of the year in *date*
X	YEAR(*date*)	An integer representing the year component of *date*

4.5.4 System Functions

Notation	Function	Return Value
B	DATABASE	Current database
B	IFNULL(*expression*, *value*)	*Value* if *expression* is null; *expression* if not null
B	USER()	Current user

4.5.5 Conversion Functions

Notation	Function	Return Value
B	CONVERT(*value*, *SQLtype*)	*Value* converted to *SQLtype*, where *SQLtype* can be any valid SQL data type.

4.6 XML Publishing

This feature implements XML Publishing. It allows applications to generate XML data from data stored in an Advantage CA-IDMS database easily and with high performance. Although the feature's API is based on SQL, Advantage CA-IDMS SQL supports SQL DML on non-SQL defined databases. Thus, also allowing non-SQL defined Advantage CA-IDMS databases to be used as the data sources for XML Publishing.

The feature is based on and implements a subset of the SQL/XML ISO standard as described in the ISO publication WD ISO/IEC 9075-14:2007 (E), titled "Information technology - Database languages - SQL - Part 14: XML-Related Specifications (SQL/XML)". A few extensions have been made available. These are indicated in the following section.

The XML Publishing capability is made available through a set of SQL functions, a new internal XML data type, an SQL table procedure, and an XML encoding session option.

4.6.1 SQL/XML Functions

The SQL/XML functions are not true SQL functions but pseudo functions. Some of the SQL/XML functions:

- have a variable number of arguments
- use "AS" to specify an alias for an expression as an argument
- have arguments that can be of any type
- support subqueries as arguments
- have arguments that can be SQL identifiers

Following is a list of the SQL/XML routines (all functions except for one table procedure) that can be used for XML Publishing purposes:

XML Value Functions

- **XMLAGG** — aggregates a set of XML values.
- **XMLATTRIBUTES** — function-like construct, only allowed as argument of XMLELEMENT. Generates XML attributes.
- **XMLCOMMENT** — generates an XML comment.
- **XMLCONCAT** — concatenates multiple XML values.
- **XMLELEMENT** — generates an XML element.
- **XMLFOREST** — generates a forest (collection) of XML elements.
- **XMLNAMESPACES** — function-like construct, only allowed as argument of XMLELEMENT or XMLFOREST. Generates XML namespaces.

- **XMLPARSE** — checks if an XML value is well-formed.
- **XMLPI** — generates an XML processing instruction, that is, an XML declaration or style sheet specification.
- **XMLROOT** — sets the XML version and/or standalone option in the XML declaration of a root XML element. If an XML declaration is not yet present, one is created with the ENCODING pseudo attribute set to the sessions current XML encoding.

Advantage CA-IDMS Scalar Functions

- **XMLPOINTER** — returns a pointer to a character large object or CLOB, representing a serialized XML value. This Advantage CA-IDMS extension can be used in programs, running in the same address space as Advantage CA-IDMS.
- **XMLSERIALIZE** — returns a character or binary value with a maximum length of 30,000, representing a serialized XML value.

Table Procedure

- **XMLSLICE** — retrieves character or binary slices of equal length from the serialization of an XML value.

For More Information: For detailed syntax and semantics of the above SQL/XML routines, see B.4, “SQL/XML Functions and Table Procedure” on page B-61.

4.6.2 XML Data Type and XML Values

In this feature, the XML data type is a new internal only data type to represent XML data. XML values usually are used as arguments to some of the SQL/XML functions.

The only way to produce an XML value is through the invocation of an XML value function, possibly indirectly through using a subquery that returns an XML value. The return value of all XML value functions is of the XML data type. A **subquery** used as an **XML-value-expression** must be of the XML data type, which implies that its SELECT list contains an XML value function.

Data of the XML type cannot be stored in a database or directly used in application programs using the standard SQL API. Programs running in the Advantage CA-IDMS CV address space or in batch local mode can access serialized XML data using the **XMLPOINTER** function. After serialization and casting to CHAR or VARCHAR through the **XMLSERIALIZE** function, XML data can be accessed using any supported SQL API on any platform.

To bypass the 30,000 length limit of the character string returned by XMLSERIALIZE, use the **XMLSLICE** table procedure.

Examples of valid XML values are:

- An XML element
- A forest of XML elements
- Textual content of an XML element
- The NULL value
- An XML subquery (Select XMLAGG(XMLELEMENT(...)) from ATABLE where ...)

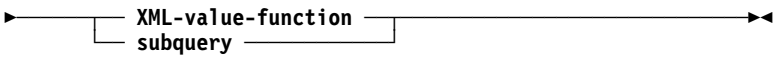
4.6.2.1 Syntax

No syntax is available. This is a special, internal only data type.

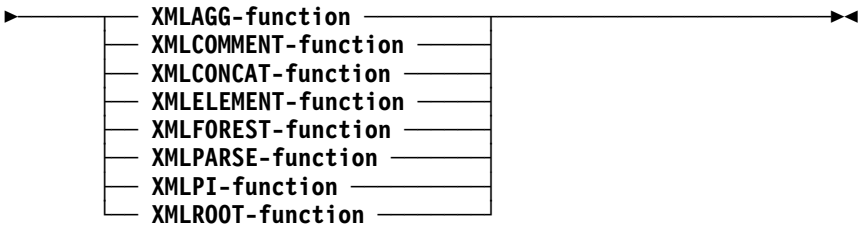
4.6.3 XML-value-expression

Specifies an XML value. For more information on XML values, see 4.6.2, “XML Data Type and XML Values” on page 4-34.

4.6.3.1 Syntax



Expansion of XML-value-function



4.6.3.2 Parameters

XML-value-function
Specifies an **XML-value-function** that returns an XML value.

For a complete description of the syntax and parameters for the **XML-value-function**, see B.4, “SQL/XML Functions and Table Procedure” on page B-61.

subquery
Specifies a **subquery** that must return a single XML value or the NULL value.

Note: If **subquery** is used as an XML value, it must return 0 or 1 row.

4.6.4 Mappings

SQL and XML are two different languages with their own specific language elements and grammar. When using SQL to produce XML, SQL language elements must be mapped to XML using appropriate rules.

This section describes the rules that are used for mapping of:

- Plain text SQL to XML
- SQL identifier to XML
- SQL data type values to XML schema data type values

4.6.4.1 Mapping Plain Text SQL to XML

This mapping is between the character set(s) of the SQL language and Unicode.

This feature supports mapping from the SQL character set (EBCDIC) to Unicode using encodings UTF-8, UTF-16-BE (big endian), and UTF-16-LE (little endian). This mapping is implemented using the standard Advantage CA-IDMS code tables (RHDCCODE) and is controlled through the XML encoding session option. The default is not to map; the serialized XML values are encoded in EBCDIC.

For information on encoding XML values, see the XML encoding session option under B.3.17, “SET SESSION Statement” on page B-58.

4.6.4.2 Mapping SQL Identifier to XML

You can use a much greater range of characters in an SQL identifier than in an XML name. Note that any character can be used in an SQL identifier delimited by double quotes.

The normative definition of valid XML Name characters is found in the SQL/XML ISO standard. Valid first characters of XML Names are:

```
Letters, <underscore>, and <colon>
```

Valid XML Name characters, after the first character, are:

```
Letters, Digits, <period>, <minus sign>, <underscore>, <colon>,
CombiningChars, and Extenders
```

Note that the XML definition of Letter and Digit is broader than <simple Latin letter> and <digit> respectively.

There are two types of XML names: XML NCName and XML QName. An XML NCName is an XML non-colonized name, that is, an XML name that does not contain any colon (:) character. An XML QName is an XML-qualified name that consists of the XML namespace prefix and the local part of the name, separated by a colon (:) character. The namespace prefix and the local part of the name must be XML

NCNames. For example, xsd:string is an XML Qname, where xsd is the namespace prefix which must have been declared for a namespace URI, and string is the local part of the name.

There are two types of mapping of SQL identifiers to XML: fully escaped and partially escaped mapping. Fully escaped mapping is used for all SQL identifiers that are derived from an SQL column name, that is, in the XMLATTRIBUTES and XMLFOREST functions. Partially escaped mapping is used in all the other cases, that is, in the AS clause of the XMLATTRIBUTES, XMLFOREST, and XMLNAMESPACES functions, and in the NAME clause of the XMLELEMENT and XMLPI functions.

XML names that begin with the characters "xml" (in any combination) are reserved by W3C for use in future recommendations, and therefore, cannot be used.

The following table shows some mapping examples:

SQL Identifier	Fully Escaped XML Name	Partially Escaped XML Name
department	DEPARTMENT	DEPARTMENT
"department"	department	department
"last name"	last_x0020_name	last_x0020_name
"last_xname"	last_x005F_name	last_x005F_name
"dept:id"	dept_x003A_id	dept:id
":id"	_x003A_id	_x003A_id
"xmlcolumn"	_x0078_mlcolumn	xmlcolumn
"Xmlcolumn"	_x0058_mlcolumn	Xmlcolumn
xmlcolumn	_x0058_MLCOLUMN	XMLCOLUMN

4.6.4.3 Mapping SQL Data Type Values to XML Schema Data Type Values

This feature supports mapping SQL data type values to XML schema data type values; however, mapping of GRAPHIC and VARGRAPHIC are not supported.

You can map null values using absence or xsi:nil="true".

The complete mapping rules are described in the SQL/XML ISO standard specification. As an oversimplification, mapping can be described as the result of the casting of the SQL data value to VARCHAR(max).

The following table shows some of the character value mappings:

SQL Character Value	Mapped Value
<	&lt;
>	&gt;
&	&amp;
Carriage Return	&#x0d;
'	&apos;
"	&quot;

This mapping does not apply to the characters belonging to an XML CDATA section; a CDATA section begins with the string "<![CDATA[" and ends with the string "]]>".

4.6.5 Example

In the following example, the use of many of the SQL/XML functions is shown. The result of the SELECT is an XML document that contains all the employees from the DEMOEMPL.EMPLOYEE table, grouped by department. The DEMOEMPL.EMPLOYEE and DEMOEMPL.DEPARTMENT tables are equi-joined on the DEPT_ID. Note also how the SELECT statement clearly and naturally reflects the structure of the XML document.

```
select
  xmlserialize
  ( CONTENT
    xmlroot
    ( xmlconcat
      ( xmlpi
        ( Name "xml-stylesheet"
        , 'type="text/xsl"
          href="http://usilca11:26720/IDD/SQLXML_XSL2/xsl"'
        )
      , xmlelement
        ( Name "EmployeesByDepartment"
        , xmlnamespaces('http:/ca.com/ns1' as "ns1")
        , xmlagg
          ( xmlelement
            ( Name "Department"
            , xmlattributes(DEPT_ID as "DeptId")
            , xmlelement
              ( Name "Employees"
              , select xmlagg
                ( xmlelement
                  ( Name "Employee"
                  , xmlattributes(EMP_ID as "EmpId")
                  , EMP_FNAME
                  , EMP_LNAME
```

```
                          , xmlelement
                            ( Name "Address"
                            , xmlforest
                              ( e.STREET     as  "Street"
                              , e.CITY       as  "City"
                              , e.STATE      as  "State"
                              )
                            )
                          )
                        )
                         from DEMOEMPL.EMPLOYEE e
                        where d.DEPT_ID = e.DEPT_ID
                      )
                    )
                  )
                )
              )
            , version '1.0', standalone yes
            )
            as CHAR(3000)
          )
from DEMOEMPL.DEPARTMENT d
```

The result, which has been formatted for clarity, is similar to this:

```
<?xml version="1.0" encoding="EBCDIC" standalone="yes"?>
<?XML-STYLESHEET TYPE="TEXT/XSL"
                   HREF="HTTP://USILCA11:26720/IDD/SQLXML_XSL2/XSL"?>
<EMPLOYEESBYDEPARTMENT xmlns:NS1="HTTP:/CA.COM/NS1">

<DEPARTMENT DEPTID="1120">PURCHASING - SERVICE
  <EMPLOYEES>
    <EMPLOYEE EMPID="2898">Mary                Umidy
      <ADDRESS><STREET>895A Braintree Circle               </STREET>
               <CITY>Medford             </CITY>
               <STATE>MA</STATE>
      </ADDRESS>
    </EMPLOYEE>

    <EMPLOYEE EMPID="3338">Mark                White
      <ADDRESS><STREET>560 Camden St                       </STREET>
               <CITY>Canton              </CITY>
               <STATE>MA</STATE>
      </ADDRESS>
    </EMPLOYEE>

    <EMPLOYEE EMPID="3294">Carolyn             Johnson
      <ADDRESS><STREET>79 High St                          </STREET>
               <CITY>Brookline           </CITY>
               <STATE>MA</STATE>
      </ADDRESS>
    </EMPLOYEE>
```

```
    <EMPLOYEE EMPID="2004  ">Eleanor           Johnson
      <ADDRESS><STREET>225 Fisk St                          </STREET>
               <CITY>Medford               </CITY>
               <STATE>MA</STATE></ADDRESS>
    </EMPLOYEE>
  </EMPLOYEES>
</DEPARTMENT>

<DEPARTMENT DEPTID="4200">LEASING - NEW CARS
  <EMPLOYEES>
  </EMPLOYEES>
</DEPARTMENT>
<DEPARTMENT DEPTID="2210">SALES - NEW CARS
  <EMPLOYEES>
    <EMPLOYEE EMPID="4027">Cecile              Courtney
      <ADDRESS><STREET>99 West Main St
                 </STREET>
               <CITY>Natick              </CITY>
               <STATE>MA</STATE>
      </ADDRESS>
    </EMPLOYEE>
    <EMPLOYEE EMPID="3991">Fred                Wilkins
      <ADDRESS><STREET>344 Stevens St                       </STREET>
               <CITY>Taunton             </CITY>
               <STATE>MA</STATE>
      </ADDRESS>
   </EMPLOYEE>
  ⋮
  </EMPLOYEES>
</DEPARTMENT></tt>
</EMPLOYEESBYDEPARTMENT>
```

4.6.6 SQLSTATE Values

This section contains the SQL/XML enhancements in the SQLSTATE variable.

▸▸ For more information on the SQLSTATE variable, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

The following table contains the SQL/XML SQLSTATE values:

Category	Condition	Class	Subcondition	Subclass
X	Data exception	22	(no subclass)	000
			invalid comment	00S
			invalid processing instruction	00T
			invalid XML content	00N

Category	Condition	Class	Subcondition	Subclass
			invalid XML document	00M
			nonidentical notations with the same name	00J
			nonidentical unparsed entities with the same name	00K
			not an XML document	00L
			XML value overflow	00R
X	SQL/XML mapping error	0N	(no subclass)	000
			unmappable XML Name	001
			invalid XML character	002
W	Warning	01	(no subclass)	000
			column cannot be mapped	010

Chapter 5. Administrative and Operational Enhancements

5.1 Overview

This chapter describes the following utility, sysgen, and security enhancements in Release 16.0:

- Online execution of utilities
- LOCK AREA command
- ALREADY LOCKED option
- Database name for utility use
- Two-phase commit enhancements
- Cloning LTERM and PTERM sysgen definitions
- Security in IDMSBCF and OCF for the utility commands
- IDMSBCF input/output reassignment
- Online compiler enhancements
- PRINT SPACE utility enhancement
- Multiple segment processing using the EXTRACT JOURNAL, ROLLBACK, and ROLLFORWARD utilities
- System startup enhancements
- #WTL macro enhancements

5.2 Online Execution of Utilities

Release 16.0 enables many utilities to be executed online that previously could only be executed through the batch command facility (IDMSBCF). By extending the environment in which these utilities can be executed, the DBA is able to perform more work from a single user interface, thus increasing their productivity.

The following utilities can now be run under a central version, using OCF or IDMSBCF:

- CLEANUP SEGMENT
- FIX PAGE
- FORMAT AREA
- FORMAT SEGMENT
- INSTALL STAMPS
- LOCK AREA
- PRINT INDEX
- PRINT PAGE
- PRINT SPACE FOR AREA
- PRINT SPACE FOR SEGMENT
- SYNCHRONIZE STAMPS
- TUNE INDEX
- UPDATE STATISTICS

5.2.1 Usage Considerations

Area usage mode: In order to execute a utility online, the affected areas must be available to the central version in the appropriate mode. For utilities that perform updates, the affected areas must be in update mode to the central version. For utilities that perform only retrievals, the affected areas must be in either retrieval or update mode. If the above requirement is not met, you receive a DB002352 error message indicating that the required lock mode is not available.

Committing prior work: Before executing certain utilities online, you must commit any previous work that has been done within the current SQL session. This requirement applies to the following utilities:

- FIX PAGE
- FORMAT AREA
- FORMAT SEGMENT
- LOCK AREA

- LOCK SEGMENT

The following sequence of statements illustrates how to commit prior work before issuing a FORMAT AREA statement:

```
SELECT * FROM SYSTEM.TABLE;
COMMIT;
FORMAT AREA VSAMT.KSDS2;
```

If you omit the COMMIT, you receive a DB002043 error message:

```
Command not allowed with an open transaction
```

Log messages: If you run an online FORMAT statement or FIX PAGE statement, an informational message is written to the log identifying the area name being updated and the time of the update.

Batch-only utilities: If you attempt to execute a utility online that is supported only in batch local mode, such as UNLOCK or FORMAT FILE, you receive a DB002990 error message indicating that the statement is not supported in central version.

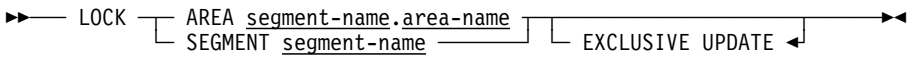
5.3 LOCK AREA Statement

Release 16.0 provides a new LOCK AREA/SEGMENT utility statement that allows a DBA to explicitly lock an area. This enables an administrator to place a lock on an area that remains in effect across several commands. In this way, access to an area by other users can be prevented while a series of operations are performed on it.

5.3.1 Authority

In order to lock an area, you need DBAWRITE authority on the area.

5.3.2 Syntax



5.3.3 Parameters

AREA

Directs the LOCK utility statement to lock a specified area.

segment-name

Specifies the name of the segment associated with the area to be locked.

area-name

Specifies the name of the area to be locked.

SEGMENT

segment-name

Specifies the name of the segment to be locked.

EXCLUSIVE UPDATE

Specifies the update mode. EXCLUSIVE UPDATE is the default mode and the only mode currently supported.

5.3.4 Usage

Local mode execution: If the LOCK AREA statement is issued through IDMSBCF executing in local mode, a physical lock is placed on the area. The lock remains in effect until an explicit UNLOCK AREA is issued. If the area is already locked, the LOCK AREA statement fails with a DB002035 error message as illustrated below:

```
LOCK AREA USERDB.EMP_AREA;
Status = -4       SQLSTATE = 5000B        Messages follow:
DB002352 C-4M353: Area USERDB.EMP_AREA required area lock mode not available
```

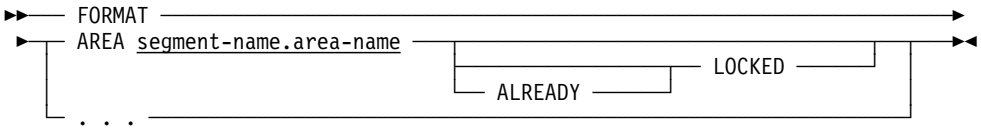
Online execution: When the LOCK AREA utility is run under a central version using OCF or IDMSBCF, a logical lock is placed on the area. This lock prevents all access to the area by other users until a commit or rollback is issued. If executing online, a commit is automatically issued at end of task prior to the pseudo-converse unless autocommit is disabled through a SET OPTIONS statement.

5.4 ALREADY LOCKED Option

Release 16.0 provides an optional ALREADY LOCKED clause for the FORMAT AREA utility and the FIX PAGE utility. This parameter allows you to continue processing the FORMAT AREA or FIX PAGE commands even if the target area is currently locked.

5.4.1 FORMAT AREA Utility Statement

5.4.1.1 Syntax



5.4.1.2 Parameters

ALREADY LOCKED

Specifies that if the target area of a FORMAT command is locked, the FORMAT command continues processing. If you omit the ALREADY LOCKED option and the target area of a FORMAT is locked, you receive a DB002352 error message and the command fails.

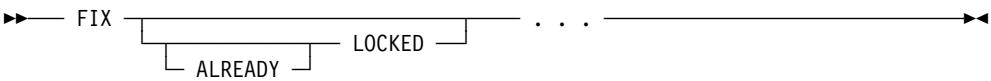
5.4.1.3 Usage

Formatting a locked area: If you are executing the format utility in local mode against a target area that is physically locked, you must specify ALREADY LOCKED. Otherwise, you'll receive a DB002352 error message. If ALREADY LOCKED is specified, the area remains locked after the format is complete. The ALREADY LOCKED option is not required if formatting an area under central version using OCF or IDMSBCF and the option is ignored, if specified. If formatting a segment or a file, the ALREADY LOCKED option cannot be specified and no area lock validation is performed.

Formatting an unlocked area: If executing the format utility against an area that is not locked, ALREADY LOCKED is ignored if specified.

5.4.2 FIX PAGE Utility Statement

5.4.2.1 Syntax



5.4.2.2 Parameters

ALREADY LOCKED

Specifies that if the target area or areas of a FIX PAGE are locked, the FIX PAGE command continues processing. If you omit the ALREADY LOCKED option and the target area of a FIX PAGE command is locked, you receive a DB002352 error message, and the command fails.

5.4.2.3 Usage

Repairing a locked area: If executing the fix page utility in local mode against a target area that is physically locked, you must specify ALREADY LOCKED otherwise you receive a DB002352 error message. The ALREADY LOCKED option is not required if repairing an area under central version using OCF or IDMSBCF and is ignored if specified.

Unlocking a locked area: The fix page utility cannot be used to update an area's physical area lock. Instead use the LOCK and UNLOCK area utility statements to do this.

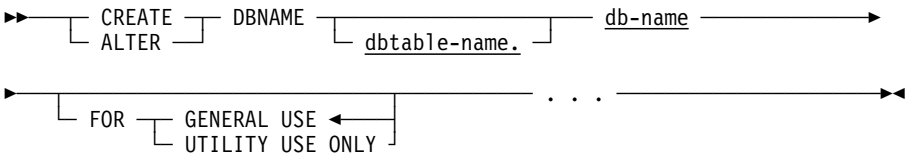
5.5 Database Name for Utility Use

Release 16.0 allows you to designate a database name for utility-use-only. Doing so has the following two effects:

- The DBNAME cannot be used to access data through SQL or navigational DML. Any attempt to do so fails.
- The DBNAME is not validated during startup or by the LOOK utility for such things as duplicate area names. By avoiding this validation, no warning messages are issued. This feature enables the DBA to create database names for administrative convenience while avoiding warning messages indicating an incorrectly defined database name. For example, the DBA may wish to create a utility-use-only database name that includes all segments in the DMCL for use with the QUIESCE DBNAME system task.

5.5.1 CREATE DBNAME Statement

5.5.1.1 Syntax



5.5.1.2 Parameters

FOR GENERAL USE
Specifies this DBNAME is for general use, such as accessing data through navigational or SQL DML requests. FOR GENERAL USE is the default.

UTILITY USE ONLY
Specifies that this DBNAME is for administrative purposes only; for example, as in the QUIESCE system task. The DBNAME cannot be used to access data through SQL or navigational DML.

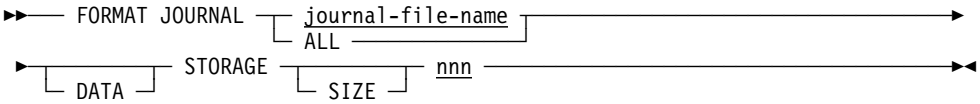
5.5.1.3 Usage

Utility-use-only DBNAMEs: The ability to designate a database name for utility-use-only, allows the DBA to define arbitrary collections of areas for administrative convenience while avoiding warnings for such things as duplicate area names. Since a utility-use-only DBNAME cannot be used to access data through navigational or SQL DML, there is no need to restrict the areas that it includes.

5.6 FORMAT JOURNAL Utility Statement

To provide sufficient space in the journal files for recording information about other systems with which a system communicates, the FORMAT JOURNAL command is enhanced in Release 16.0. In most cases, the default size is sufficient and no explicit size parameter is needed; however, if a system's journal block size is very small or it communicates with many other Advantage CA-IDMS or CICS systems, it may be necessary to reserve additional space.

5.6.1 Syntax



5.6.2 Parameters

DATA STORAGE SIZE nnn

Specifies the amount of space to reserve in 1K (1024 byte) increments for Data Storage in a journal file, where *nnn* is an integer from 1 32,767.

5.6.3 Usage

Specifying a journal storage size: All journals must have the same amount of space since the data in one journal is replicated to every other journal.

The actual size allocated may be higher than specified due to rounding. Space is allocated in blocks whose size is (journal block size - 256). By default, one block is allocated. Additional blocks are allocated if needed until the total size meets or exceeds the size specified.

5.7 Two-Phase Commit Enhancements

Release 16.0 enhances the recovery utilities to report on distributed transactions and support the use of a manual recovery control file for addressing incomplete distributed transactions. This section describes these common enhancements as they apply to the following recovery utility statements:

- EXTRACT JOURNAL
- FIX ARCHIVE
- MERGE ARCHIVE
- PRINT JOURNAL
- ROLLBACK
- ROLLFORWARD

5.7.1 Reporting on Distributed Transactions

A distributed transaction journal checkpoint record consists of a fixed portion and up to three variable arrays of data. The fixed portion contains the distributed transaction identifier (DTRID) and a local branch ID (BID), which identifies an individual branch of the distributed transaction. The fixed portion can be followed by any combination of the following:

- A list of local transaction identifiers (LIDs), one for each transaction branch that made local database changes
- A list of external transaction identifiers if the transaction is known externally by another identifier, such as an XA XID or RRS URID.
- A list of interests that other resource or transaction managers have in the distributed transaction

The recovery utilities report some or all of the above information in their detailed report and list distributed transactions that were incomplete at stop time in their summary report.

The example below shows the output that is produced by PRINT JOURNAL REPORT FULL when it encounters a typical DCOM record. If the REPORT TERSE option is specified, neither external transaction identifiers nor resource manager interests are included. Other recovery utilities show similar information.

```
NODE SYSTEM74 DTRID-BID   SYSTEM74::01650C9509CE38A3-01650C90A4CDA0BD         DCOM
  LOC_ID    10016  PGM_ID PROCDISM
  RRS_URID  B8DEBCA57E84B6700000000D01020000                                 *...v=d.....                *
  RM NAME   SYSTEM74::RRS_RMI   TYPE RRS     ROLE SDSRM  STATE InDoubt      FLG1/2 0001 EXITS 40 00340000000000000
            D9D9E240C2404040404040404040404040B8DEBCA57E84B6700000000D01020000  *RRS B           ...v=d.....*
            18C1E3D94BC2F8F9F0F9F7F8F6C1F2C1F8C1C1F4F04BC9C2D4                 *.ATR.B8909786A2A8AA40.IBM       *
  RM NAME   SYSTEM73::DSI_CLI   TYPE IDMS    ROLE CRM    STATE InDoubt      FLG1/2 0000 EXITS 76 00000000000000000
            E2E8E2E3C5D4F7F301650C90A4CDA904000000000800000001650C2E949172E101  *SYSTEM73....u.z.....mj...*
            650C9509CE38A380000000000000000000E2E8E2E3C5D4F7F303D9C5C1D340C4C3  *..n...t.........SYSTEM73.REAL DC*
            400000000000000000000D5C4E2E8E2E3C5D4F7F3000000000000000000000000  * .........NDSYSTEM73.....*
            00000000                                                           *....                       *
```

A brief description of the report's contents follows. For an in-depth discussion of the meaning of this report, see Chapter 3, “Two-Phase Commit Support.”

- Node SYSTEM74 — Identifies the name of the system that produced the journal entry.
  - DTRID-BID SYSTEM74::01650C9509CE38A3-01650C90A4CDA0BD — Identifies the DTRID and the BID of the top-level branch of the distributed transaction for which the DCOM record was written. The DTRID is SYSTEM74::01650C9509CE38A3 and the BID is 01650C90A4CDA0BD.
  - DCOM — The type of distributed transaction journal record that is being reported.
- LOC_ID 10016 — Identifies the work done by a local transaction branch that is included in the distributed transaction. In this case, the local transaction identifier is 10016.
- PGM_ID PROCDISM — Identifies the name of the application program that started the local transaction branch. In this case, the program is PROCDISM.
- RRS URID B8DEBCA57E84B6700000000D01020000 — Identifies the transaction, as it is known externally.
- RM NAME SYSTEM74::RRS_RMI — Identifies a resource manager that has registered an interest in the distributed transaction. In this case, the resource manager is RRS.
  - TYPE RRS — Indicates that the RM type is RRS.
  - ROLE SDSRM — Indicates that this interest is the controlling interest for the transaction, and therefore RRS is the transaction's coordinator.
  - STATE InDoubt — Indicates the interest's state. In this case, the interest is in an InDoubt state.
  - FLG1/2 0001 — Displays flags that are used to restart the transaction following a system failure.
  - EXITS 40 0034000000000000 — Shows the exits that have been registered by the resource manager and the responses returned by the exits that have already been called during the life of the transaction.
  - D9D9E240C2... — Shows the data (in hex and character format) that the resource manager wishes preserved should it be necessary to restart the transaction following a system failure. This information varies depending on the resource manager that registered the interest.
- RM NAME SYSTEM73::DSI_CLI — Identifies a resource manager that has registered an interest in the distributed transaction. In this case, the resource manager is an Advantage CA-IDMS system named SYSTEM73.
  - TYPE IDMS — Indicates the type of the resource manager.

- ROLE CRM — Indicates that this interest is not a controlling interest for the transaction. Therefore, the associated resource manager (SYSTEM73) is a participant in the transaction.
- STATE InDoubt — Indicates the interest's state. In this case, the interest is in an InDoubt state.
- FLG1/2 0000 — Displays flags that are used to restart the transaction following a system failure.
- EXITS 76 0000000000000000 — Shows the exits that are registered by the resource manager and the responses returned by the exits that have been called during the life of the transaction.
- E2E8E2E3C5... — Shows the data (in hex and character format) that the resource manager wishes to have preserved if it is necessary to restart the transaction following a system failure. This information varies depending on the resource manager that registered the interest.

5.7.2 Manual Recovery Input Control File

A manual recovery input control file can be used to specify if an InDoubt distributed transaction should be committed or backed out. While considered optional, if this file is included in a utility's execution JCL, it is used as input to the following recovery operations:

- EXTRACT JOURNAL (unless ALL is specified)
- FIX ARCHIVE
- MERGE ARCHIVE (if COMPLETE is specified)
- PRINT JOURNAL
- ROLLBACK
- ROLLFORWARD (unless ALL is specified)

The file contains 80-byte records whose format is:

```
<Dtrid> <Action>
```

Where:
<Dtrid>is a 26-character display-format DTRID and <Action> is a COMMIT or BACKOUT. If more than one record specifies the same DTRID value, all but the last one are ignored.

The following example specifies that the transaction identified by DTRID SYSTEM74::01650C9509CE38A3 should be backed out:

```
SYSTEM74::01650C9509CE38A3 BACKOUT
```

If manual control input entries are used in a recovery operation that creates an output journal file (FIX ARCHIVE, EXTRACT JOURNAL and MERGE ARCHIVE), then

additional distributed transaction journal records are written to the output file to complete the transaction in the specified way.

The following is a sample of the report generated by FIX ARCHIVE. It lists entries in the manual recovery input control file and shows the effect of those entries in its summary report. In this example, the distributed transaction identified by CICSCICS::B8AD18E5A9BF0F41 is committed by the generation of new DCOM and DFGT journal records.

```
Input Control Records:
    CICSCICS::B8AD18E5A9BE1300   BACKOUT
    CICSCICS::B8AD18E5A9BF0F41   COMMIT
   . . .

Incomplete Distributed Transactions At Stop Time:

      NODE     DTRID-BID                                      STATE     ACTION
 **** SYSTEM74 CICSCICS::B8AD18E5A9BF0F41-016507A67C2E6D53   InDoubt   Commit
 *GEN SYSTEM74 DTRID-BID  CICSCICS::B8AD18E5A9BF0F41-016507A67C2E6D53         DCOM
               LOC_ID         28 PGM_ID CICSDML1
 *GEN SYSTEM74 DTRID-BID  CICSCICS::B8AD18E5A9BF0F41-016507A67C2E6D53         DFGT
```

5.7.3 Manual Recovery Output Control File

Since a manual recovery control file is an 80-byte card image file, you can create it with a text editor. If a manual recovery output control file is specified in the execution JCL, the following recovery options can create a prototype control file:

- FIX ARCHIVE
- MERGE ARCHIVE
- PRINT JOURNAL
- ROLLFORWARD (unless FROM EXTRACT is specified)

When a control file is generated, an entry is created for every distributed transaction whose final state is InDoubt. Automatically generated entries always specify that the transaction should be backed out. The resulting file should be edited prior to using it as input to a recovery operation.

5.7.4 Execution JCL Changes

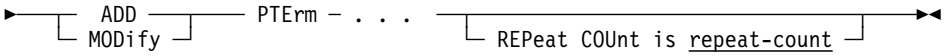
Manual recovery control files are optional, so no execution JCL changes are necessary unless their use is desired.

To use a manual recovery input control file, include a CTRLIN file definition or DD statement in the IDMSBCF execution JCL. To use a manual recovery output control file, include a CTRLOUT file definition or DD statement in the IDMSBCF execution JCL. The format for these files is fixed blocked with a record length of 80.

5.8 Cloning LTERM and PTERM Definitions

A new clause on the PTERM sysgen statement facilitates the definition of multiple physical and logical terminal definitions with identical characteristics. This eliminates the need for using individual LTERM and PTERM statements for each terminal.

5.8.1 Syntax



5.8.2 Parameters

repeat-count

Specifies the number of times the physical and eventually associated logical terminal should be cloned when a central version is started. *Repeat-count* must be an integer in the range 0 through 32767. *Repeat-count* 0 means no cloning. If a non-zero *repeat-count* is specified, the physical and logical terminal name should end on a sequence number and the sum of that sequence number and the repeat count should not cause a digit overflow.

5.8.3 Usage

Cloning PTERM/LTERM uses a naming convention: In order to clone PTERM/LTERM definitions, their names must end with a numeric value called the sequence number. This sequence number is incremented for each cloned PTERM and its associated LTERM, if the LTERM exists. Sysgen ensures that enough digits are available. It is the DBA's responsibility to ensure that a name conflict does not exist. A conflict occurs if a PTERM/LTERM is defined with the same name as a cloned PTERM/LTERM. If a name conflict is encountered, a warning message is generated and the explicitly defined PTERM/LTERM is used instead of the cloned definition.

Note: A single record in the dictionary represents cloned PTERM/LTERMs. Cloning starts after all dictionary PTERM/LTERM records are read and their associated control blocks built. If there is a name conflict, the PTERM/LTERM with conflicting name is built as defined by the dictionary record and the cloned PTERM/LTERM is discarded.

5.8.4 Example

```
ADD PTERM BULKP01
       REPEAT COUNT 98
       TYPE IS BULK.

ADD LTERM BULKL01
       PTERM BULKP01.
```

This definition results in the creation of 99 PTERM/LTERM pairs:

- BULKP01/BULKL01,

- BULKP02/BULKL02,
- BULKP03/BULKL03
- ...
- Until BULKP99/BULKL99

If a PTERM with name BULKP21 is also defined in the dictionary, this occurs:

- The PTERM BULKP21 and its associated LTERM (if any) is built according to the dictionary definition of BULKP21.
- Warning message DC391009 is output.
- The clone pair BULKP21/BULKL21 is not built, but cloning proceeds with BULKP22.

5.9 Security Enhancements

Advantage CA-IDMS 16.0 provides the ability to secure the individual utility commands that a user can execute in the Batch Command Facility (BCF) or the Online Command Facility (OCF). This is provided as an alternative to securing the individual resources that are accessed by a utility command.

A utility command may be secured by creating an activity resource for it. A user must then be granted execution privilege to run the utility.

The #UTABGEN macro is provided so you can associate activity resource numbers with utility commands.

5.9.1 Creating The Resource

The command: "CREATE RESOURCE ACTIVITY *application-name.activity-name* NUMBER *activity-number*" is used to create an activity type resource.

application-name

When creating an activity resource for a utility command, specify an application name of OCF for activities that are to be secured when running under OCF. To secure batch utility activities specify BCF for an application-name. If the same command is to be secured in both online and batch then two activity resources must be created.

activity-name

A user defined name assigned to this activity number. It must be 1 to 18 characters in length.

activity-number

A user defined number in the range of 1-256. It is unique within the application-name. It must match the number assigned in the #UTABGEN macro.

5.9.2 Assigning OCF/BCF Activity Numbers

OCF/BCF security provides the #UTABGEN macro for assigning activity numbers to OCF/BCF utility commands. In the #UTABGEN macro, you associate an activity number with an OCF/BCF command code.

5.9.2.1 #UTABGEN Example

In this example, #UTABGEN assigns the activity number of 14 to the OCF/BCF commands FORMAT and PRINTPAGE as represented by their command codes:

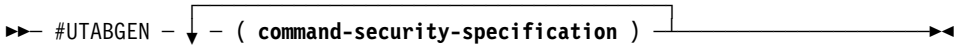
```
#UTABGEN (FORMAT,14,PRINTPAGE,14)
```

5.9.3 #UTABGEN

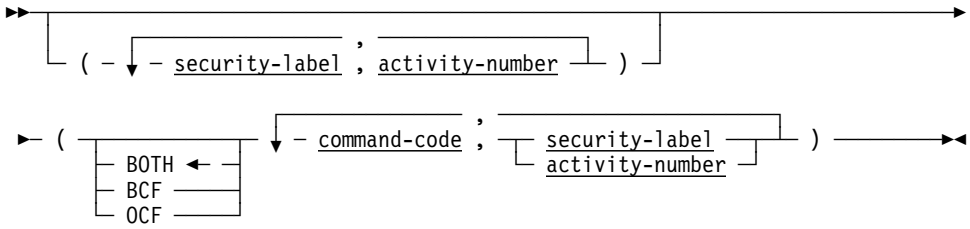
5.9.3.1 Purpose

Assigns activity numbers to utility commands.

5.9.3.2 Syntax



Expansion of command-security-specification



5.9.3.3 Parameters

security-label

Defines a security label and associates it with a BCF or OCF activity number. A security label can be used to classify utility commands by assigning a security label to one or more utility command codes. All commands with the same security label are associated with the security label's activity number.

A security label must be one alphabetic character (A through Z). You can define a maximum of 26 security labels in the #UTABGEN macro.

activity-number

Valid *activity-numbers* range from 0 to 255.

Note: An activity number of zero means no security.

BOTH

Specifies that the following set of command code security assignments applies to BCF and OCF commands. BOTH is the default.

BCF

Specifies that the following set of command code security assignments applies to BCF.

OCF

Specifies that the following set of command code security assignments applies to OCF commands.

command-code

Identifies a utility command to be secured. *Command-code* must match a code in the utility command code table, shown later in this section.

For example, FORMAT identifies the Format utility. PRINTSPACE identifies the Print Space utility.

security-label
Specifies a previously defined security label that you are associating with *command-code*.

activity-number
Specifies the BCF/OCF activity number you are associating with *command-code*. Valid *activity-numbers* are in the range of 0-255.

Note: An activity number of zero means no security.

5.9.3.4 Usage

Coding Considerations

- All lines except the first one must start in column 16.
- All lines except the last one must have a non-blank character in column 72.

General: When you use the #UTABGEN macro, you can assign an OCF/BCF activity number to one or more Utility commands.

- You can associate a specific OCF/BCF activity number (0 through 255) with a utility command.
- You can associate a security label (A through Z) with a utility command.

An activity number of zero turns off security for that security-label or command-code.

Coding zero is a useful way to turn off security, without deleting the command-code from the #UTABGEN source definition.

Only commands that are being secured must be coded. If omitted, they default to an activity code of zero.

Security labels must be defined before they can be assigned to command codes.

Use of security labels makes it easier to maintain security definitions when several commands are assigned the same OCF/BCF activity number. You define a security label in the #UTABGEN macro. You need only change the security label definition in the #UTABGEN macro to modify the security for all associated DCMT commands.

Generating the #UTABGEN Macro: The source file that specifies the #UTABGEN macro can only contain one macro. Once assembled, the resulting object must be link edited with IDMSDDAM.

5.9.3.5 Examples

Example 1

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
         #UTABGEN (A,3,B,10),                                          X
               (OCF,FORMAT,A,LOCK,A,UNLOCK,A),                         X
               (BCF,ARCHIVEJOURNAL,B,ARCHIVELOG,B),                    X
               (FIXPAGE,50)
         END
```

This example shows activity number 3 assigned to security-label A and activity number 10 assigned to security-level B.

OCF indicates that the commands that follow (within the parentheses) are assigned an activity number only when running in the online command facility OCF. Commands FORMAT, LOCK and UNLOCK are associated with security-label A. Since security-label A is currently assigned to the OCF/BCF activity number 3, the FORMAT, LOCK, and UNLOCK commands are assigned activity number 3.

BCF indicates that the commands within that group are only secured when invoked by the batch command facility: IDMSBCF. In this example, the ARCHIVE JOURNAL and ARCHIVE LOG commands are assigned to activity number 10, by the security-label B.

FIXPAGE is not qualified so activity number 50 is assigned to the FIX PAGE utility in OCF and BCF.

Example 2

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
         #UTABGEN (FORMAT,14,FIXPAGE,14)
         END
```

In this example, activity number 14 is assigned the utility command codes FORMAT and FIXPAGE. Because the codes are not identified as being OCF or BCF, the commands associated with these codes are secured in online and batch, and both use the same activity number.

5.9.3.6 For More Information

For more information about the #UTABGEN macro and the JCL associated with it, refer to the *Advantage CA-IDMS Security Administration*.

5.9.3.7 Utility Command Codes

Code	Utility Command
ARCHIVEJOURNAL	Archive Journal
ARCHIVELOG	Archive Log

Code	Utility Command
BACKUP	Backup
BUILD	Build
CLEANUP	Cleanup Segment/Area
CONVERTCATALOG	Convert Catalog
CONVERTPAGE	Convert Page
EXPANDPAGE	Expand Page
FASTLOAD	Fastload
FIXARCHIVE	Fix Archive
FIXPAGE	Fix Page
FORMAT	Format Area/Segment/File
INSTALLSTAMPS	Install Stamps
LOAD	Load
LOCK	Lock Area/Segment
MAINTAINASF	Maintain ASF
MAINTAININDEX	Maintain Index
MERGEARCHIVE	Merge Archive
PRINTINDEX	Print Index
PRINTJOURNAL	Print Journal
PRINTLOG	Print Log
PRINTPAGE	Print Page
PRINTSPACE	Print Space
PUNCHLOADMODULE	Punch Load Module
RELOAD	Reload
RESTORE	Restore
RESTRUCTURE	Restructure Segment
RESTRUCTURECONNECT	Restructure Connect
ROLLBACK	Rollback
ROLLFORWARD	Rollforward/Extract Journal
SETOPTIONS	Set BCF/OCF options
SYNCHRONIZESTAMPS	Synchronize Stamps
TUNEINDEX	Tune Index

Code	Utility Command
UNLOAD	Unload
UNLOCK	Unlock Area/Segment
UPDATESTATISTICS	Update Statistics
VALIDATE	Validate

5.10 IDMSBCF Input/Output Reassignment

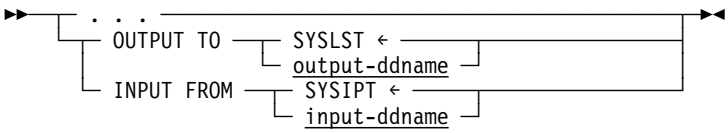
This feature enhances IDMSBCF with the ability to:

- Direct the output of IDMSBCF commands to a file rather than to SYSLST. The output file contains only the data, no headings. The data are represented in string format, not in the native format.
- Read the input to IDMSBCF from a file other than SYSIPT.

This feature enables you to create output from IDMSBCF that is not restricted to the present SYSLST restrictions and then use this (or other input) as input to IDMSBCF (without the restrictions of SYSIPT). The source of the input or target for the output is controlled by the SET OPTIONS command.

5.10.1 Syntax

Expansion of format-option



5.10.2 Parameters

OUTPUT TO
(IDMSBCF batch only) Specifies where to write data output.

SYSLST
Writes data output to SYSLST. If the prior assignment of the OUTPUT stream was not SYSLST, the prior OUTPUT assignment is closed.

output-ddname
Specifies the z/OS or OS/390 DD name, z/VSE or VSE/ESA file name, or BS2000/OSD link name of a sequential dataset to use for writing the data output.

When output is assigned to *output-ddname*, these rules apply:

- WIDTH PAGE is automatically set to the record length (or maximum record length for variable record files) that was specified when the file associated with *output-ddname* was created. If no record length and record format were specified, the record format defaults to variable and the record length to block size - 4; if no block size was specified, a block size of 4096 is used.
- HEADINGS are set to OFF.
- The "non-data" information like the echoed command, eventual headers, the number of rows processed, and the SQL return code are output to SYSLST.

- Output data are not prefixed by "*+".
- The *output-ddname* file is closed on the next SET OPTIONS OUTPUT or at program end.

INPUT FROM

(IDMSBCF batch only) Specifies where to read input.

SYSIPT

Reads input from SYSIPT.

input-ddname

Specifies the DD name of a sequential dataset to use for reading commands.

When input is assigned to *input-ddname*, these rules apply:

- Input from *input-ddname* can be any type and length supported by the operating system, that is, input is not limited to 80 character lines.
- Columns 73 through 80 of the input are NOT considered as a line sequence number, that is, they should contain valid input data.
- End-of-file on the *input-ddname* file automatically reassigns input to SYSIPT.

5.10.3 Usage

Input and output assignment: You can use the OUTPUT TO parameter to output the resulting data of, for example, SQL commands to an intermediate file, which can then be used as input to IDMSBCF or a user written program.

Combining the OUTPUT TO and INPUT FROM parameters allow you to write IDMSBCF scripts to:

- Unload/Load or copy of selective table(s) using SQL DML.
- Automatic access module recompile script for all access modules that are affected by an update statistics or change in table or any other condition that can be detected by looking in the catalog/dictionary.
- Build LOAD file for loading data using SQL DML.
- Build XML scripts to unload/load data from/to Advantage CA-IDMS to/from XML documents.

Output data:

- The data are represented in string format, not in the native format. For example, a column defined as INT with value 12345678 is internally stored as a 4-byte binary value X'00BC614E'; in the output data however, the column value is 8-byte character string '12345678'.
- The width of each column in the output file is determined by the larger value of the column width and the column header. For example, a column named "Date", defined as CHAR(10) uses 10 positions in the output file; a column named "MiddleInitial", defined as CHAR(1) uses 13 positions.

- IDMSBCF inserts two blanks in between successive columns.

5.10.4 Example

Sample IDMSBCF script: The IDMSBCF example below is a fairly generic script to unload/load or copy a table or set of tables. The sample script allows null values; however, it does not allow data containing quotes, more exotic data types, such as GRAPHIC, VARGRAPHIC, BINARY, etc.

Input Script

```
------------------------------------------------------------------------
-- This scripts copies the rows from a source table to a target table.
-- It is assumed that the target table is already defined
------------------------------------------------------------------------
--
-- Helper view to set the params of the Table copy
--
drop   view defje01.CopyTabParm;
create view defje01.CopyTabParm as
  select SCHEMA     as SrcSchema
       , Name       as SrcTable
       , 'DEFJE01'  as TgtSchema  -- Set value of TgtSchema
       , 'EMPLOYEE' as TgtTable   -- Set value of TgtTable
   from SYSTEM.TABLE
  where SCHEMA = 'DEMOEMPL'       -- Set value of SrcSchema
    and NAME   = 'EMPLOYEE'       -- Set value of SrcTable
;

--
-- Create the Unload syntax
--
set options OUTPUT to Unload;

select 'select ''insert into '
     || trim(TgtSchema) || '.' || trim(TgtTable)
     || ' VALUES( '''
     , '-'||'-', 0 as sequence
from defje01.CopyTabParm
union
SELECT
 '||''' || SUBSTR(', ', CAST(1/NUMBER as SMALLINT) + 1, 1)
 || '''||TRIM(VALUE('
 || SUBSTR('CAST(      ''''''''||     ''''''''||CAST('
         , (11 * (  LOCATE(TYPE, 'CHARACTER           ', 1)
                  + LOCATE(TYPE, 'VARCHAR             ', 1)
                  + LOCATE(TYPE, ' DATE               ', 1)) + 1)
         , 11)
 || trim(NAME) || ' '
 || SUBSTR(
    'as char(10))      ||''''''''            as char(10))||'''''''''
         , (18* (  LOCATE(TYPE, 'CHARACTER           ', 1)
                 + LOCATE(TYPE, 'VARCHAR             ', 1)
                 + LOCATE(TYPE, ' DATE               ', 1)) + 1)
         ,18)
 || ','''NULL'''))'
```

```
  , '-'||'-', NUMBER as sequence
FROM SYSTEM.COLUMN, defje01.CopyTabParm
 WHERE TABLE  = SrcTable
   and schema = SrcSchema
union
select '||'');'' from '
     || trim(SrcSchema) || '.' || trim(SrcTable) || ';'
     , '-'||'-', 99999 as sequence
from defje01.CopyTabParm
order by sequence
;

--
-- Create the Load syntax for the new Table
--
set options OUTPUT to Load;
set options INPUT from Unload;

--
-- Load the new Table
--
set options OUTPUT to SYSLST;
set options INPUT from Load;
```

Output from Sample Generic Table Copy Script

Unload OUTPUT

```
select 'insert into DEFJE01.EMPLOYEE VALUES( ' -- 0
||' '||TRIM(VALUE(CAST(      EMP_ID as char(10))      ,'NULL'))           --  1
||','||TRIM(VALUE(CAST(      MANAGER_ID as char(10))      ,'NULL'))       --  2
||','||TRIM(VALUE(''''||     EMP_FNAME ||''''            ,'NULL'))       --  3
||','||TRIM(VALUE(''''||     EMP_LNAME ||''''            ,'NULL'))       --  4
||','||TRIM(VALUE(CAST(      DEPT_ID as char(10))      ,'NULL'))         --  5
||','||TRIM(VALUE(''''||     STREET ||''''             ,'NULL'))         --  6
||','||TRIM(VALUE(''''||     CITY ||''''            ,'NULL'))            --  7
||','||TRIM(VALUE(''''||     STATE ||''''            ,'NULL'))           --  8
||','||TRIM(VALUE(''''||     ZIP_CODE ||''''            ,'NULL'))        --  9
||','||TRIM(VALUE(''''||     PHONE ||''''            ,'NULL'))           --  10
||','||TRIM(VALUE(''''||     STATUS ||''''            ,'NULL'))          --  11
||','||TRIM(VALUE(CAST(      SS_NUMBER as char(10))      ,'NULL'))       --  12
||','||TRIM(VALUE(''''||CAST(START_DATE as char(10))||'''','NULL'))      --  13
||','||TRIM(VALUE(''''||CAST(TERMINATION_DATE as char(10))||'''','NULL')) --  14
||','||TRIM(VALUE(''''||CAST(BIRTH_DATE as char(10))||'''','NULL'))      --  15
||');' from DEMOEMPL.EMPLOYEE; -- 99999
```

Load OUTPUT

```
insert into DEFJE01.EMPLOYEE VALUES(  2299,NULL,'Samuel
','Spade               ',4600,'47 London St
','Canton              ','MA','02020    ',NULL,'L',33892200,'1991-02-04',NULL,'1958-01-09');
insert into DEFJE01.EMPLOYEE VALUES(  3411,2894,'Catherine
','Williams            ',5200,'566 Lincoln St
','Boston              ','MA','02010    ',NULL,'A',83356561,'1993-09-30',NULL,'1967-10-28');
insert into DEFJE01.EMPLOYEE VALUES(  4773,3082,'Janice
','Dexter              ',3510,'399 Pine St
','Medford             ','MA','02432
','5083847566','A',89675632,'1997-06-14',NULL,'1969-11-19');

   ...
   ...
   ...
```

```
insert into DEFJE01.EMPLOYEE VALUES(  3118,3222,'Alan
','Wooding              ',4500,'196 School St
','Canton               ','MA','02020
','5083766984','A',98746783,'1992-11-18',NULL,'1969-05-17');
insert into DEFJE01.EMPLOYEE VALUES(  3769,2894,'Julie
','Donelson             ',3520,'14 Atwood Rd
','Grover               ','MA','02976
','5084850432','A',67783532,'1994-08-31',NULL,'1967-08-15');
```

5.11 Online Compiler Enhancements

The online compilers (DDDL, SCHEMA, SUBSCHEMA, and SYSGEN) and the online command facility (OCF) have been enhanced to increase the maximum data lines they can display from 20,916 lines to 41,916 lines. Also, the CV node name has been added in the command line header (line 1 on the display).

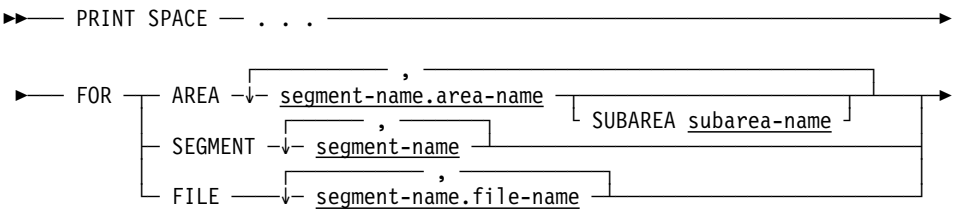
Example of IDD Online Compiler Display

```
Command area     Compiler name     Message        Dictionary/   Current/  CV node
                 and release       area           database      last line name
      ↓               ↓             ↓               ↓             ↓        ↓

                   IDD 16.0       NO ERRORS      DICT=SYSDICT  1/29497 SYSTEM72

     DISPLAY RECORD NAME IS DC-AID-CONDITION-NAMES VERSION IS 1 .
*+   ADD
*+   RECORD NAME IS DC-AID-CONDITION-NAMES VERSION IS 1
*+       DATE CREATED IS      11/30/93
*+       TIME LAST UPDATED IS 11131732
*+       PREPARED BY HARRU01
*+       RECORD LENGTH IS 1
*+       PUBLIC ACCESS IS ALLOWED FOR ALL
*+       RECORD NAME SYNONYM IS DC-AID-CONDITION-NAMES VERSION 1
*+       .
*+       RECORD ELEMENT IS DC-AID-IND-V VERSION 1
*+       LINE IS 000100
*+       LEVEL NUMBER IS 03
*+       PICTURE IS  X
*+       USAGE IS DISPLAY
*+       ELEMENT LENGTH IS 1
*+       POSITION IS 1
*+       .
*+       SUBORDINATE ELEMENT IS ENTER-HIT VERSION 1
*+       LINE IS 000200
*+       LEVEL NUMBER IS 88
*+       USAGE IS CONDITION-NAME
```

5.12 PRINT SPACE Utility Enhancement

The PRINT SPACE utility now provides the capability to report on space utilization on a SUBAREA within an area.

5.12.1 Syntax



5.12.2 Parameters

AREA

Directs the PRINT SPACE utility to report on space utilization in one or more areas or subareas. If no SUBAREA clause is specified, this option produces a report for the entire area plus a report for each file in the area. If a SUBAREA clause is specified, reporting is restricted to the specified subarea.

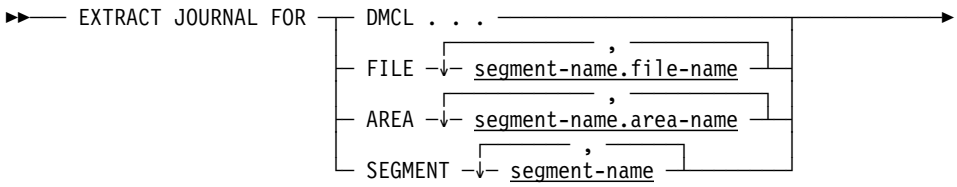
subarea-name

Specifies the name of the subarea associated with the area.

5.13 EXTRACT JOURNAL Utility Enhancement

The EXTRACT JOURNAL utility now provides processing of multiple segments.

5.13.1 Syntax



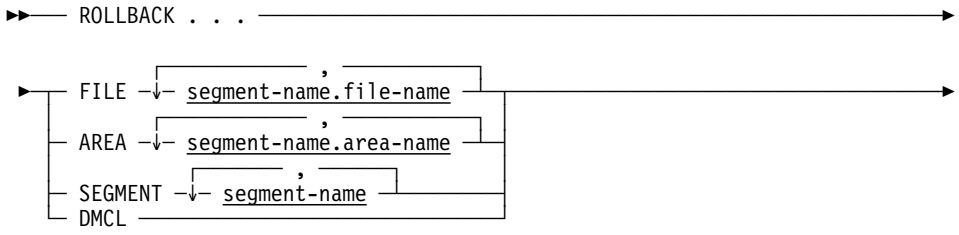
5.13.2 Parameters

SEGMENT
Includes dbkeys for all areas defined in the specified segments in the extract file.

5.14 ROLLBACK Utility Enhancement

The ROLLBACK utility now provides processing of multiple segments.

5.14.1 Syntax



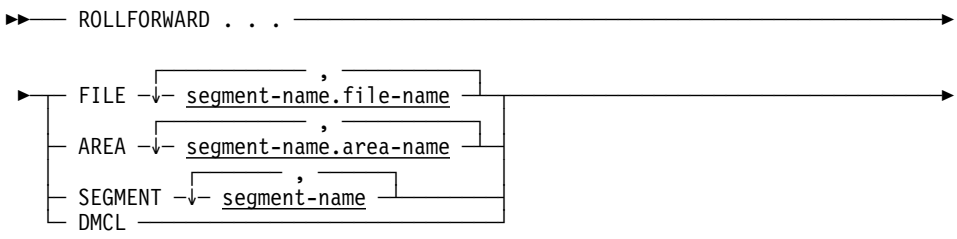
5.14.2 Parameters

SEGMENT
Rolls back and unlocks all areas associated with the specified segments.

5.15 ROLLFORWARD Utility Enhancement

The ROLLFORWARD utility now provides processing of multiple segments.

5.15.1 Syntax



5.15.2 Parameters

SEGMENT
Restores and unlocks all areas associated with the specified segments.

5.16 System Startup Enhancements

In z/OS or OS/390, z/VM, and BS2000/OSD, system startup now allows runtime options to be specified as keyword/value pairs making them easier to code and understand. Additionally, in z/VSE or VSE/ESA, z/OS or OS/390, and z/VM, startup supports the specification of additional options which eliminate the need for coding a #DCPARM macro and linking exit modules with the startup routine. This in turn enables a single startup module to be used for multiple DC/UCF systems making product installation and maintenance easier. On BS2000/OSD, IDMSMOD function DCPARM still has to be executed once, but all parameters can be overridden with BS2KSTAR parameters.

Runtime options are specified through the PARM field of the EXEC statement in z/OS or OS/390 and z/VSE or VSE/ESA, and in the PARM field of the OSRUN command in z/VM or as BS2KSTAR parameters on BS2000/OSD. The following z/OS EXEC statement shows how runtime options are coded using freeform keyword/value pairs. This example starts DC/UCF system version 74 as a multitasking system with 3 subtasks. It also specifies to enable RRS and to use DMCL CVDMCL74 to access the database.

```
//SYSTEM74  EXEC PGM=IDMSDC,PARM='DMCL=CVDMCL74,S=74,MT=Y,RRS=Y,SUBTASKS=3'
```

Note: In z/OS or OS/390, the positional format for specifying runtime options is still supported for upward compatibility.

All options that can be specified through the #DCPARM macro can now be specified as runtime options. This makes use of the #DCPARM macro optional. If used, the values specified through the #DCPARM parameters are treated as defaults that can be overridden through runtime options.

Additionally, runtime options can now be used to identify the write-to-operator and write-to-operator-reply exit modules to be used. By identifying these at runtime, you eliminate the need for linking these modules as part of the system startup routine.

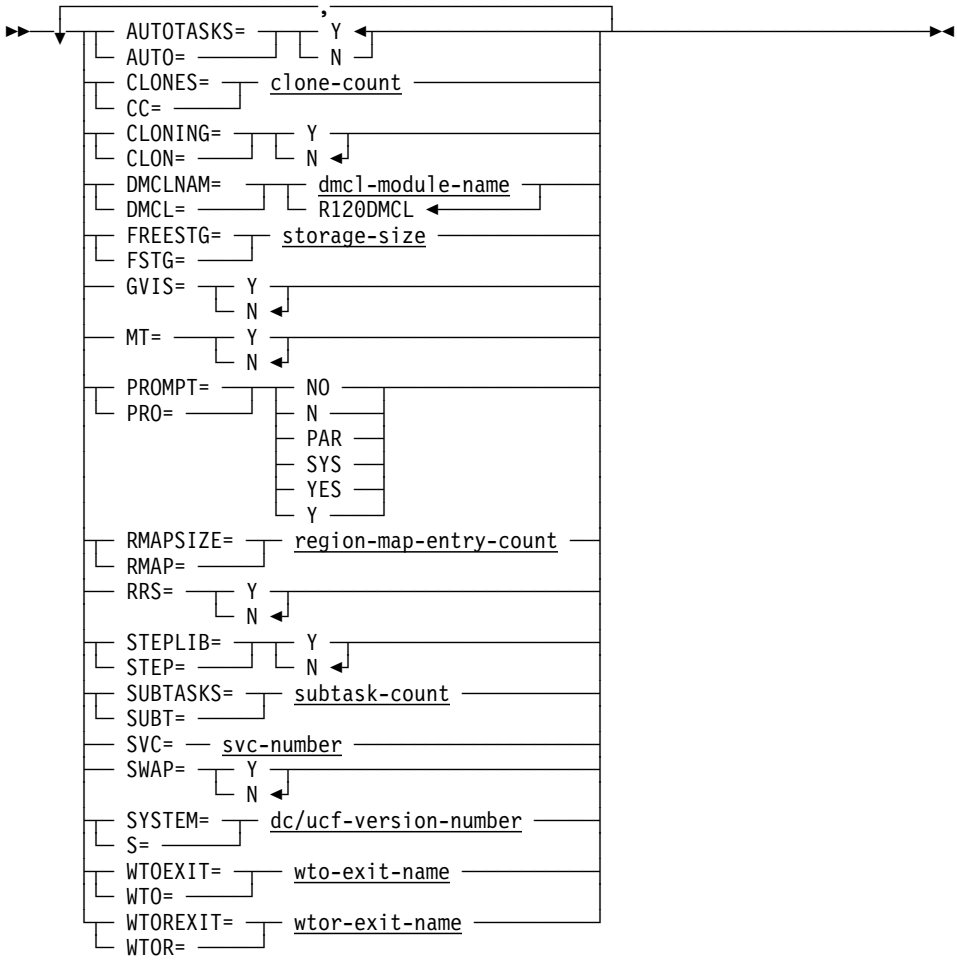
Because neither a tailored #DCPARM macro assembly nor exit modules need to be linked with the startup routine, it is now possible to use a single startup routine for multiple DC/UCF systems. In z/OS or OS/390 and z/VM, a startup routine called IDMSDC is now created during installation that has neither exit modules nor a #DCPARM module included. You can use this as the startup routine for your DC/UCF systems provided that you specify all necessary runtime options through the PARM parameter of the EXEC or OSRUN statement.

Notes:

- In z/VM, the IDMSDC startup routine is linked with the USVCOPT module that was created by assembling the #SVCOPT macro during installation.
- In BS2000/OSD, the CV startup module is created by IDMSMOD customization during installation. Each BS2KSTAR input parameter must be specified on a separate line, without any comma.

Following, is a description and examples of the new runtime option syntax.

5.16.1 Syntax



5.16.2 Parameters

AUTOTASKS/AUTO=Y/N

(z/OS or OS/390, z/VM, and z/VSE or VSE/ESA systems only) Specifies whether to execute startup autotasks.

Valid values are:

- Y specifies to execute startup autotasks
- N specifies not to execute startup autotasks

CLONES/CC=clone-count

(z/OS or OS/390 systems only) Specifies the maximum number of clones to use. *clone-count* must be a positive integer.

CLONING/CLON=Y/N

(z/OS or OS/390 systems only) Specifies whether to activate system cloning.

Valid values are:

- Y specifies to activate cloning
- N specifies not to activate cloning

DMCLNAM/DMCL=dmcl-module-name

Identifies the DMCL to be used by the DC/UCF system. *dmcl-module-name* must be the name of a DMCL module residing in the DC/UCF load (core image) library.

This parameter is optional; however if not specified, you must specify the DMCL to use through a #DCPARM macro.

Note: The abbreviated form of this parameter is not valid for BS2000/OSD.

FREESTG/FSTG=storage-size

Specifies the amount of storage, in K bytes, to be returned (freed) to the operating system at DC/UCF startup time. The storage is freed for operating system use during DC/UCF operations. *storage-size* must be a positive integer.

If not specified and a #DCPARM module is used, the amount of storage to be freed defaults to that specified in the assembled #DCPARM macro; otherwise, if no #DCPARM module is used, 512 is the default.

GVIS=Y/N

(z/VSE or VSE/ESA systems only) Specifies the type of storage management to use.

Valid values are:

- Y specifies to use GETVIS storage management
- N specifies to use COMREG storage management

MT=Y/N

(z/OS or OS/390 systems only) Specifies whether the system runs in multitasking mode.

Valid values are:

- Y specifies to run the system in multitasking mode
- N specifies to run the system in unitasking mode

PROMPT/PRO=NO/N/PAR/SYS/YES/Y

(z/OS or OS/390, z/VM, and z/VSE or VSE/ESA systems only) Specifies whether and for what information to prompt the operator during startup.

Valid values are:

- NO/N specifies not to prompt the operator for any information during startup.
- PAR specifies to prompt the operator for system generation related options.
- SYS specifies to prompt the operator for the version of the DC/UCF system to be started.

- YES/Y specifies to prompt the operator for both system generation related options and the DC/UCF system version number.

If no PROMPT option is specified and a #DCPARM module is used, the prompt option defaults to that specified in the assembled #DCPARM macro; otherwise, if no #DCPARM module is used, NO is the default.

►► For information about how operators respond to the startup prompts, see *Advantage CA-IDMS System Tasks and Operator Commands*.

RMAPSIZE/RMAP=region-map-entry-count
(z/OS or OS/390, z/VM, and z/VSE or VSE/ESA systems only) Specifies the number of entries to allocate in the DC/UCF region map. *region-map-entry-count* must be a positive integer.

If not specified and a #DCPARM module is used, the number of entries in the region map defaults to that specified in the assembled #DCPARM macro; otherwise, if no #DCPARM module is used, 30 is the default.

The default region map entry count should satisfy most sites; however, if a system uses many optional features, (for example, many line drivers), you may have to increase this value. Issue a DCMT DISPLAY MEMORY MAP command to determine if the map displays all the modules you think it should.

RRS=Y/N
(z/OS or OS/390 systems only) Specifies whether to enable RRS support.

Valid values are:

- Y specifies to enable RRS support
- N specifies not to enable RRS support

STEPLIB/STEP=Y/N
(z/OS or OS/390 systems only) Specifies from which library to load RHDCCKUR and RHDCTCKR.

Valid values are:

- Y specifies to load from the STEPLIB concatenation
- N specifies to load from the CDMSLIB concatenation

SUBTASKS/SUBT=subtask-count
(z/OS or OS/390 systems only) Specifies the number of subtasks (TCBs) to use. *subtask-count* must be a positive integer.

This parameter is ignored unless either multitasking or RRS is enabled. If not specified and multitasking or RRS is enabled, the number of TCB's defaults to the number of CPU's available to the operating system.

SVC=svc-number
(z/OS or OS/390 systems only) Identifies the Advantage CA-IDMS SVC number to use during initial system startup. *svc-number* must be the number of an active Advantage CA-IDMS SVC.

Specifying an SVC number allows Advantage CA-IDMS load modules to reside in a PDSE without running Advantage CA-IDMS as an authorized program.

SWAP=Y/N

(z/OS or OS/390 systems only) Specifies whether to run the system as swappable.

Valid values are:

- Y specifies to run the system as swappable
- N specifies not to run the system as swappable

SYSTEM/S=dc/ucf-version-number

(z/OS or OS/390, z/VM, and z/VSE or VSE/ESA systems only) Identifies the DC/UCF system to be started. *dc/ucf- version-number* must be the version number of the target system.

This parameter is optional; however if not specified, then you must either specify the system to start through a #DCPARM macro or enable the operator to be prompted for the system version number.

WTOEXIT/WTO=wto-exit-name

Identifies the write-to-operator (WTO) exit to be used by the DC/UCF system if none is linked with the startup module. *wto-exit-name* must be the name of a WTO exit module residing in the DC/UCF load (core image) library.

For z/OS or OS/390, z/VM, and BS2000/OSD, if this parameter is not specified, you must link the WTO exit module with the CV startup module if you want to exploit the WTO exit.

For z/VSE or VSE/ESA, if this parameter is not specified and no WTO exit module is linked with the startup module, the system tries to load the default WTO exit module named WTOEXIT.

Note: The abbreviated form of this parameter is not valid for BS2000/OSD.

WTOREXIT/WTOR=wtor-exit-name

Identifies the write-to-operator-reply (WTOR) exit to be used by the DC/UCF system if none is linked with the startup module. *wtor-exit-name* must be the name of a WTOR exit module residing in the DC/UCF load (core image) library.

For z/OS or OS/390, z/VM, and BS2000/OSD, if this parameter is not specified, you must link the WTOR exit module with the CV startup module if you want to exploit the WTOR exit.

For z/VSE or VSE/ESA, if this parameter is not specified and no WTOR exit module is linked with the startup module, the system tries to load the default WTOR exit module named WTOREXIT.

Note: The abbreviated form of this parameter is not valid for BS2000/OSD.

5.16.3 Examples

- The following OSRUN command in z/VM starts system version 400 with DMCL DMCL400.

```
"OSRUN IDMSDC PARM='SYSTEM=400,DMCL=DMCL400'"
```

- In the VSE example below, the EXEC IDMSDC statement includes a PARM parameter that requests the GETVIS storage allocation technique and overrides the DMCL used by the DC/UCF system.

```
// EXEC IDMSDC,SIZE=40K,PARM='GVIS=Y,DMCL=CVDMCL'
```

- The following z/OS example starts DC/UCF system version 74 with DMCL CVDMCL74 and write-to-operator exit WTOEXIT.

```
//SYSTEM74 EXEC PGM=IDMSDC,PARM='DMCL=CVDMCL74,S=74,WTO=WTOEXIT'
```

- The following BS2000/OSD example starts DC/UCF system version 74 with DMCL CVDMCL74 and write-to-operator exit PRODWTOX.

```
/ASSIGN-SYSDTA TO=*SYSCMD
/START-PROG*MOD(ELEM=BS2KSTAR,LIB=idms.dba.loadlib,RUN-MODE=*ADV)
DMCLNAM=CVDMCL74
WTOEXIT=PRODWTOX
SYSGEN OVERRIDES FOLLOW
74
END
```

5.17 #WTL Macro Enhancements

The #WTL macro has been enhanced with two new parameters to allow the passing of the CV node name and the Advantage CA-IDMS/DC release number.

This section describes only the new parameters. For more information, see the *Advantage CA-IDMS IDD DDDL Reference Guide*.

Message occurrence structure

Operand	Replacement value
&$7	*Advantage CA-IDMS/DC system node name* (from the SDSNODE field in the SDS block)
&$8	*Advantage CA-IDMS/DC release number*

Chapter 6. Performance Enhancements

6.1 Overview

Release 16.0 provides the following performance enhancement features that are described in this chapter:

- File cache in memory
- Parallel access volume exploitation
- Improved PDSE support
- Improved performance for LE COBOL programs
- Improved journaling performance
- Improved recovery performance
- High performance storage protection

6.2 File Cache in Memory

With the introduction of 64-bit hardware and the z/OS 1.2 operating system, the amount of virtual storage available to an application increased to an incomprehensible amount of 16 exabytes. In Release 16.0, Advantage CA-IDMS can exploit this high amount of storage by caching entire database files in memory.

The major benefits of this feature are:

- Reduced number of I/Os
- Increased throughput
- Less CPU usage

For more information on 64-bit addressing, see the IBM manual, *z/OS MVS Extended Addressability Guide*.

6.2.1 Terminology

The following terms are used in this discussion of file cache in memory:

- *The bar* — The bar marks the 2-gigabyte limit of 31-bit addressing. This is analogous to *the line*, which marks the 16-megabyte limit of 24-bit addressing.
- *Z-storage* — Virtual storage above the bar.

6.2.2 Exploiting File Cache in Memory

Database files with a high number of I/Os are good candidates for the file cache in memory feature. The DBA should use standard performance-monitoring tools to determine which database files these are. Once the decision is made as to which files will use this feature, the DBA should perform these steps:

- Compute the total amount of storage that is needed to cache the selected files. To do this, for each file multiply the number of blocks in the file by the file's block size and sum all results. This sum is the total amount of Z-storage needed.
- Make sure that the jobs that use the modified DMCL have enough Z-storage (at least the amount computed above) at their disposal. The amount of Z-storage available to a job is limited by the MEMLIMIT parameter. For an explanation of MEMLIMIT, see the IBM manual, *z/OS MVS Extended Addressability Guide*.

  You can set MEMLIMIT in different ways:

  – Through an installation default. For more information, refer to the IBM manual *z/OS MVS Initialization and Tuning Reference*.

  – In the JOB and EXEC statements. For more information, see the IBM manual, *z/OS MVS JCL Reference*.

  – Through an installation exit. For more information, see the IBM manual, *z/OS MVS Installation Exits*.

- Change the DMCL definition for each file to specify MEMORY CACHE YES. For details, see 6.2.3, “Altering the DMCL Definition” on page 6-5.

  Note: If your DMCL contains file overrides from a previous release directing the use of dataspace caching, Release 16.0 automatically uses Z-storage instead of dataspaces when executing in a z/OS 1.2 or later operating system. As of Release 16.0, the term "memory cache" replaces "dataspace" in syntax and on displays.

Note: Exploitation of 64-bit storage is incompatible with subspaces. If an attempt is made to use 64-bit storage in an address space that previously used subspaces, an ABEND (system DC2 with reason code 0012) is forced. Therefore, make sure that a central version or batch application that caches files in memory never reuses an address space in which subspaces were used. One way to accomplish this is to define distinct job classes for Advantage CA-IDMS and have address spaces that can execute only these job classes. CICS is known to use subspaces. For more information about subspaces, see the IBM manual, *z/OS MVS Extended Addressability Guide*.

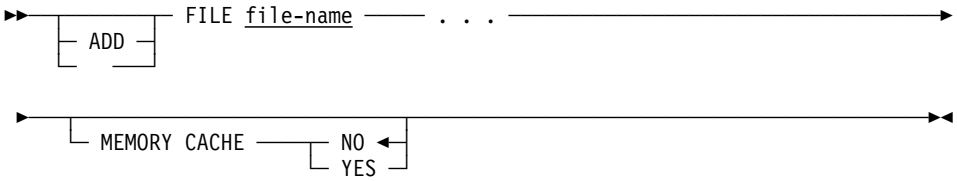
6.2.3 Altering the DMCL Definition

To exploit memory caching, alter your DMCL definition as follows:

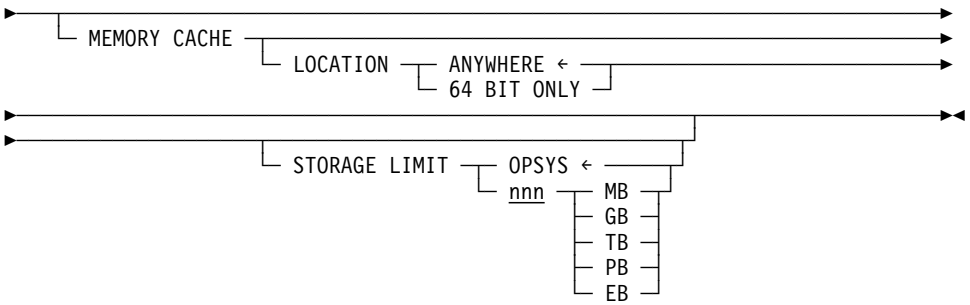
- Include a file override specification for each file that you want to cache in memory.
- Specify the desired MEMORY CACHE options to set global control options.

6.2.3.1 Syntax

Expansion of file-override-specification



Memory Cache Options



6.2.3.2 Parameters

YES
Indicates the file is cached in memory.

NO
Indicates the file is not cached in memory.

MEMORY CACHE
Indicates global options for memory cache:

LOCATION
Indicates where to allocate the storage for memory cache:

ANYWHERE
Memory cache storage is allocated from 64-bit storage; if no or not enough 64-bit storage is available, dataspace storage is acquired.

64 BIT ONLY
Memory cache storage is allocated from 64-bit storage; if no or not enough 64-bit storage is available, memory caching fails.

STORAGE LIMIT
Controls the amount of storage used for memory caching:

OPSYS
Memory cache storage can be acquired until the operating system limit is reached. For 64-bit storage, the operating system limit is set through the MEMLIMIT parameter; for dataspace storage, the limit is optionally imposed by an operating system exit.

nnn **MB, GB, TB, PB, EB**
Advantage CA-IDMS controls the amount of memory cache storage if the value specified is smaller than the operating system limit. nnn must be a positive value between 1 and 32767. MB, GB, TB, PB, EB indicate the unit in which nnn is expressed. The abbreviations stand for Mega Byte (2**20), Giga Byte (2**30), Tera Byte (2**40), Peta Byte (2**50), and Exa Byte (2**60).

6.2.3.3 Usage

DATASPACE versus MEMORY CACHE: The MEMORY CACHE clause replaces the use of the DATASPACE clause. The latter is still accepted for upward compatibility, but is no longer generated on displays. The choice of whether to cache a file in memory or in a dataspace is determined at runtime based on the operating system:

- In a z/OS 1.2 or later environment, files are cached in Z-storage.
- In earlier releases of the operating system files are cached in dataspaces.

Controlling Memory Cache: Use the DMCL-wide MEMORY CACHE options to control where and how much memory cache storage can be allocated.

Insufficient storage for memory cache: If MEMORY CACHE YES is specified and not enough storage is available to cache a file in memory, processing continues as if MEMORY CACHE NO was specified.

Dynamically changing memory cache specification: The MEMORY CACHE YES specification can be changed dynamically:

- Use DCMT VARY DMCL to change DMCL-wide MEMORY CACHE options
- Use DCMT VARY FILE to change the MEMORY CACHE specification for a file.

For more information on DCMT VARY DMCL and DCMT VARY FILE, see Appendix A, “New and Revised DCMT Commands.”

6.3 Parallel Access Volume Exploitation

This feature provides Advantage CA-IDMS I/O performance improvements through exploitation of the Parallel Access Volume feature on Enterprise Storage System DASD devices, such as IBM's Shark. This feature allows multiple users and multiple jobs to simultaneously access the same logical volume and perform concurrent I/Os to a file.

PAV devices support multiple concurrent I/Os against the same disk unit. However, by default PAV devices ensure that multiple I/Os to the same disk extent are single-threaded. This is known as collision checking. Because Advantage CA-IDMS routinely issues concurrent I/Os to the same extent, collision checking prevented full exploitation of PAV devices. Since Advantage CA-IDMS ensures that the I/O requests it issues do not conflict with each other, IDMS is able to disable collision checking, allowing PAV devices to be fully exploited.

When disk I/Os for the same file are waiting because of disk extent collision checking, implementing PAV support reduces I/O wait times. Reduced I/O wait times should increase transaction throughput and improve response times.

PAV support occurs automatically when a file is on a properly defined PAV device and does not occur otherwise. The systems programmer is responsible for defining the device to the operating system correctly. For example, if no alias Unit Control Blocks (UCBs) are defined for a PAV device, the I/Os are single threaded on the primary UCB and negate the advantage of no collision checking.

6.4 Improved PDSE Support

PDSEs provide two primary capabilities:

- It is the only library capable of containing load modules of greater than 16 Megabytes. Although Advantage CA-IDMS does not require such support, it is anticipated that clients will require this support in the future.
- PDSEs do not require condensing.

For Release 14.0 Advantage CA-IDMS added support for loading Advantage CA-IDMS programs from a PDSE. This required early initialization of the Advantage CA-IDMS program call (PC) environment; and you were required to start Advantage CA-IDMS as an authorized program.

In Release 16.0, you can specify an IDMS SVC number on the execute parameter in columns 29-31 of the startup JCL. This SVC acquires sufficient authorization to construct a PC environment, without requiring you to start Advantage CA-IDMS as an authorized program.

6.4.1 Startup JCL Parameters

```
S=nnn              System version number, length is not counted in following info
+0  override options  Passed to RHDCSTRT
+20 C'U' or blank     Uni-tasking
    C'M'              Multi-tasking
    C'R'              Uni-tasking with RRS TCBs
    C'T'              Multi-tasking with RRS TCBs
+21 nn                TCB limit count for Multi-tasking
+23 C'S'              Run swappable
+24 C'C'              Cloned system
+25 nnn               Limit count for searching for available CV number.
+28 nnn               IDMS SVC number
```

6.4.2 Parameter Descriptions

- Column 21 — By default, Advantage CA-IDMS runs in uni-tasking mode. A value of:
  - R — Activates Advantage CA-IDMS's interface to IBM's RRS facility while retaining uni-tasking mode for all other Advantage CA-IDMS facilities.
  - T — Activates Advantage CA-IDMS's interface to IBM's RRS facility and also activates Advantage CA-IDMS's multi-tasking interface.
  - M — Activates Advantage CA-IDMS's multi-tasking interface only.
- Column 22 — If 'M' was specified in column 21, then Advantage CA-IDMS defaults to using one more TCB than the number of CPUs found in the machine or LPAR. If 'M', 'R', or 'T' was specified in column 21, the number of TCBs are specified in columns 22 and 23, left justified, blank filled.

- Column 24 — By default Advantage CA-IDMS runs non-swappable. Specify a 'S' to force Advantage CA-IDMS to run swappable.
- Columns 25 — To activate the cloned system feature, enter a 'C' in this column.
- Column 26 — Enter a number between 1 and 255, left justified and blank filled, to indicate the limit for the search to find an available CV number. The search starts from the system number and wraps at 255. When using this option, the system number and the CV number are identical.
- Column 29 — Specifying a valid Advantage CA-IDMS SVC number permits using a PDSE to contain the Advantage CA-IDMS system load modules. Enter the number of any valid IDMS SVC. It may be the one the system runs with; however, it is not required.

6.4.3 General Usage Rules

- All fields are optional and are only supplied to request the designated option.
- All fields are positional.
- Unused fields are left blank.
- All numeric fields are left justified and blank filled.

6.5 Improved Performance for LE COBOL Programs

New system generation syntax in Release 16.0 enables control over whether multiple COBOL programs in the same task can share a Language Environment (LE) enclave within an Advantage CA-IDMS DC/UCF system. Since creating LE processes and enclaves involves considerable storage and CPU overhead, the ability to eliminate this processing when not needed may significantly improve performance.

The underlying support for this feature was implemented for Release 15.0. It enables multiple LE COBOL programs executing within a task to share a single process and enclave. Since not all programs are eligible to share an enclave, Release 16.0 provides system generation syntax to:

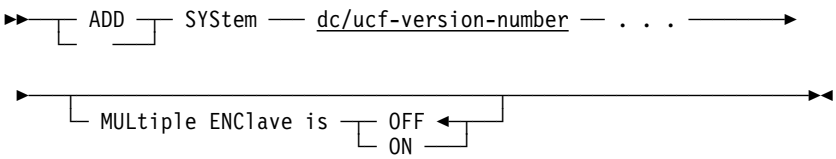
- Controls whether enclave sharing is in effect for a system
- Controls whether an individual program shares an enclave with other programs executing in the same task.

Note: Release 16.0 eliminates the use of optional PTF bit 232 for controlling whether enclave sharing is in effect for a system.

6.5.1 System Generation SYSTEM Statement

Use the system generation SYSTEM statement to specify if multiple COBOL programs within the same task can share an LE enclave.

6.5.1.1 Syntax



6.5.1.2 Parameters

MULtiple ENClave is

Specifies whether the system allows programs in the same task to share an LE enclave. This parameter affects only COBOL programs.

OFF

Specifies multiple enclave support is disabled for the system. This is the default.

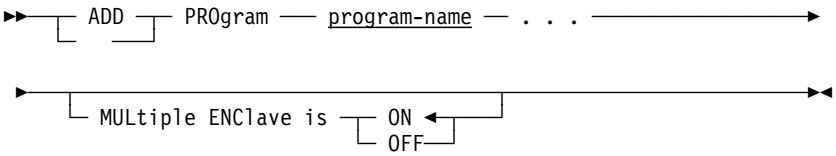
ON

Specifies multiple enclave support is enabled for the system.

6.5.2 System Generation PROGRAM Statement

Use the system generation PROGRAM statement to specify if an individual COBOL program may share an LE enclave with other programs in the same task.

6.5.2.1 Syntax



6.5.2.2 Parameters

MULtiple ENClave is

Specifies if this program can use the same language enclave as other LE programs in the same task. This parameter is only meaningful for COBOL programs.

ON

Specifies that this program can participate in a multiple program LE enclave. This is the default. If enclave sharing is disabled for the system this parameter is ignored.

OFF

Specifies that the program cannot participate in a multiple program LE enclave.

6.6 Improved Journaling Performance

Release 16.0 has improved journaling performance by extending the impact of a non-zero setting for the journal transaction level. Specifying a journal transaction level has the effect of deferring the write of a partially full journal buffer when a transaction terminates provided that the number of active transactions in the system is greater than the transaction level specified. Release 16.0 extends this effect to apply to partially full journal buffers that contain before images for database pages that are being flushed from the buffer. By deferring the journal write, journal efficiency can be improved thereby reducing the number of journaling I/Os.

For more information on specifying a journal transaction level, see the *Advantage CA-IDMS Database Administration Guide*.

6.7 Improved Recovery Performance

Release 16.0 lets you control the following commit and rollback behavior:

- The type of journal record written on a commit
- Whether a new local transaction ID is assigned on a rollback continue or commit

Exploiting these new capabilities may improve recovery time during warmstart and rollback operations and reduce the likelihood of duplicate transaction IDs when the local transaction ID values wrap.

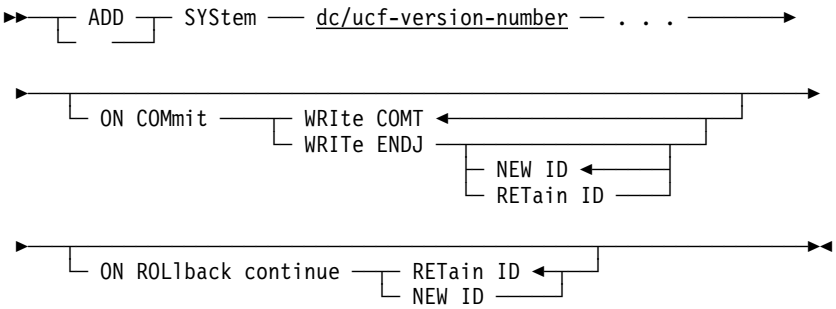
You can implement this feature through new clauses on the system generation SYSTEM and TASK statements. Extensions to the DCMT VARY TASK command and the DCMT VARY DYNAMIC TASK command let you override this setting at runtime.

The system generation statements are explained below. For more information on using the DCMT commands, see Appendix A, “New and Revised DCMT Commands.”

6.7.1 System Generation SYSTEM Statement

Use the system generation SYSTEM statement to specify default commit and backup behavior for a system.

6.7.1.1 Syntax



6.7.1.2 Parameters

ON COMmit
Specifies options that control commit behavior. These options apply only to commit operations in which the database session remains active.

WRIte COMT
Specifies that a COMT journal record should be written.

WRIte ENDJ
Specifies that an ENDJ journal record should be written.

NEW ID
Specifies that a new local transaction ID should be assigned to the next transaction associated with the database session.

RETain ID
Specifies that the existing local transaction ID should be assigned to the next transaction associated with the database session.

ON ROLlback continue
Specifies options that control rollback behavior. These options apply only to rollback operations in which the database session remains active.

RETain ID
Specifies that following a rollback, the current local transaction ID should be assigned to the next transaction associated with the database session.

NEW ID
Specifies that following a rollback, a new local transaction ID should be assigned to the next transaction associated with the database session.

6.7.1.3 Usage

Specifying commit and rollback options: You can specify options that control the following commit and rollback behavior:

- The type of journal record written on a commit
- Whether a new local transaction ID is assigned on a rollback continue or commit

You can control whether a COMT or ENDJ journal record is written on a commit operation in which the database session remains active. Writing an ENDJ can reduce recovery time because less data has to be examined to locate the start of a recovery unit. This benefit applies to online recovery, warmstart, and ROLLBACK and ROLLFORWARD recovery operations. ENDJ is most beneficial in cases where long-running sessions infrequently perform a burst of updates and then issue a commit.

Note: ENDJ journal records are always written when system run units are committed, regardless of the ON COMMIT option specified.

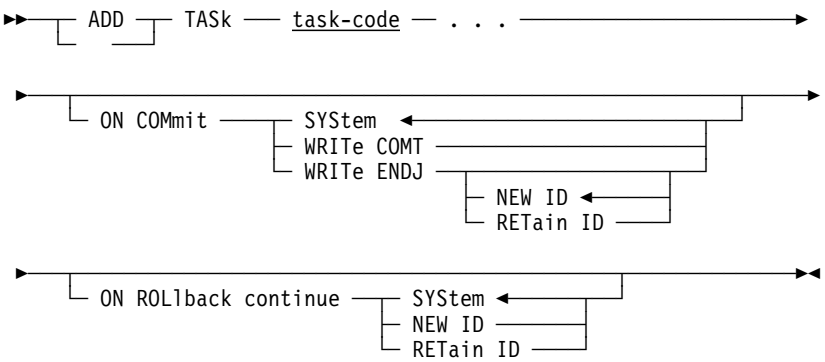
You can control whether a new local transaction ID is assigned following a commit or rollback operation in which the database session remains active. Assigning a new transaction ID reduces the chance of duplicate IDs should this value wrap within a single cycle of a central version. It also has the effect of recording journal statistics for a database session using a different transaction identifier for each recovery unit. You can assign a new ID on a commit operation only if you also specify that an ENDJ checkpoint record be written.

Note: A new transaction ID is always assigned when system run units are committed or rolled out.

6.7.2 System Generation TASK Statement

Use the system generation TASK statement to specify commit and backup behavior for transactions initiated by a specific task.

6.7.2.1 Syntax



6.7.2.2 Parameters

ON COMmit
Specifies options that control commit behavior. These options apply only to commit operations in which the database session remains active.

SYStem
Specifies that the commit behavior for the task should default to that specified for the system.

WRIte COMT
Specifies that a COMT journal record should be written.

WRIte ENDJ
Specifies that an ENDJ journal record should be written.

NEW ID
Specifies that a new local transaction ID should be assigned to the next transaction associated with the database session.

RETain ID
Specifies that the existing local transaction ID should be assigned to the next transaction associated with the database session.

ON ROLlback continue
Specifies options that control rollback behavior. These options apply only to rollback operations in which the database session remains active.

SYStem
Specifies that the rollback behavior for the task should default to that specified for the system.

RETain ID
Specifies that following a rollback, the current local transaction ID should be assigned to the next transaction associated with the database session.

NEW ID
Specifies that following a rollback, a new local transaction ID should be assigned to the next transaction associated with the database session.

6.7.2.3 Usage

Specifying commit and rollback options: You can specify options that control the following commit and rollback behavior:

- The type of journal record written on a commit
- Whether a new local transaction ID is assigned on a rollback continue or commit

You can control whether a COMT or ENDJ journal record is written on a commit operation in which the database session remains active. Writing an ENDJ can reduce recovery time because less data has to be examined to locate the start of a recovery unit. This benefit applies to online recovery, warmstart, and ROLLBACK and ROLLFORWARD recovery operations. ENDJ is most beneficial in cases where long-running sessions infrequently perform a burst of updates and then issue a commit.

Note: ENDJ journal records are always written when system run units are committed, regardless of the ON COMMIT option specified.

You can control whether a new local transaction ID is assigned following a commit or rollback operation in which the database session remains active. Assigning a new transaction ID reduces the chance of duplicate IDs should this value wrap within a single cycle of a central version. It also has the effect of recording journal statistics separately by commit recovery unit rather than across all recovery units within a database transaction. You can assign a new ID on a commit operation only if you also specify that an ENDJ checkpoint record be written.

Note: A new transaction ID is always assigned when system run units are committed or rolled out.

6.8 High Performance Storage Protection

To avoid inadvertently utilizing the High Performance Storage Protection feature, check the sysgens for all existing CVs to ensure that none of them are defaulting to storage protect key 9. Regenerate any system using a storage protect key of 9 to explicitly specify a key in the range of 10 through 15.

With the introduction of the z-architecture machines, the cost of providing storage protect in the Advantage CA-IDMS region has increased dramatically. This is largely due to a combination of the internal architecture of the "MVS-like" operating systems and the number of processors supported in this hardware. The operating system normally employs translation look-aside buffers (TLBs) to improve performance by retaining recently used pages. Unfortunately, this requires the use of processor signals and TLB flushes every time the storage protect key of a page is changed.

However, storage protect key 9 is unique and behaves much differently than other protect keys. By properly setting the control registers, a program can write in key 9 storage regardless of what key the program is currently executing with. This capability has permitted making the use of storage protection much less costly and therefore much more viable for use in the client's production CVs. It is now possible to economically insulate Advantage CA-IDMS from any potential damage caused by user written programs, as well as preventing user written code from invalidly or maliciously invoking privileged functions, and thereby jeopardizing the integrity of both Advantage CA-IDMS and the operating system.

To achieve these performance benefits, we do not make any attempt to protect the non-reentrant user programs or the users' storage from each other. The entire non-reentrant program pool is swapped to key 9 during system startup, as well as every storage pool that was generated to support any form of user storage. This permits swapping to and from user mode to consist solely of changing the PSW key from Advantage CA-IDMS's key to key 9, and back again when the user returns, which results in a dramatically reduced CPU load compared to the use of standard storage protect.

To enable this facility, the DBA must employ key 9 and segregate all user-oriented storage from the Advantage CA-IDMS system storage. Storage types: user, user-kept, shared, and shared-kept, can be together, but they must be defined to secondary storage pools and must be isolated from any secondary pools that contain database or terminal type storage. Use of this feature will probably be done only on the production systems. The programmers still have full access to storage protection during coding and debugging by generating their test systems to use some key other than 9.

An additional benefit is that the client is no longer forced to choose between speed and operating system integrity. Performance testing has revealed that using this option is virtually indistinguishable from running without storage protect in terms of CPU cost.

Assuming that a storage protected system has already been successfully created, the following steps can be used to enable this feature:

1. On the existing system, display all the storage pools (DCMT DISPLAY ALL STORAGE POOLS), taking note of what pools support any type of user storage, that is, user, user-kept, shared, shared-kept, or ALL.
2. The storage pools must be defined in such a manner that all forms of user-oriented storage are segregated from the system storage. In other words, define both an XA and a non-XA storage pool for user storage types. Storage types: user, user-kept, shared, and shared-kept, can be together, but they must be defined to secondary storage pools and must be isolated from any storage pools which contain database or terminal type storage.
    a. In sysgen, redefine at least one storage pool in the range 128 to 254 to support types user, user-kept, shared, and shared-kept.
    b. To remove user storage types from pool 0, you must sysgen at least one pool in the range of 1 to 127 that supports types user, user-kept, shared, and shared-kept.

    Note: You cannot define a storage pool or user pool to support type ALL (the default) or a mixture of types user, user-kept, shared, and shared-kept and system (database or terminal) storage.
3. In sysgen, on the system statement, specify STORAGE KEY IS 9.
4. Generate and start the system. If the storage pool definitions have not been properly set, message DC004001 HPSPO HAS BEEN DISABLED DUE TO INCORRECT STORAGE POOL DEFINITIONS is issued at startup.

Note: Any user programs that attempt to perform I/O directly through operating system facilities may need to be modified because they must run with storage protection enabled. This ensures that the storage key and the execution key match, which is an operating system requirement for I/O.

Chapter 7. Non-Stop Processing Features

7.1 Overview

This chapter describes the following features of Release 16.0 that provide increased non-stop processing capability:

- Dynamic trace control
- Ability to modify any program attribute
- New command for determining CPU effectiveness
- "Short on Storage" message
- Enhanced messaging for full journals

7.2 Dynamic Trace Control

Dynamic trace control enables the DBA to alter system and database trace attributes. System and database tracing provide Computer Associates technical personnel with diagnostic information that can be useful when researching problems. System trace attributes are initially established through the SYSTRACE parameter of the system definition SYSTEM statement. Database trace attributes are initially established through the SYSIDMS DB_TRACE_TABLE parameter.

New DCMT commands in Release 16.0 enable the DBA to:

- Display the status of the system trace and the number of entries in the system trace table with the DCMT DISPLAY SYSTRACE command
- Disable the system trace with the DCMT VARY SYSTRACE OFF command
- Enable the system trace or vary the number of entries in the system trace table with the DCMT VARY SYSTRACE ON command
- Display the status of the database trace and the size of the database trace table with the DCMT DISPLAY DBTRACE command
- Disable the database trace with the DCMT VARY DBTRACE OFF command
- Enable the database trace or vary the size of the database trace table with the DCMT VARY DBTRACE ON command

  For more detailed information on how to use these commands, see Appendix A, “New and Revised DCMT Commands.”

7.3 Modifying Program Attributes

The DCMT VARY PROGRAM command has been extended in Release 16.0 to let you modify any program attribute that was defined with the sysgen compiler. The new attributes that you can now modify with this command are listed below:

- CONCURRENT/NONCONCURRENT
- OVERLAYABLE/NONOVERLAYABLE
- REENTRANT/NONREENTRANT/QUASIREENTRANT
- SAVEAREA/NOSAVEAREA
- MAINLINE/NOMAINLINE
- NEW COPY ENABLED/DISABLED
- ISA SIZE *nnn*
- LANGUAGE ADSO/ASSEMBLER/COBOL/PLI
- MPMODE ANY/SYSTEM
- TYPE DIALOG/MAP/PROGRAM/SUBSCHEMA/TABLE

For more detailed information on these attributes and on how to use the DCMT VARY PROGRAM command, see Appendix A, “New and Revised DCMT Commands.”

7.4 Determining CPU Effectiveness

The CPU effectiveness of a central version represents the percentage of time the CPU was available when one or more subtasks of the Advantage CA-IDMS system was ready to run. You can display this CPU effectiveness with the new command DCMT DISPLAY SUBTASK EFFECTIVENESS.

For more detailed information on this command, see Appendix A, “New and Revised DCMT Commands.”

7.5 Short on Storage Message

Release 16.0 writes a new message if a short on storage condition occurs. This makes it easier for a DBA to diagnose a system hang and implement automated emergency procedures.

When a request to obtain storage is processed by the system, there may not be enough storage available to service the request. One of two possibilities could occur:

- A short on storage condition: After the system allocates storage, the total amount of free storage remaining in the storage pool is less than the storage cushion for that pool.
- A storage not available condition: After selecting a storage pool, the Advantage CA-IDMS system determines that not enough contiguous storage was available in the pool to satisfy the request.

In these situations, message DC015007 is written to the console:

```
DC015007 Pool &01: SOS condition &02
```

Where:

- &01 identifies the storage pool number
- &02 is 0 (short on storage) or 1 (storage not available)

The DC015007 message is output at the time the storage allocation algorithm encounters either condition. If storage cannot be allocated from the first selected storage pool, the storage allocation algorithm looks into alternate storage pools if they are defined. As a result, it is possible that one request for storage results in multiple DC015007 messages.

For performance reasons, the frequency with which the DC015007 message is output is limited to one per storage pool per minute.

The DBA can set up automated procedures to send an alert when this error message is detected.

7.6 Waiting on Full Journal Message

Release 16.0 provides enhanced handling for the situation in which the journal files fill because long running transactions do not commit their changes. Such transactions can fill the journals because the ARCHIVE JOURNAL utility is unable to remove the BFOR images for uncommitted transactions. When the journals fill, the system comes to a halt. In order to correct the situation, the task that is filling the journals must be canceled.

To assist in this process, Release 16.0 writes the following message for each task that is waiting to write to a full journal file:

```
DC205024 Journal Write waiting on full Journal
```

The message is repeated every few seconds until tasks are no longer waiting on a full journal.

To recover from this situation:

1. Identify the task that is filling the journal files and abort the task.
2. After its changes are rolled out and an ABRT checkpoint is written, issue a DCMT VARY JOURNAL command so the central version swaps to a new journal and the full journal can be offloaded and condensed by ARCHIVE JOURNAL.

It is likely that DCMT VARY JOURNAL will need to be issued more than once, since several journal files may have filled and require offloading.

Once the system swaps back to the initial journal file on which tasks waited, processing should continue without the need for further intervention.

Chapter 8. Tool Product Enhancements

8.1 Overview

Release 16.0 provides several enhancements to Advantage CA-IDMS tool products. These enhancements are described in this chapter.

8.2 Advantage CA-Culprit

Advantage CA-Culprit is a batch utility that generates reports from conventional and database files. You can store frequently used pieces of code for Advantage CA-Culprit so that several reports or users can access them. Using these stored parameters helps to establish standard file definitions, procedures, and reports. These parameters are stored in the data dictionary (IDD), partitioned datasets (z/OS), source statement libraries (VSE/ESA), AllFusion™ CA-Panvalet® libraries, or AllFusion™ CA-Librarian® libraries.

Prior to Release 16.0, the AllFusion CA-Librarian and AllFusion CA-Panvalet file access routines were linked with Advantage CA-Culprit routines to form the respective interfaces. For Release 16.0, the code for these interfaces has been changed to dynamically load and call the AllFusion CA-Librarian or AllFusion CA-Panvalet routines. You do not have to relink the Advantage CA-Culprit interface module when you install a new maintenance release of AllFusion CA-Librarian or AllFusion CA-Panvalet.

To the user, there is no difference in the AllFusion CA-Librarian or AllFusion CA-Panvalet interfaces with the enhancements to Advantage CA-Culprit in Release 16.0.

8.2.1 Invoking the AllFusion CA-Librarian Interface

To invoke the AllFusion CA-Librarian interface, you must take the steps given below. These procedures are the same as those required before Release 16.0:

- Specify PARMLIB=LIBRARIAN3 in the system profile or on the PROFILE parameter.
- Code =COPY, =MACRO, or USE statements to copy the stored code into the Advantage CA-Culprit parameter input stream at runtime.
- Add a DDNAME for MASTER in the JCL that points to the Librarian library where the source to be copied resides.
- Include the AllFusion CA-Librarian loadlib in the STEPLIB DDNAME.

8.2.2 Invoking the AllFusion CA-Panvalet Interface

To invoke the AllFusion CA-Panvalet interface, you must take the steps given below. These procedures are the same as those required before Release 16.0:

- Specify PARMLIB=PANVALET in the system profile or on the PROFILE parameter.
- Code =COPY, =MACRO, or USE statements to copy the stored code into the Culprit parameter input stream at runtime.
- Add a DDNAME for PANDD1 in the JCL that points to the AllFusion CA-Panvalet library where the source to be copied resides.

- Include the AllFusion CA-Panvalet loadlib in the STEPLIB DDNAME.

8.3 Advantage CA-IDMS Journal Analyzer

Advantage CA-IDMS Journal Analyzer is a comprehensive batch facility that gathers and combines management and performance data from the archived Advantage CA-IDMS journal and reports on it in precise logical formats. Advantage CA-IDMS Journal Analyzer provides the following three distinct types of printed output: reports, displays, and audit information.

Release 16.0 enhancements to Advantage CA-IDMS Journal Analyzer are listed below:

- Enhanced RECORD and DBKEY display processing
- Enhanced Audit Report
- Enhanced Chronological Report

Note: The Release 16.0 version of Advantage CA-IDMS Journal Analyzer is not compatible with previous releases. In addition, pre-release 16.0 versions of Advantage CA-IDMS Journal Analyzer cannot operate with a Release 16.0 archived journal.

8.3.1 RECORD and DBKEY Display Processing

Release 16.0 enhances the parameters controlling the generation of RECORD and DBKEY displays to allow the addition of start and stop dates and times. To add this information to the displays, use the START= and STOP= fields and also specify ALL=Y. If you do not include the new parameters, the displays are identical to those in Release 15.0.

8.3.2 Audit Report

The Audit Report contains informative, error, and processing messages. Release 16.0 enhances this report so that it includes detail counts for all the new distributed transaction record types. These new record types are listed below:

- DPRP: Prepare to Commit
- DIND: Commit In-Doubt
- DCOM: Transaction Committed
- DBAK: Transaction Being Rolled Back
- DPND: Forget Pending
- DFGT: Transaction Forgotten

No user action is required to generate the Distributed Transaction Record statistics in the Audit Report.

For more information on these new journal record types, see 3.6.2, “Journal File Formatting Considerations” on page 3-22.

8.3.3 Chronological Report

Release 16.0 enhances the Chronological Report so that it provides the timestamp and Distributed Transaction Record ID (DTRID) field, as well as details on all distributed transaction records encountered. The Chronological Report now also includes details for any Local ID (LID) records included in each distributed transaction record.

This report is optional; you can request it by specifying REPORT=CHRONO on the controlling parameters.

For more information on transaction identifiers, see 3.3.6.2, “Transaction Identifiers” on page 3-11.

8.4 Advantage CA-IDMS DME

Advantage CA-IDMS DME is an online program development facility used to create, edit, and browse modules stored in the dictionary. Release 16.0 enhancements to Advantage CA-IDMS DME are listed below:

- Invocation via a 'fast-in' access method
- Change in the print class to accept any value
- Browse screen error highlighting

8.4.1 'Fast-In' Access Method

The 'fast-in' access method jumps directly to the Module Selection screen, with an Action Mode of 'E' for Edit selected. The default mode of operation is to display the Main Menu screen as done in previous releases.

To set the 'fast-in' access method, the Database Administrator (DBA) must set the EDITMOD parameter. This can be done at installation or any time thereafter. Set the parameter to Y (Yes) for the 'fast-in' access method; set it to N (No) for the standard access method. The default parameter is N. The DBA must then reassemble and relink the USETPARM source module to create a USETPARM load module.

8.4.2 DME Print Class

In Release 16.0, you can specify any value, including a null value, for the print class. The value selected is displayed on the Main Menu screen of Advantage CA-IDMS DME.

To set the DME print class, the DBA must set the value of PRTCLASS in the USETPARM source module. This can be done at installation or any time thereafter. The DBA must then reassemble and relink the USETPARM source module to create a USETPARM load module.

8.4.3 Browse Screen

Release 16.0 highlights any errors encountered on the compiler browse screen of Advantage CA-ADS. If you are using this compiler and an error occurs, Advantage CA-IDMS DME is invoked (if present) and it displays and highlights the line in error. No user action is required to activate this feature.

8.5 Advantage CA-IDMS DMLO

Advantage CA-IDMS DMLO is an interactive productivity tool that allows on-demand navigation, retrieval, and update of databases in Advantage CA-IDMS/DB Release 16.0 enhancements to Advantage CA-IDMS DMLO are listed below:

- Highlighted exit key
- Help Dictionary default to current working dictionary
- Dynamic message processing in the User Exit Program

8.5.1 Highlighted Exit Key

Release 16.0 alters the initial entry screen for Advantage CA-IDMS DMLO to highlight the key that you should use to exit the tool. This key is the ATTNKEY or the INTERRUPT key.

No user action is needed to activate this feature.

8.5.2 Help Dictionary

Release 16.0 modifies the HELP feature in Advantage CA-IDMS DMLO so that the HELP modules are loaded from the current working dictionary if no HLPDICT setting is specified in the USDTPARM installation parameter module. In previous releases, a null setting for this parameter meant that the help dictionary defaulted to TOOLDICT.

To use the current working dictionary as the default help dictionary, the DBA must set the value of HLPDICT in the installation parameter module USDTPARM to blanks. This can be done at installation or any time thereafter. The DBA must then reassemble and relink the USDTPARM source module to create a USDTPARM load module.

8.5.3 Dynamic Message Processing

Release 16.0 enhances the user exit program USDMLXIT to support dynamic message processing. With this enhancement, USDMLXIT can pass a message to display on the Advantage CA-IDMS DMLO command line.

To use the dynamic messaging feature in the user exit program, the DBA must set a USERCODE value of 99 (decimal) when returning control to Advantage CA-IDMS DMLO A message of up to 64 bytes can be passed for subsequent display. The message text must be stored at address USERWORK. These fields are defined in the USDGLOB2 DSECT.

Note: It is important to initialize the USERWORK field with spaces before passing the message text. If the USERWORK field is not initialized, it may contain data from previous messages.

8.6 Advantage CA-ADS Alive

Advantage CA-ADS Alive is an online tool that allows developers to test dialogs and intercept errors for review and analysis in an online environment. Release 16.0 provides the following enhancements to Advantage CA-ADS Alive:

- The number of records per dialog that Advantage CA-ADS Alive can handle has been increased. Advantage CA-ADS Alive can now process up to 200 records per dialog on the Record Display screen at runtime.
- An Installation Parameter option that disables the Post Abort Browse screen feature has been added. If a dialog abend occurs and you have implemented this disable option, no Process code is displayed. However, a USG0071E message is displayed. The details of the dialog abend continue to be written to the queue DEBUGQUEUE. You can later access these details using the QREVIEW task code.

No action is required to use the increased number of records per dialog.

To set the Post Abort Browse screen option, the DBA must set the value of ABRTSCR in the source module USGTPARM. This can be done at installation or any time thereafter. The DBA must then reassemble and relink the USGTPARM source module to create a USGTPARM load module.

8.7 Online Mapping

Advantage CA-IDMS Mapping Facility is an online and batch development facility used to create, maintain, and display formatted terminal screens, called maps, for communication between a terminal operator and an application program. The formatted screen definitions created by this facility exist as a collection of database records stored in a dictionary.

RHDCMAP1 is the batch compiler that accepts the formatted screen source syntax and creates a dictionary definition of the formatted screen from this syntax. RHDCMAP1 moves the dictionary definitions of the formatted screens from one dictionary to another in the batch environment, for example, from a development dictionary to a QA dictionary or to a production dictionary.

In Release 16.0, RHDCMAP1 has been enhanced to accept up to 40 database records in one map. RHDCMAP1 could previously accept a maximum of only 20 database record references.

8.8 Advantage CA-IDMS PL/I Compiler Enhancements

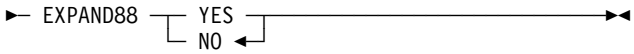
The Advantage CA-IDMS Data Manipulation Language (DML) is comprised of statements that direct Advantage CA-IDMS/DB and data communications processing. DML statements are embedded in the program source as if they are a part of the host language. The DML PL/I compiler, also known as the DMLP processor or pre-processor, performs these tasks:

- Translates the DML statements into PL/I statements
- Retrieves data descriptions and source code from the IDD
- Validates the DML syntax
- Provides an output listing for errors

The INCLUDE IDMS statement directs the DMLP processor to retrieve a record description from the IDD and include it in the application program. The DMLP processor generates the appropriate declare statements for elements and their redefinitions.

In prior releases of Advantage CA-IDMS, the DMLP pre-processor ignored level-88 condition names defined in the dictionary. With Release 16.0 you can specify a parameter on the EXEC card or in an EDBPPARM module that directs the DMLP processor to generate a named constant using the VALUE attribute for a level-88 condition name.

8.8.1 Syntax



8.8.2 Parameters

YES
Directs the DMLP processor to generate a declare statement for each level-88 condition name defined to the record specified in the INCLUDE IDMS statement.

NO
Directs the DMLP processor to ignore level-88 condition names defined to a record specified in the INCLUDE IDMS statement.

8.8.3 Notes

- A named constant can only represent a single value; therefore the DMLP pre-processor ignores any level-88 condition names that specify more than one value.
- To avoid compile errors, verify your PL/I compiler supports named constants before using this feature.

- When executing the DMLP processor against an existing program and using the new EXPAND88 parameter, the potential exists for a generated named constant to be a duplicate of an existing field in the PL/I program, resulting in compile errors. Modifications must be made to remove/rename the duplicate field or remove the EXPAND88 parameter from the program.

8.9 Support for 31-Digit Zoned and Packed Decimal Elements

ELEMENT statements are used to define group or elementary data elements in IDD. Also known as fields and data items, elements can participate in records. Currently, the IDD supports COBOL picture options that allow for a maximum length of 18 digits for zoned decimal and packed decimal elements, or COMP-3, fields. Newer compilers, such as Enterprise COBOL for z/OS and OS390 and Enterprise PL/I for z/OS and OS/390, allow a maximum length of 31 digits for these fields. In r16 SP1, IDD ELEMENT statements are enhanced to support a maximum length of 31 digits for elements with a usage clause of packed and zoned decimal. All Advantage CA-IDMS tools that access records in the dictionary can access these elements.

Note: Caution should be used when exploiting the new 31-digit zoned and packed decimal feature. The external picture clause in mapping provides a maximum of 32 bytes for displaying and formatting data.

Chapter 9. TCP/IP API Support

9.1 Using TCP/IP with Advantage CA-IDMS

TCP/IP is an industry standard communications protocol. In order to understand this section, you should be familiar with the terminology and base concepts of TCP/IP. Tutorials on TCP/IP can be found on the Internet by doing a search on a general search web site with keywords "TCP/IP" and "tutorial".

Release 16.0 of Advantage CA-IDMS can exploit TCP/IP in the following ways:

- An online application can use the TCP/IP socket program interface to communicate with another TCP/IP application, possibly on another platform.
- Remote applications can directly access a central version and start an online task.

A "communication" consists of two socket programs exchanging messages. The program that initiates a service request is the **client**. The program receiving incoming requests is the **server**.

Typically, the client communicates with one server at a time. However, a server processes requests from multiple clients. The server type depends on how the client requests are processed:

- Iterative server — accepts a single client request, processes it and returns the result to the client and waits for the next client request.
- Concurrent server — accepts a client requests and spawns a "child" task to process it.

Advantage CA-IDMS TCP/IP functionality is available for these operating systems:

- z/OS and OS/390
- VSE/ESA
- z/VM

9.1.1 VSE/ESA Systems

There are limitations associated with the VSE/ESA implementation of TCP/IP. These include:

- Domains — only AF_INET is supported
- Protocol — only TCP is supported
- Sockets — only streaming sockets are supported

9.2 Generic Listener Service

9.2.1 Introduction

The generic listener service facilitates the implementation of concurrent servers quickly and easily.

9.2.2 Functionality

Generic listening performs these tasks:

- Creates a stream socket on a given port, optionally on a specific TCP/IP stack.
- Listens on the socket.
- Accepts connection requests, acquires a PTERM/LTERM pair and attaches a server task on it. This continues until the service is stopped.
- Waits for input on the socket if a server task ends normally without closing its socket. This allows implementation of suspend/resume processing, which is useful when a client application wants to keep the connection alive without tying up an Advantage CA-IDMS/DC task. Whenever the client application is ready to proceed, it sends another message over the connection. When the generic listener service receives this message it attaches a new server task on the same PTERM/LTERM pair. The task code that is invoked on a resume can be specified in the prior task by using the NEXT TASK clause of the DC RETURN statement. If the next task code is not set, the task code specified in the listener PTERM definition is invoked.

9.2.3 Implementation

Generic listening is a service provided by the SOCKET line driver. The parameters that control the listener service are defined in:

- A listener PTERM: it defines the port on which to listen, the backlog, the task code to invoke when a connection is established and the mode in which to invoke the task. Optionally, if running on a multi-homed host, the TCP/IP stack can be selected. Also optionally, a character string can be defined to pass to the attached task.
- A task and associated program definition.

  Note: The task and program should be defined to the security system so that anyone can execute them.

The program associated with the server task receives control with a parameter list containing:

- The address of an 80-byte character string set to the value of the string specified in the listener PTERM definition or blanks if none was specified.

- The address of the socket descriptor.
- The address of a 4-byte field named the resume counter. The resume counter is provided for suspend/resume processing.

Notes:

- If the listener program is written in Advantage CA-ADS the parameters are passed in the SOCKET-LISTENER-PARMS record. This record must be included as work record in the dialog definition.
- If MODE IS SYSTEM is specified in the LISTENER PTERM definition, the listener program must be written using DC/UCF calling convention conventions as described in *Advantage CA-IDMS System Operations*.

The program associated with the server task responds to the message sent from the client application. In addition to performing the required business function, it is also responsible for the following services:

- Security — When the program receives control, no user has been signed on to the system. For security purposes, the executing program must immediately signon to the system. To provide signon capabilities you must link to the RHDCSNON program. Or, for Assembler programs, you can code a #SECSGON macro.

  For more information:

  – About linking to RHDCSNON, refer to *Advantage CA-IDMS System Tasks and Operator Commands*

  – About #SECSGON, refer to *Advantage CA-IDMS Security Administration*

- Character conversion — If the remote host sends text messages in a character set other than the one used on the central version, these text messages might need translation. The program is responsible for performing this translation and IDMSIN01 functions are provided to assist in this process.

- Closing the socket — Once the conversation is over, the socket should be closed. Closing the socket causes a sign off when the task terminates. If the task ends normally without closing the socket, generic listening starts a "receive" on the socket because it interprets this situation as a suspend. As a result, the LTERM/PTERM pair remains in use and long-term resources, such as the signon element, remain allocated. These resources are subject to Advantage CA-IDMS time-out processing and can be deleted with the DCMT VARY LTERM ... RESOURCE DELETE command.

  Note: If the task abends, Advantage CA-IDMS closes the socket and the PTERM/LTERM pair is signed off automatically.

9.3 TCP/IP Considerations

This section describes how to setup and manage TCP/IP support within Advantage CA-IDMS. TCP/IP is supported in the following environments:

- z/OS and OS/390
- VSE/ESA
- z/VM

For information on how to use the TCP/IP API and the generic listener service, refer to the *Advantage CA-IDMS Callable Services*.

9.3.1 Establishing TCP/IP Support

Setting up TCP/IP support within Advantage CA-IDMS requires these steps:

1. **Updating the System Startup JCL** For DNS functions to operate correctly, a SYSTCPD card must be added to the central version JCL.

   - **z/OS and OS/390**

     This card is needed if either of the following is true:

     – The TCP/IP run time is from IBM and the TCP/IP installation specified a prefix different from the default ("TCPIP") prefix:

     ```
     //SYSTCPD  DD  DISP=SHR,DSN=prefix.TCPIP.DATA
     ```

     where *prefix* should be replaced by the TCP/IP installation prefix.

     – The TCP/IP implementation is Unicenter TCPaccess Communications Server: refer to the "*Customization Guide*".

     Contact your systems programmer to obtain this information. See the documentation from the TCP/IP vendor for information about the SYSTCPD content.

   - **VSE/ESA**

     The Advantage CA-IDMS TCP/IP implementation supports TCP/IP stacks from Connectivity Systems Incorporated (CSI) and Barnard Software Incorporated (BSI). For details regarding the required JCL for their respective BSD/C API support, refer to appropriate documentation.

     Regardless of the TCP/IP stack implementation, if the default stack id *is not* '00', the following JCL statement is required:

     ```
     // OPTION SYSPARM='nn'                    Set stack ID
     ```

     DNS socket function calls are supported in the following implementations:

     – Advantage CA-IDMS

     – BSI

     – CSI

Comparing Advantage CA-IDMS and BSI/CSI DNS socket function calls

An advantage of using the BSI/CSI DNS implementation is the GETHOSTBYNAME and GETHOSTBYADDR functions return names that are defined to the local TCP/IP stack. However, only the primary name for an IP address or primary IP address associated with the host name is returned.

An advantage of using the Advantage CA-IDMS DNS resolver is its ability to return all alias names and IP addresses that are defined to the host name or host IP address.

The BSI and CSI implementations support DNS socket function calls. but if the following optional SYSTCPD file is coded, Advantage CA-IDMS uses its internal DNS resolver instead of calling BSI or CSI.

```
// DLBL SYSTCPD,'tcpip.tcpip.data',,SD
// EXTENT SYS001,vvvvvv
// ASSGN SYS001,DISK,VOL=vvvvvv,SHR
```

where *tcpip.tcpip.data* is the file-id of the file.

For more information about the SYSTCPD layout, refer to 9.3.3, “Supporting DNS Functions Using the SYSTCPD File” on page 9-8.

- **z/VM**

  This SYSTCPD must be added to the startup EXEC:

  ```
  FILEDEF SYSTCPD DISK fn ft fm
  ```

  where the *fn ft fm* is the FILE ID of SYSTCPD file. The default value is TCPIP DATA.

  See 9.3.3, “Supporting DNS Functions Using the SYSTCPD File” on page 9-8 for more information.

2. **Modifying the Sysgen**

   - Define the SOCKET line in sysgen — A communication line of type SOCKET must be defined in sysgen to make DC/UCF available for TCP/IP API programs.
   - To enable generic listening on the DC/UCF system:
     - For each generic listener, define a LTERM/PTERM pair and its associated task code and program.
     - Define a number of BULK PTERMs. The system generation syntax permits you to define one pair, and the REPEAT clause of the PTERM statement can be used to facilitate the definition of multiple PTERMs and LTERMs.

       To determine the number of BULK PTERM/LTERM pairs:

       — Each connection accepted by a generic listener uses one BULK PTERM/LTERM pair.

       — A server task, started by a generic listener, can exploit psuedo-conversational programming. This means that the TCP/IP

connection remains open - and the BULK PTERM/LTERM pair - without an associated active task.

▸▸ For more information about the system generation statements, refer to *Advantage CA-IDMS System Generation*

9.3.2 Managing TCP/IP Support

There are several DCMT commands that are enhanced to help you manage the TCP/IP environment:

- DCMT DISPLAY LINE — a new parameter on this statement causes TCP/IP-related information to be displayed.
- DCMT VARY LINE — allows you to dynamically enable and disable TCP/IP support.
- DCMT VARY PTERM — enables you to manage the generic listener service dynamically.

Note: Multiple lines of type SOCKET can be defined to the DC/UCF system, but only one SOCKET line can be active at a time.

▸▸ Refer to *Advantage CA-IDMS System Tasks and Operator Commands* for more information.

9.3.3 Supporting DNS Functions Using the SYSTCPD File

9.3.3.1 z/OS and OS/390

The DNS socket functions (GETHOSTBYADDR and GETHOSTBYNAME) are directly supported in the operating system dependant TCP/IP interface used by the #SOCKET macro interface. The SYSTCPD file is used internally by the operating system to process the DNS requests.

9.3.3.2 VSE/ESA

The SYSTCPD file is optional in the VSE/ESA environment. If the SYSTCPD file is coded, the DNS socket functions are resolved using Advantage CA-IDMS's DNS resolver. For more information about using Advantage CA-IDMS DNS resolver, see 9.3.3.4, “Advantage CA-IDMS DNS Resolver” on page 9-9.

9.3.3.3 z/VM

The DNS socket functions are not supported by the operating system interface. Therefore, an internal DNS resolver is implemented in Advantage CA-IDMS. The resolver communicates with a name server to retrieve the requested DNS information.

Note: Local table lookup is not provided.

9.3.3.4 Advantage CA-IDMS DNS Resolver

The following information applies to z/VM and optionally VSE/ESA.

During the open of the socket line:

1. The file associated with the SYSTCPD card is read.
2. The DNS specific parameters are parsed.
3. The Advantage CA-IDMS resolver is configured.

Coding the SYSTCPD file: The following syntax rules apply for this file:

- All records starting with a ';' character are treated as comments.
- Blanks and <end-of-line> characters delimit the tokens.
- The format for each configuration statement is : Keyword Value.

The following table lists the SYSTCPD parameters:

Keyword	Default Value	Range of values	Meaning
DOMAINORIGIN		Max 64 chars	Suffix that is appended to a hostname that doesn't contain any dots
NSINTERADDR			Up to 4 different NSINTERADDR input lines pointing to different DNS servers
NSPORTADDR	53	1 to 65535	Port of the name server
RESOLVEVIA	UDP	UDP or TCP	Protocol to use for the communication. Only TCP is supported on VSE/ESA.
RESOLVERTIMEOUT	30	1 to 65535	Time in seconds that the resolver waits for a response from the server
RESOLVERUDPRETRIES		1 to 65535	Number of times the resolver tries to communicate with the name server (when RESOLVEVIA is UDP only)

Keyword	Default Value	Range of values	Meaning
TCPIPUSERID	TCPIP	Max 8 chars	Userid of the default TCP/IP virtual machine. 1

Note: 1 — This entry defines the userid of the TCP/IP virtual machine that is used as the default TCP/IP stack by Advantage CA-IDMS. This is only applicable for z/VM.

Sample SYSTCPD

```
DOMAINORIGIN  CA.COM
NSINTERADDR  172.24.255.255
NSPORTADDR 53
RESOLVEVIA TCP
RESOLVERTIMEOUT 3
RESOLVERUDPRETRIES 1
```

VSE/ESA SYSTCPD File: The SYSTCPD file:

- Must contain 80-byte records
- Can be blocked up to a block size to 32720

Warning: Do not code the BLKSIZE= DLBL parameter.

z/VM SYSTCPD File: The SYSTCPD file:

- Can be fixed or variable length
- Can use the default file, TCPIP DATA *, or the user can create his own file with specific definitions
- Contains the userid of the TCP/IP virtual machine that is used as the default TCP/IP stack by the Advantage CA-IDMS system
- Can specify up to 8 additional TCP/IP stacks that can be defined to a Advantage CA-IDMS system. These TCP/IP stacks can be used concurrently by programs running under the DC/UCF system. For more information about the TCP/IP_STACK_1 to TCP/IP_S TACK_8 SYSIDMS parameters, refer to *Advantage CA-IDMS Callable Services*.

9.3.4 Link RHDCT1IP module (VSE/ESA Only)

A DOST1IP.OBJ module is delivered with the product. It must be linked with object modules delivered by CSI or BSI as phase RHDCT1IP.

Linking RHDCT1IP for CSI: When using TCP/IP from CSI, the following INCLUDEs are required to link the RHDCT1IP module. IPNRxxxx modules should get autolinked.

```
// LIBDEF *,SEARCH=(tcpiplib.sublib,idmslib.sublib)
// LIBDEF PHASE,CATALOG=idmslib.sublib
// OPTION CATAL
PHASE   RHDCT1IP,*
INCLUDE DOST1IP
ENTRY   T1IPEP1
/*
// EXEC LNKEDT
/*
```

Linking RHDCT1IP for BSI: When using TCP/IP from BSI, the following INCLUDEs are required to link the RHDCT1IP module. The IPNRxxxx entry points should get resolved in BSI module BSTTENVR.

```
// LIBDEF *,SEARCH=(tcpiplib.sublib,idmslib.sublib)
// LIBDEF PHASE,CATALOG=idmslib.sublib
// OPTION CATAL
PHASE   RHDCT1IP,*
INCLUDE DOST1IP
INCLUDE BSTTENVR
ENTRY   T1IPEP1
/*
// EXEC LNKEDT
/*
```

9.3.4.1 Parameters

tcpiplib.sublib	name of the sublibrary within the library containing TCP/IP modules
idmslib.sublib	name of the sublibrary within the library containing Advantage CA-IDMS modules

9.4 TCP/IP Programming Support for Online Applications

TCP/IP programming support within Advantage CA-IDMS allows an application to communicate through TCP/IP protocols with a second application. The second application can reside on the same platform or another platform.

The socket program interface depends upon the programming language used to write the application:

- Programs written in Assembler use the #SOCKET macro interface.
- Programs written in COBOL or PL/I use a call interface to IDMSOCKI.
- Applications written in Advantage CA-ADS can use the SOCKET built-in function or the call interface to IDMSOCKI.

9.4.1 Socket Macro Interface For Assembler Programs

Programs written in the assembler language use the #SOCKET macro to exploit TCP/IP sockets. The #SOCKET macro takes this general form:

```
label     #SOCKET function,                                    X
                RETCODE=return-code,                           X
                ERRNO=errno,                                   X
                RSNCODE=reason-code,                           X
                PLIST=parameter-list-area,                     X
                RGSV=(rgsv),                                   X
                CALL=call-value,                               X
                function-specific-parameters
```

9.4.1.1 Parameters

label
Optional assembler label.

function
The name of the function to execute. See Appendix G, “TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion,” for a list of valid functions..

return-code
The name of a fullword that receives the outcome of the operation. Possible values are:

0
No error occurred

-1
A socket error was encountered; the *errno* and *reason-code* fields contain more detailed information about the error.

errno
The name of a fullword that receives the ERRNO value when return-code is -1. See Appendix G, “TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion,” for more information.

reason-code
The name of a fullword that receives the reason code value when *return-code* is -1. See Appendix G, “TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion,” for more information.

parameter-list-area
Name of an area or register pointing to the area that is used to build the #SOCKET parameter list. The default is SYSPLIST. The length of the *parameter-list-area* used by the macro depends on the #SOCKET function that is called; the longest parameter-list currently needed for a #SOCKET call is 16 fullwords.

rgsv
Registers to be saved. This parameter applies only to system mode programs. The default is (R2-R8).

call-value
Indicates whether to generate the parameter list and/or execute the function. Possible values are

YES
Generate the parameter list and execute the function. This is the default.

NO
Generate the parameter list, but don't execute the function.

ONLY
Execute the function for which a parameter list is pre-built.

9.4.1.2 Notes

- The syntax doesn't show Assembler column conventions (label starts in column 1; statement in column 10; continuation line in column 16; continuation character in column 72).
- On return from the #SOCKET call, R15 is always 0, except in cases of a parameter-list error where the RETCODE field cannot be found; in this case R15 is set to -1.
- The parameter values assigned to the three return code parameters (RETCODE, ERRNO and RSNCODE) and to all the function-specific-parameters can be specified in data field notation or in register notation.

  In data field notation, the program specifies the name of a variable field containing the parameter value.

  In register notation, the program specifies a register containing the address of the variable field containing the parameter value (not the value itself). General

registers 2 to 15 can be used in this notation; the register reference must be enclosed in parentheses.

- Some parameters also accept a value in the form of an absolute expression. Where applicable, this is mentioned under the corresponding parameter's description.
- Some parameters from the #SOCKET macro are optional. There are two ways to address an optional parameter:
  - Omit the parameter on the #SOCKET macro call.
  - Assign a null value to the parameter. For example, HOSTNAME=NULL.

  Both ways are equivalent.
- The #SOCKET macro uses the following registers when building its parameter list:

  R0 — A work register to build the parameter list
  - R1 — Address the parameter list
  - R14 and R15 — The branch and link registers for the call sequence to socket services
- #SOCKET TCPIPDEF generates DSECTs and EQUates needed to write a TCP/IP program.
- #SOCKET ERRNOS generates all EQUates for Advantage CA-IDMS specific errno values.

9.4.2 The Advantage CA-ADS Socket Interface

Applications written in Advantage CA-ADS can use one of two methods to exploit TCP/IP sockets:

- Using an Advantage CA-ADS system-supplied built-in function, SOCKET. It follows the same general rules as other Advantage CA-ADS built-in functions. The following is an example of the code required to invoke the SOCKET built-in function in your Advantage CA-ADS dialog:

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

  Parameters can be records or record elements.
- Using an Advantage CA-ADS control statement to invoke the socket call interface, IDMSOCKI. IDMSOCKI is the same socket call interface that can be used with COBOL programs. In this scenario, the LINK control statement is used to invoke IDMSOCKI:

```
LINK TO PROGRAM 'IDMSOCKI' USING
        (function,
       return-code,
       errno,
       reason-code,
       function-dependent-parameter1,
       ...)
```

Each parameter must be a separate record.

9.4.2.1 Parameters

For both methods, the first four parameters are identical except that if linking to IDMSOCKI, each parameter must be defined as a record whose first element is a field described below. If using the SOCKET built-in function the parameters can be records or record elements.

function

A 4-byte, full-word aligned, integer field that the program sets to the desired socket function. A detailed description of the supported functions can be found in Appendix G, "TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion."

return-code

A 4-byte, full-word aligned, integer field that 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.

errno

A 4-byte, full-word aligned, integer field that receives the ERRNO value when *return-code* is -1. Refer to Appendix G, "TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion," for more information.

reason-code

A 4-byte, full-word aligned, integer field that receives the reason code value when *return-code* is -1. Refer to Appendix G, "TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion,"for more information.

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

9.4.2.2 Comparing IDMSOCKI and SOCKET

While either of these methods allows you to utilize the TCP/IP API functionality, there are benefits to using the SOCKET built-in function:

- Parameters can be a record element. When IDMSOCKI is used, each parameter must be defined as a separate record.
- It is easier to use.
- It provides optimum performance. Calling a system-defined built-in function is more efficient than LINKing to another program type.
- It is possible to use the system-defined record SOCKET-CALL-INTERFACE, which contains the definition of the first four parameters. To use this record, add it to the dialog as a work record.
- SOCKET supports omitted parameters.

Because of these advantages, use of the SOCKET built-in function is recommended.

9.4.2.3 Notes

- To omit an optional parameter in the parameter list, replace the parameter with the @ symbol.
- An Advantage CA-ADS dialog associated with a server task (a task started by a generic listener):
  - Must be a mapless dialog
  - Should include SOCKET-LISTENER-PARMS as a work record
- The following pre-defined records are provided during installation and can be attached to a dialog as work records:
  - SOCKET-CALL-INTERFACE — describes the socket functions, return codes and errno values used to issue all socket requests
  - SOCKET-MISC-DEFINITIONS — describes options and flags specific to individual functions
  - SOCKET-SOCKADDR-IN, SOCKET-SOCKADDR-IN6, SOCKET-HOSTENT, SOCKET-TIMEVAL and SOCKET-ADDRINFO — describe structures that may be useful for certain socket applications
  - The SOCKET-CALL-INTERFACE record contains fields that can be used for SOCKET built-in function common parameters:
    - *function*
    - *return-code*
    - *errno*
    - *reason-code*

    Each supported function is represented by a field, whose value is the function number. The following example illustrates how to issue a READ socket

request using the SOCKET built-in function and fields within the SOCKET-CALL-INTERFACE record:

```
IF (SOCKET (SOCKET-FUNCTION-READ,
            SOCKET-RETCD,
            SOCKET-ERRNO,
            SOCKET-RESNCD, . . .) = 0)
```

– For more information about Advantage CA-ADS and built-in functions, refer to the *Advantage CA-ADS for CA-IDMS Reference Guide*.

9.4.3 Socket Call Interface For COBOL

Programs written in COBOL use the CALL statement to exploit TCP/IP sockets:

```
CALL 'IDMSOCKI' USING
         function,
         return-code,
         errno,
         reason-code,
         function-dependent-parameter1,
         . . .
```

9.4.3.1 Parameters

A call to IDMSOCKI must pass the following first four parameters:

function

A 4-byte, full-word aligned, integer field that the program sets to the desired socket function. Sample definition of a *function* field:

```
01  SOCKET-FUNCTION   PIC S9(8) COMP.
```

A detailed description of the supported functions can be found in Appendix G, "TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion."

return-code

A 4-byte, full-word aligned, integer field that 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.

Sample definition of a *return-code* field:

```
01  SOCKET-RETCD   PIC S9(8) COMP.
```

errno

A 4-byte, full-word aligned, integer field that receives the ERRNO value when *return-code* is -1. Sample definition of an *errno* field:

```
01  SOCKET-ERRNO   PIC S9(8) COMP.
```

Refer to Appendix G, “TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion,” for more information.

reason-code

A 4-byte, full-word aligned, integer field that receives the reason code value when *return-code* is -1. Example definition of a *reason-code* field:

```
01  SOCKET-RSNCD   PIC S9(8) COMP.
```

Refer to Appendix G, “TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion,”for more information.

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

9.4.3.2 Notes

- If an optional parameter is not specified in the parameter list, it should be replaced by a parameter that depends on the COBOL compiler:
  - For COBOL for z/OS and OS/390, specify reserved keyword OMITTED.
  - For ANSI COBOL85, specify BY VALUE dummy-variable; dummy-variable should be set to 0.
- The following pre-defined records are provided during installation to assist in writing socket applications:
- SOCKET-CALL-INTERFACE — describes the socket functions, return codes and errno values used to issue all socket requests
- SOCKET-MISC-DEFINITIONS — describes options and flags specific to individual functions
- SOCKET-SOCKADDR-IN, SOCKET-SOCKADDR-IN6, SOCKET-HOSTENT, SOCKET-TIMEVAL and SOCKET-ADDRINFO — describe structures that may be useful for certain socket applications
- The SOCKET-CALL-INTERFACE record contains fields that can be used for the socket call common parameters:
  - *function*
  - *return-code*
  - *errno*
  - *reason-code*

  Each supported function is represented by a field, whose value is the function number. The following example illustrates how to issue a READ socket request using the fields within the SOCKET-CALL-INTERFACE record:

```
CALL 'IDMSOCKI' USING SOCKET-FUNCTION-READ,
                      SOCKET-RETCD,
                      SOCKET-ERRNO,
                      SOCKET-RESNCD,
                      . . .
```

Note: The SOCKET-CALL-INT record is identical to the SOCKET-CALL-INTERFACE record except that function values are defined as condition names instead of fields. Unless storage is critical, the SOCKET-CALL-INTERFACE record should be used.

- The program associated with a server task (a task started by a generic listener) must specify:
  - In the LINKAGE SECTION:

```
01  SOCKET-PARMS        PIC X(80).
01  SOCKET-DESCRIPTOR   PIC S9(8) COMP.
01  SOCKET-RESUME-COUNT PIC S9(8) COMP.
```

  - In the PROCEDURE DIVISION:

```
PROCEDURE DIVISION USING
          SOCKET-PARMS,
          SOCKET-DESCRIPTOR,
          SOCKET-RESUME-COUNT.
```

9.4.4 Socket call interface for PL/I

Programs written in PL/I use the CALL statement to exploit TCP/IP sockets:

```
CALL 'IDMSOCKI'
       (function,
       return_code,
       errno,
       reason_code,
       function_dependent_parameter1,
       . . . );
```

9.4.4.1 Parameters

A call to IDMSOCKI must pass the following first four parameters:

function

A 4-byte, full-word aligned, integer field that the program sets to the desired socket function. Sample definition of a *function* field:

```
DCL SOCKET_FUNCTION   FIXED BINARY(31);
```

A detailed description of the supported functions can be found in Appendix G, “TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion.”

return_code

A 4-byte, full-word aligned, integer field that 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.

Sample definition of a **return_code** field:

```
DCL SOCKET_RETCD   FIXED BINARY(31);
```

Value definitions for return codes can be found in Appendix G, "TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion."

errno

A 4-byte, full-word aligned, integer field that receives the ERRNO value when *return_code* is -1. Sample definition of an *errno* field:

```
DCL SOCKET_ERRNO   FIXED BINARY(31);
```

Refer to Appendix G, "TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion," for more information.

reason_code

A 4-byte, full-word aligned, integer field that receives the reason code value when *return_code* is -1. Sample definition of a *reason_code* field:

```
DCL SOCKET_RSNCD   FIXED BINARY(31);
```

Refer to Appendix G, "TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion," for more information.

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

9.4.4.2 Notes

- Some PL/I compilers limit the length of an external name to 7 characters. Since IDMSOCKI contains 8 characters, this can lead to errors at compile time. Such errors can be solved in these ways:
  - Use the compile option LIMITS(EXTNAME(8))
  - Use entry point IDMSOCK, which is defined as a synonym to IDMSOCKI.
- If an optional parameter is not to be specified in the parameter list, replace it by an asterisk (*).
- The following pre-defined records are provided during installation to assist in writing socket applications:
  - SOCKET_CALL_INTERFACE — describes the socket functions, return codes and errno values used to issue all socket requests.
  - SOCKET_MISC_DEFINITIONS — describes options and flags specific to individual functions.
  - SOCKET_SOCKADDR_IN, SOCKET_SOCKADDR_IN6, SOCKET_HOSTENT, SOCKET_TIMEVAL and SOCKET_ADDRINFO — describe structures that may be useful for certain socket applications.

Note: Some of these records contain condition names. To generate the appropriate declare statements, specify the following pre-compiler option:

```
EXPAND88=YES
```

- The SOCKET_CALL_INTERFACE record contains fields that can be used for socket call common parameters:
  - *function*
  - *return_code*
  - *errno*
  - *reason_code*

  Each supported function is represented by a field whose value is the function number. The following example illustrates how to issue a READ socket request using the fields within the SOCKET_CALL_INTERFACE record:

```
CALL 'IDMSOCKI' USING (SOCKET_FUNCTION_READ,
                       SOCKET_RETCD,
                       SOCKET_ERRNO,
                       SOCKET_RESNCD,
                       . . .);
```

  Note: The SOCKET_CALL_INT record is identical to the SOCKET_CALL_INTERFACE record except that function values are defined as condition names instead of fields. Unless storage is critical, the SOCKET_CALL_INTERFACE record should be used.

- The program associated with a server task (a task started by a generic listener) must specify:

```
PROCEDURE (P1, P2, P3)
  OPTIONS (REENTRANT,FETCHABLE);

DCL (P1,P2,P3)          POINTER;
DCL SOCKET_PARMS        CHAR(80)          BASED (ADDR(P1));
DCL SOCKET_DESCRIPTOR   FIXED BINARY(31)  BASED (ADDR(P2));
DCL SOCKET_RESUME_COUNT FIXED BINARY(31)  BASED (ADDR(P3));
```

9.4.5 Application Design Considerations

The TCP/IP socket program interface is available only to Advantage CA-IDMS/DC applications running under a central version. A batch program trying to use the interface receives a socket return code of RNOSLIND.

Server tasks started by a generic listener cannot do any terminal I/O such as #LINEIN, #LINEOUT, #TREQ etc. If written in Advantage CA-ADS, they should be mapless dialogs.

9.4.5.1 Using Stream Sockets

TCP allows for arbitrary amounts of data to be sent and received over a stream socket. Since a stream is interpreted as a sequence of bits, TCP has no knowledge of the organization, content or amount of data being processed. Therefore, a TCP application should use its own protocol to logically divide a stream into messages. The most common way of doing this is to prefix the data with the data length.

9.4.5.2 Receiving Data

TCP may choose to break a block of data into pieces and transmit each piece separately or it may accumulate data in its buffer and send it in one block. Thus, the data sent in a single "send" can be received in a single "receive", or it can be received in small chunks. It is possible the receiving application may get the data of multiple send requests in a single receive. TCP receives data until the expected message is completely received.

9.4.5.3 Sending Data

As with receiving data, there is no guarantee that a send request is completely serviced. TCP may decide that the amount of data in the send request is too large. If so, it returns the amount of data already processed and the application must re-issue the send with an updated data length and buffer pointer. TCP sends data until the message is completely sent.

9.4.6 TCP/IP Coding Samples

The Advantage CA-IDMS installation media contains these sample programs, which are intended for demonstration purposes only:

- TCPASM01 — An Assembler program that tests the #SOCKET API calls. TCPASM01 can be invoked in one of two ways:
  - As a user task code at the "ENTER NEXT TASK CODE" prompt. Depending on the command line parameters, a client or server program is initiated. The program converses with a partner program using SEND and RECV calls. If no parameters are specified, a HELP screen containing the full syntax and its options is displayed.
  - As a server program started by a listener PTE.

    Note: The listener's PTERM definition should specify MODE is USER.
- TCPADS01 — A TCP/IP client program written in Advantage CA-ADS.
- TCPCOB01 — A TCP/IP generic listener server program written in COBOL.
- TCPPLI01 — A TCP/IP generic listener server program written in PL/I.

TCPADS01 and TCPASM01 (as a client) provide the same functionality: they connect to a port number that matches a port number of a generic listener PTERM.

TCPPLI01, TCPCOB01, and TCPASM01 (as a server) can be invoked by the task code associated with the generic listener PTERM.

Note: The header section of each sample program contains compiler option information that is required to successfully compile the program.

Sample TCP/IP programs can also be found in *Advantage CA-IDMS Callable Services*.

9.5 Miscellaneous TCP/IP Considerations

The following sections provide information related to TCP/IP use, including:

- The TCP/IP trace facility
- Multiple TCP/IP stacks
- Assigning a timeout value to each TCP/IP socket

9.5.1 Using the TCP/IP Trace Facility

To help debug socket programs, a TCP/IP trace facility is available. It is activated using the DCMT VARY LTERM command. For more information about this command, refer to the *Advantage CA-IDMS System Tasks and Operator Commands*.

9.5.2 Using Multiple TCP/IP Stacks

In a multiple TCP/IP stack environment, an Advantage CA-IDMS system is able to use several available TCP/IP stacks concurrently. Only z/OS and z/VM support multiple TCP/IP stacks.

In the z/OS environment, the Common INET (CINET) configuration is required to run multiple TCP/IP stacks concurrently. Advantage CA-IDMS uses special system calls to get a list of the available TCP/IP stacks in the system. For more information about the CINET feature, refer to the *z/OS Communication Server IP Configuration Guide*

For z/VM, multiple TCP/IP stacks are implemented by starting each stack in its own virtual machine. The additional TCP/IP stacks are defined in the SYSIDMS parameters TCP/IP_STACK_1 to TCP/IP_STACK_8. For more information:

- About the SYSTCPD file, refer to *Advantage CA-IDMS System Operations*.
- About the SYSIDMS parameters, refer to *Advantage CA-IDMS Common Facilities*.

Default TCP/IP stack: On z/OS and z/VM, the default TCP/IP stack for an Advantage CA-IDMS system is displayed by using DCMT DISPLAY LINE <tcpip> IPINFO command.

Current TCP/IP stack for a DC task: When a DC task is started, the current TCP/IP stack for the DC task is the default TCP/IP stack from the Advantage CA-IDMS system. The SETSTACK function can be used to assign another value to the current TCP/IP stack or to restore the default value for the DC task.

Stack affinity: This concept refers to sockets. When a socket is created and it is exclusively attached to a specific TCP/IP stack, it is said to have "stack affinity". The stack affinity is equal to the value of the current TCP/IP stack when the socket was created. A socket that is not attached to a specific TCP/IP stack has no stack affinity.

Default stack affinity: When a socket is created in the default DC task environment, that is, no specific SETSTACK calls have been issued in the task yet, the default stack affinity is the default TCP/IP stack.

When a socket has set stack affinity to *ALL and the application issues an ACCEPT socket call with the IP address equal to INADDR_ANY in the corresponding socket address structure, then the ACCEPT request is propagated to all available TCP/IP stacks, and therefore the application can accept connections from clients specifying an IP address from any of the available TCP/IP stacks. This configuration is possible on z/OS only. If the ACCEPT'ing socket is assigned a specific stack affinity, the client must specify the IP address corresponding to that specific stack.

Two socket functions return values that are influenced by which TCP/IP stack is current on the DC task.

- GETHOSTID — returns the IP address of the current TCP/IP stack.
- GETHOSTNAME — returns the hostname of the current TCP/IP stack.

The output from the DCMT DISPLAY LINE <tcpip-line-name> IPINFO command shows the available TCP/IP stacks with their associated IP address and hostname. For more information about DCMT DISPLAY LINE, refer to *Advantage CA-IDMS System Tasks and Operator Commands*.

9.5.3 Associating Timeouts to Sockets

In the standard POSIX socket interface, timeout conditions can only be detected through the use of the SELECT socket function. The socket interface provided by Advantage CA-IDMS offers an extension that assigns a timeout value to each socket created in the DC/UCF environment. The FCNTL socket function enables you to specify or retrieve a socket's timeout value.

When a socket is created, a default timeout value is assigned. The default timeout value depends on the type of socket:

- For a socket created by the SOCKET function, or "**client socket**", the default timeout value is set to the corresponding DC task's INACTIVE INTERVAL parameter.
- For a socket created by the ACCEPT function, or "**server socket**", the default timeout value is set to the corresponding DC task's EXTERNAL WAIT parameter.

These socket functions:

- ACCEPT
- CONNECT
- READ
- RECV
- RECVFROM

- SEND
- SENDTO
- WRITE

check the timeout value at runtime. When a timeout condition occurs, the socket function returns a ETIMEDOUT ERRNO code to the application.

For more information:

- About the FCNTL socket function, refer to G.2.5, “FCNTL” on page G-10.
- About the INACTIVE INTERVAL or EXTERNAL WAIT parameters of the TASK statement, refer to the *Advantage CA-IDMS System Generation*.

Chapter 10. Type 4 JDBC Driver

10.1 Overview

IDMSJSRV is the built-in TCP/IP server program that enables the JDBC driver supplied with Advantage CA-IDMS Server 5.0 or later to function as a "Type 4" driver. This allows client applications written in Java to communicate directly with the Advantage CA-IDMS address space using the native "wire" protocol, with no intervening middleware.

10.2 Installing the Java Runtime Environment

The Type 4 JDBC driver performs all data conversion on the client platform, using the character converter classes provided by the Java Runtime Environment (JRE). The JRE includes converter classes for most of the character encodings in use around the world. However, by default, the JRE installer installs only European language support on machines that support only European languages. The mainframe encodings based on EBCDIC, such as CP037, are not included.

When installing the JRE, select the "Custom" option and then select "Support for Additional Languages". This installs the complete set of character encodings, contained in the file charsets.jar.

This is generally not necessary when using the JRE included with the Java Software Development Kit (SDK), which includes charsets.jar.

10.3 Enabling the Type 4 JDBC Driver

To enable the Type 4 JDBC driver, define a TCP/IP line and a listener PTERM on the mainframe and specify the host name and port on the client. The *Advantage CA-IDMS System Generation* guide contains detailed information on defining the TCP/IP line and listener. The *Advantage CA-IDMS Server 5.0 User Guide* contains detailed information on how to specify client settings. A Java application can use either an IdmsDataSource object or the static DriverManager class to get a connection.

The Type 4 JDBC Driver supports a large number of concurrent connections and is essentially limited only by the resources allocated to your Advantage CA-IDMS system. In particular, you may need to increase the size of storage pools to support high volume applications that create many concurrent connections. Advantage CA-IDMS uses a minimum of approximately 250 KB of storage for each dynamic SQL session, whether started by the Type 4 driver, the Type 2 or ODBC drivers, OCF, or IDMSBCF.

10.3.1 Listener PTERM Options

To use the Advantage CA-IDMS Server "wire protocol" drivers, define a listener PTERM/LTERM pair for the built-in server program, IDMSJSRV. This PTERM must specify the RHDCNP3J task defined during installation, SYSTEM mode, and the port used by the driver. The default port, 3709, is used by the drivers and registered with the Internet Assigned Number Authority (IANA) for Advantage CA-IDMS. This can be used if only a single DC/UCF system is used on the host machine. Otherwise, a recommended convention is to append the system number to 37.

Additional listener options parameters can be specified as keyword-values pairs in the PARM string on the PTERM definition. The following are the PARM keyword guidelines:

- Options are not case-sensitive.
- Options can appear in any order, separated by commas, with no embedded spaces in the string (leading spaces are ignored).
- If an error occurs, the server writes a message to the DC log and ignores the rest of the options.

The parameters are:

ACCt=profile-key-name
Profile key name for accounting information. If specified, any accounting information supplied by the client as part of the signon is added to the user profile. This allows SQL statements access to the accounting information using the PROFILE function. Procedures written in COBOL or Advantage CA-ADS can use the IDMSIN01 GETPROF callable service to access the accounting information.

BUFLen=default-length
Length of the default buffer used for data received from and sent to the client. The default buffer is allocated when the task starts and freed when the task ends.

Larger buffers are allocated dynamically as needed for individual requests and freed immediately after results are returned. The default is 1024 bytes. Specifying a larger buffer can minimize storage allocation requests while using more of the storage pool. The client can override this value.

BUFMax=max-length
Maximum length the client interface can specify for the default buffer.

TASK=task-code
Specifies the default task code used for signon security, timeouts, and as the next task after a pseudo converse. The IDMSJSRV task is defined during installation as a model task for this purpose. The client can override this task code.

ATTACH=Y
Causes IDMSJSRV to end the initial task immediately after processing the signon request. The specified task is attached on the next request from the client interface. If not specified (or No), IDMSJSRV copies the timeouts from specified task to the current task, which ends when (and if) explicitly requested by the client interface.

TIMeout=seconds
Specifies the maximum timeout the client can request for the external wait or resource timeout interval. A value of -1 allows the client interface to specify any value. A value of 0, the default, takes the value from the task or system default.

10.3.2 Listener TASK Security

You must grant execute authority on task RHDCNP3J to group PUBLIC or all groups. Alternatively, you can turn off security for task RHDCNP3J by including an entry in the SRTT. The task specified in the PARM string, IDMSJSRV for example, can be restricted to specific users or groups. See *Advantage CA-IDMS Security Administration* for detailed information on securing tasks.

10.3.3 IdmsDataSource Options

The following describes how to specify IdmsDataSource properties:

databaseName
Specify the DBNAME of the dictionary that contains table, view, and procedure definitions. Note that on Windows and USS, this is interpreted as the name of an ODBC style DSN in the registry or configuration file. The JDBC driver searches for this DSN and overrides the DBNAME with the Dictionary value if found.

accountInfo
Specify optional accounting information to be processed by the sign on user exit in Advantage CA-IDMS.

nodeName
Do not specify the nodeName property. The nodeName property is used by the native interfaces that communicate through CCI, and it is not compatible with the Type 4 JDBC driver. When it is set, a connection attempt fails with a DB001070 message from Advantage CA-IDMS.

networkProtocol
Specify "TCP".

serverName
Specify the DNS name or TCP/IP address of the machine where Advantage CA-IDMS/DC is running.

portNumber
Specify the port defined for the IDMSJSRV listener in the SYSGEN PTERM definition. Note that the default port for the JDBC driver is 3709, the number registered with IANA, the Internet Assigned Numbers Authority. When more than one CV runs on the same host, a recommended convention is to append the system number to 37, for example, SYST0099 would use port 3799.

taskCode
The IDMSJSRV listener task does not recognize the alternate task code passed by the Advantage CA-IDMS Server 5.0 JDBC driver. If it is necessary to use a different task for different applications, define a separate listener PTERM with a different port for each alternate task required and specify that port in the DataSource definition. The next release of the Advantage CA-IDMS Server JDBC driver will support specifying the alternate task code for the IDMSJSRV listener.

viaNodeName
This is ignored by the IDMSJSRV listener task. Equivalent routing is implemented using the Advantage CA-IDMS DBNAME and RESOURCE tables, as it is for mainframe clients.

10.3.4 DriverManager Options

Specify the DBNAME of the dictionary, the server name, and the server port in the URL. When using the DriverManager, other properties can be specified with Properties objects that correspond to the DataSource properties and follow the rules described above. These Properties objects essentially contain keyword-value pairs with the following keywords:

account
See accountInfo option described in previous section.

node
See nodeName option described in previous section.

via
Not used with the type 4 driver.

task
See taskCode option described in previous section. This option is not supported in this release.

ccihost
Not used with the type 4 driver.

cciport
Not used with the type 4 driver.

10.3.5 Additional Client Options

There are several new connection options:

- Alternate task code
- Default buffer size
- External wait and resource timeouts
- DC log and DBTRACE debugging

Default values for these options can be specified in the listener PTERM definition. The JDBC driver included in Advantage CA-IDMS Server 5.0 does not support specifying these options on the client platform. The JDBC driver included in Advantage CA-IDMS Server 16.0 will support them.

Appendix A. New and Revised DCMT Commands

A.1 Overview

This appendix describes DCMT commands that are new or that have changed in Release 16.0.

A.2 DCMT SHUTDOWN

This command has been enhanced to inhibit prompting for permission to proceed with system shutdown.

This section describes only the new parameters of this command. For more information, see *Advantage CA-IDMS System Tasks and Operator Commands*.

A.2.1 Syntax

A.2.2 Parameters

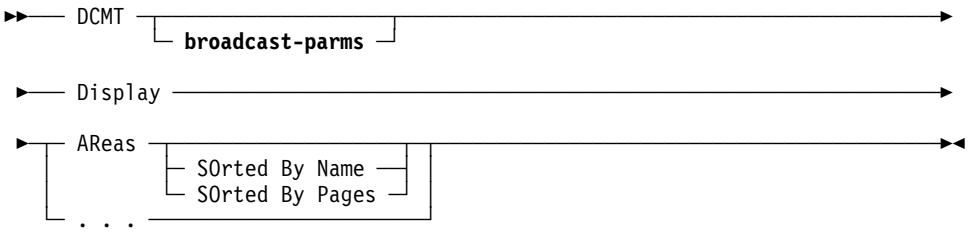
NOPrompt
Requests that the system proceed to shutdown without prompting for permission to proceed.

A.3 DCMT DISPLAY AREA

This command has been enhanced to allow sorting the display alphabetically by area name or by page group and page range.

This section describes only the new parameters of this command. For more information, see *Advantage CA-IDMS System Tasks and Operator Commands*.

A.3.1 Syntax



A.3.2 Parameters

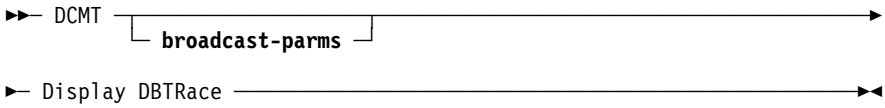
SOrted By Name
Displays the areas sorted alphabetically by area name.

SOrted By Pages
Displays the areas sorted by page group and page range.

A.4 DCMT DISPLAY DBTRACE

This new command displays information about the database trace.

A.4.1 Syntax



A.4.2 Parameters

broadcast-parms

Indicates to execute the DCMT command on all or a list of data sharing group members. Refer to A.31, “How to Broadcast System Tasks” on page A-49 for more information on broadcasting and **broadcast-parms** syntax.

A.4.3 Example

The following example illustrates the use of the DCMT DISPLAY DBTRACE command to see whether the trace is on and, if it is on, the size of the trace table:

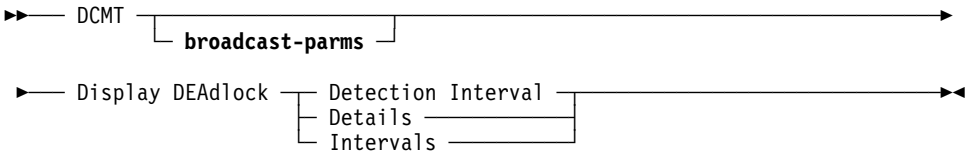
```
DCMT DISPLAY DBTRACE
DBTrace is ON size        5000
```

A.5 DCMT DISPLAY DEADLOCK

This command has been enhanced to display the deadlock details current setting.

To alter the deadlock details setting, see A.15, “DCMT VARY DEADLOCK” on page A-24.

A.5.1 Syntax



A.5.2 Parameters

Details

Displays the current ON/OFF setting for deadlock details.

A.6 DCMT DISPLAY DISTRIBUTED RESOURCE MANAGER

This new command displays information about resource managers that are known to a system.

A.6.1 Syntax

A.6.2 Parameters

broadcast-parms

Indicates to execute the DCMT command on all or a list of data sharing group members. Refer to A.31, “How to Broadcast System Tasks” on page A-49 for more information on broadcasting and **broadcast-parms** syntax.

RESource MANager

Valid values are '*rm-name*' and spaces. If '*rm-name*' is not specified, a list of all known resource managers is displayed.

rm-name

Specifies the name of the resource manager to display. The *rm-name* value must be enclosed in single quotes using the format 'xxxxxxxx::yyyyyyyy'. The *rm-name* value must match a value that appears on the summary display.

A.6.3 Examples

The example below illustrates the use of the DCMT DISPLAY DISTRIBUTED RESOURCE MANAGER command to obtain a summary of known resource managers.

```
DCMT D DIST RM

RM Name             Status      Startup time (UTC)          PndResync
  SYSTEM73::RRS_RMI   Open        N/A                                 0
  SYSTEM73::DSI_CLI   Open        2003-01-30-11.36.05.368120          0
  SYSTEM73::DSI_SRV   Open        2003-01-30-11.36.05.368120          0
  SYSTEM72::DSI_SRV   Initial     *Unknown                            1
  SYSTEM74::DSI_SRV   Open        2003-01-31-13.17.27.855555          1
```

The example below illustrates the use of the DCMT DISPLAY DISTRIBUTED RESOURCE MANAGER command to obtain detailed information about an individual resource manager and all distributed transactions in which it has an interest.

```
DCMT D DIST RM 'SYSTEM74::DSI_SRV'
RM Name             SYSTEM74::DSI_SRV
Status              Open
Startup time (UTC)  2003-01-31-13.17.27.855555
Task/LTE  |Distributed transaction ID-Branch ID       |Ctrl|State|Ind|Outcome
*none     |SYSTEM74::01650D6EDFB1AB93-01650D6A79F31E50|IDMS|InDbt|Rsy|OK
```

A.6.4 Usage

Output from DCMT DISPLAY DISTRIBUTED RESOURCE MANAGER: Output from this command shows the following summary information:

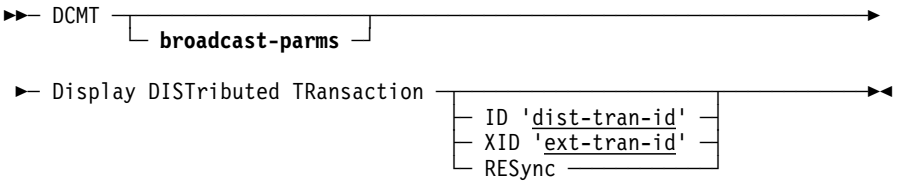
- The name of the resource manager
- The resource manager's status. Valid values are:
  - Initial or Closed — Resynchronization of the resource manager has not occurred.
  - Open — Resynchronization with the identified resource manager completed successfully.
  - ResyncQued — Resynchronization is in-progress or abnormally terminated.
  - ResyncCmpl — Resynchronization completed unsuccessfully, probably because the resource manager is not active.
- The time the resource manager was last started, if known to the local system. The time that is shown is a UTC (GMT) value.
- The number of distributed transactions pending resynchronization in which this resource manager has an interest.

Output from DCMT DISPLAY DISTRIBUTED RM 'rm-name': Output from this command includes the above summary information as well as a list of the distributed transactions in which the resource manager has an interest. The latter information may not always be available, depending on the type of resource manager being displayed. For a description of the transaction-related information, refer to the DCMT DISPLAY DISTRIBUTED TRANSACTION summary command.

A.7 DCMT DISPLAY DISTRIBUTED TRANSACTION

This new command displays information about distributed transactions.

A.7.1 Syntax



A.7.2 Parameters

broadcast-parms

Indicates to execute the DCMT command on all or a list of data sharing group members. Refer to A.31, “How to Broadcast System Tasks” on page A-49 for more information on broadcasting and **broadcast-parms** syntax.

DISTributed Transaction

Provides a list of distributed transactions. If no other options are entered, a list of all distributed transactions is displayed. Valid options are:

ID dist-tran-id

Provides detailed information about the distributed transaction that is assigned to this ID. The *dist-tran-id* value must be enclosed in single quotes, using the format 'xxxxxxxx::yyyyyyyyyyyyyyyy'. The *dist-tran-id* value must match a value that appears on the summary display.

XID ext-tran-id

Provides detailed information about the distributed transaction assigned to this ID. The *ext-tran-id* value is the hexadecimal value of an XA XID or a RRS URID. The *ext-tran-id* value must be enclosed in single quotes.

RESync

Displays a summary of all distributed transactions pending resynchronization.

A.7.3 Examples

The example below illustrates the use of the DCMT DISPLAY DISTRIBUTED TRANSACTION command to obtain a summary of known distributed transactions.

```
DCMT D DIST TR
Task/LTE  |Distributed transaction ID-Branch ID       |Ctrl|State|Ind|Outcome
*none     |SYSTEM74::01650D6EDFB1AB93-01650D6A79F31E50|IDMS|InDbt|Rsy|OK
00123     |SYSTEM74::01650D7920C25DE0-01650D75F0FC2550|IDMS|InDbt|-  |OK
```

The example below illustrates the use of the DCMT DISPLAY DISTRIBUTED TRANSACTION command to obtained detailed information about an individual transaction.

```
DCMT D DIST TR ID 'SYSTEM74::01650D6EDFB1AB93'   Top level transaction branch:
Task/LTE           *none                          Res. indicator   Rsy
Distr. tr. ID      SYSTEM74::01650D6EDFB1AB93      Control          IDMS
Branch ID          01650D6A79F31E50                State            InDoubt
Local ID           *none                          Outcome          OK
External ID        *none

Controlling interest:
   RM name         SYSTEM74::DSI_SRV              Role      SDSRM
   Interest state  InDoubt                        Protocol  Presumed Abort
   One phase commit Not Supported                 Journal   Yes
   Resync          Yes                            Manual    Yes
   Restart         Yes
```

```
Subordinate transaction branch
Branch ID          01650DA79956B32B               State            InDoubt
Local ID                 1416                     Outcome          OK
External ID        *none
```

A.7.4 Usage

Output from DCMT DISPLAY DISTRIBUTED TRANSACTION: The following summary information is shown for distributed transactions included in this display.

- The task or logical terminal element associated with the transaction. If an active task is processing the transaction, the task ID is shown. If a logical terminal but no task is associated with the transaction, the LTE's ID is shown. A distributed transaction that is pending resynchronization or pending completion by RRS or an XA transaction manager may have no associated task or logical terminal.
- The distributed transaction ID (DTRID) assigned to the transaction.
- The identifier of the top-level branch of the transaction.
- The type of the transaction manager, or coordinator, that is in control of the transaction. Possible types are:
  - IDMS — Advantage CA-IDMS
  - RRS — RRS
  - XA — XA transaction manager
  - CICS — CICS system
- The state of the transaction. Possible states are:
  - Reset — InReset
  - InFl — InFlight
  - InPrp — InPrepare

  - InDbt — InDoubt
  - LstAg — LastAgent
  - InBck — InBackout
  - InCmt — InCommit
  - Forg — Forgotten
- An indication of whether this transaction is pending resynchronization. Possible values are:
  - Rsy — The transaction is pending resynchronization
  - Rst — The transaction was restarted and is pending resynchronization
- The transaction's outcome to date. Possible outcomes are:
  - OK — OK
  - OK_P — OK_Pending
  - FGT — Forget
  - BACK — Backout
  - BK_P — Backout_Pending
  - HC — Heuristic Commit
  - HM — Heuristic Mixed
  - HR — Heuristic Reset

For information on the following, see 3.3, “Two-Phase Commit Support Within Advantage CA-IDMS” on page 3-8:

- Distributed Transaction Identifier (DTRID)
- Transaction State
- Transaction Outcome

Output from DCMT DISPLAY DISTRIBUTED TRANSACTION ID/XID: The detail displayed for a distributed transaction includes information on each of the branches comprising the transaction. A transaction always has one top-level branch and may or may not have subordinate branches.

The information listed below is displayed for a top-level branch. See the description above of the summary output for details on each of these fields:

- The task or logical terminal ID that is associated with the transaction.
- An indication of whether this transaction is pending resynchronization.
- The DTRID assigned to the transaction.
- The type of the transaction manager that is in control of the transaction.

The following information is displayed for all transaction branches:

- The identifier assigned to the branch.
- The state of the transaction branch.
- The local transaction ID (LID) if database access was performed under control of the branch.
- The commit outcome to date for the transaction branch.
- The external identifier assigned to the transaction branch if applicable.
- Information on interests in the branch that have been registered by resource managers

The following information is displayed for each interest associated with a transaction:

- An indication of whether this is a controlling interest. A controlling interest is one that was registered by the transaction's coordinator.
- The name of the resource manager that registered the interest.
- The role that the associated resource manager plays within the transaction. Possible values are:
  - SDSRM — Server Distributed Resource Manager
  - CRM — Communications Resource Manager
  - PART — Participant
- The state of the interest.
- The commit protocol used by the resource manager. Possible values are:
  - Presumed Abort
  - Presumed Nothing
- Whether the resource manager supports a one-phase commit protocol. Possible values are:
  - Supported — Indicating that the resource manager is capable of processing a one-phase commit request
  - Not Supported — Indicating that the resource manager is not capable of processing a one-phase commit request
  - Only — Indicating that the resource manager is only capable of supporting a one-phase commit request
- An indication of whether the interest is to be journaled or not.
- An indication of whether resynchronization is pending with the interest's resource manager.
- An indication of whether the transaction must be completed manually, due to a resynchronization failure.
- An indication of whether the interest was restarted following an abnormal system termination.

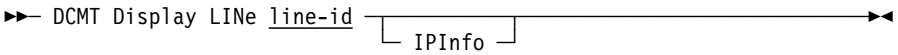
For information on the following, see 3.3, “Two-Phase Commit Support Within Advantage CA-IDMS” on page 3-8:

- Distributed Transaction Identifier (DTRID)
- Transaction and interest states
- Transaction and interest outcomes
- Transaction branches and interests

A.8 DCMT DISPLAY LINE

The DCMT DISPLAY LINE command is enhanced with a new parameter that provides TCP/IP information.

A.8.1 Syntax



A.8.2 Parameters

line-id

The ID of a line specified on the system generation line statement.

IPInfo

Optional keyword to request output of TCP/IP information, which consists of:

- Statistics: central version global data
- Default and current TCP/IP stack.
- A table that describes the TCP/IP stacks defined to the operating system.

A.8.3 Example

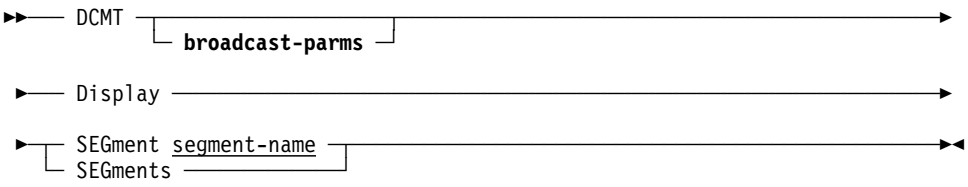
Output from the command DCMT D LINE TCPIP IPINFO:

```
Number of sockets created                          2
Number of sockets currently open                   2
HWM of concurrent open sockets (global)            2
HWM of concurrent open sockets (1 LTERM)           1
Number of "no BULK PTE" connection rejects         0
Number of socket reads                           506
Number of socket writes                          253
Default tcp/ip stack job name                RUNTCP1
Current tcp/ip stack job name

Hostname  IP address       Job name  Active  IPv6
--------  ---------------  --------  ------  ----
HOST1CA   123.456.789.1    RUNTCP1      Y      N
                           RUNTCP1B     N
HOST1IBM  123.456.789.2    TCPIPH1I     Y      N
HOST1CF   123.456.789.3    TCPIPH1X     Y      N
```

A.9 DCMT DISPLAY SEGMENT

This command has been enhanced to allow all segments to be displayed.

A.9.1 Syntax



A.9.2 Parameters

SEGment
Displays information about the areas in a segment.

segment-name
The name of the segment whose information is to be displayed.

SEGments
Lists all segments known to the runtime system.

A.9.3 Example

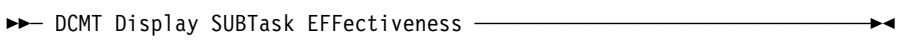
DCMT DISPLAY SEGMENTS

```
Segment-Name   Schema-Name      Type  Pg-Grp  Radix       Datetime-stamp
  AAA                        Network      25      8     2004-03-02-09.28.15
  DAR                           SQL        0      8     2004-03-02-09.28.14
  DBCR                       Network      15      8     2004-03-02-09.28.15
  EMPDEMO                    Network       0      8     2004-03-02-09.28.14
  ETOT                       Network      32      8     2004-03-02-09.28.14
  KJM                        Network      35      8     2004-03-02-09.28.15
  LRD                        Network      30      8     2004-03-02-09.28.15
  QADICT                     Network       0      8     2004-03-02-09.28.14
  QAMISC                     Network       0      8     2004-03-02-09.28.14
  R120DICT                   Network       0      8     2004-03-02-09.28.14
  SYSDAR                        SQL        0      8     2004-03-02-09.28.14
  SYSDEF                     Network       0      8     2004-03-02-09.28.14
  SYSDICT                    Network       0      8     2004-03-02-09.28.14
  SYSLOCAL                   Network       1      8     2004-03-02-09.28.14
  SYSMSG                     Network       0      8     2004-03-02-09.28.14
  SYSSQL                        SQL        0      8     2004-03-02-09.28.14
  SYSUSER                    Network       0      8     2004-03-02-09.28.14
  USERDB                        SQL        0      8     2004-03-02-09.28.14
  USERDB2                       SQL        2      8     2004-03-02-09.28.14
  VSAMT                      Network       0      8     2004-03-02-09.28.14
V74  ENTER NEXT TASK CODE:
```

A.10 DCMT DISPLAY SUBTASK

This modified command provides you with information about the CPU's effectiveness.

A.10.1 Syntax



A.10.2 Example

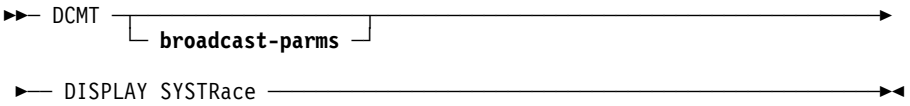
The following example illustrates the use of DISPLAY SUBTASK EFFECTIVENESS to display CPU effectiveness:

```
DCMT D SUBTASK EFFECTIVENESS
*** Subtask display ***
 Subtask                                          % CPU
  Name       Elapsed time     Total CPU time    effectiveness
 --------   --------------    --------------    -------------
 MAINTASK    00:02:41.9079     00:00:11.1475          66
```

A.11 DCMT DISPLAY SYSTRACE

This new command displays information about the system trace.

A.11.1 Syntax



A.11.2 Parameters

broadcast-parms

Indicates to execute the DCMT command on all or a list of data sharing group members. Refer to A.31, “How to Broadcast System Tasks” on page A-49 for more information on broadcasting and **broadcast-parms** syntax.

A.11.3 Example

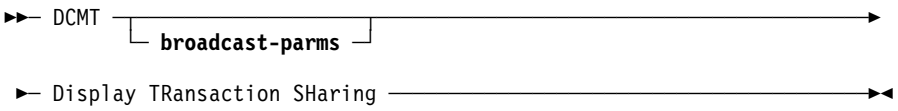
The following example illustrates the use of the DCMT DISPLAY SYSTRACE command to see whether the trace is on and, if it is on, the number of entries in the trace table:

```
DCMT DISPLAY SYSTRACE
System trace is ON entries        2500
```

A.12 DCMT DISPLAY TRANSACTION SHARING

This new command displays information about transaction sharing.

A.12.1 Syntax



A.12.2 Parameters

broadcast-parms
Indicates to execute the DCMT command on all or a list of data sharing group members. Refer to A.31, “How to Broadcast System Tasks” on page A-49 for more information on broadcasting and **broadcast-parms** syntax.

A.12.3 Example

The following example illustrates the use of the DCMT DISPLAY TRANSACTION SHARING command to see whether transaction sharing is on:

```
DCMT D TRANSACTION SHARING
Transaction Sharing OFF
```

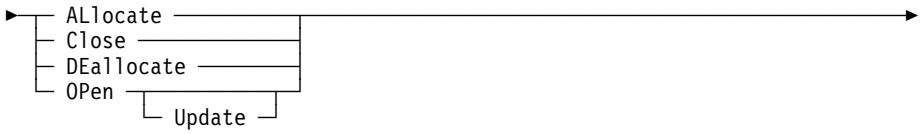
A.13 DCMT VARY AREA

This command has been enhanced to provide allocation or deallocation of all files associated with a specified area.

This section describes only the new parameters of this command. For more information, see *Advantage CA-IDMS System Tasks and Operator Commands*.

A.13.1 Syntax

Expansion of file-status



A.13.2 Parameters

ALlocate

Dynamically allocates all files associated with the specified area. The files are allocated using their currently assigned data set name.

Note: This option applies to z/OS or OS/390, z/VM, and BS2000/OSD files only.

DEallocate

Dynamically deallocates all files associated with the specified area.

Note: This option applies to z/OS or OS/390, z/VM, and BS2000/OSD files only.

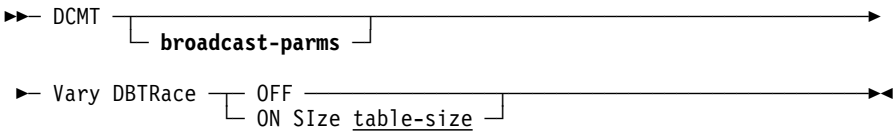
A.13.3 Usage

Dynamic file deallocation: In order to deallocate an area, it and any areas contained in the same files must be offline.

A.14 DCMT VARY DBTRACE

This new command lets you turn the database trace on and off and set the size of the trace table.

A.14.1 Syntax



A.14.2 Parameters

broadcast-parms
Indicates to execute the DCMT command on all or a list of data sharing group members. Refer to A.31, “How to Broadcast System Tasks” on page A-49 for more information on broadcasting and **broadcast-parms** syntax.

OFF
Disables database tracing.

ON
Enables database tracing.

SIze table-size
Specifies the size of the database trace table, where *table-size* is the size of the table in kilobytes.

A.14.3 Examples

The following example illustrates the use of the DCMT VARY DBTRACE command to turn the database trace off:

```
DCMT VARY DBTRACE OFF
DBTrace is OFF
```

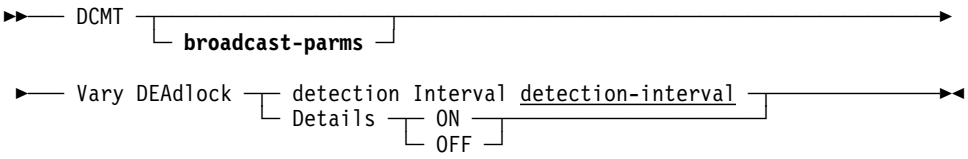
The following example illustrates the use of the DCMT VARY DBTRACE command to set the size of the database trace table:

```
DCMT VARY DBTRACE ON SIZE 6144
DBTrace is ON size   6144
```

A.15 DCMT VARY DEADLOCK

This command has been enhanced to allow control over whether to generate additional information during deadlock resolution.

A.15.1 Syntax



A.15.2 Parameters

Details
Initiates or terminates the generation of additional messages during the resolution of a deadlock.

ON
Initiates the generation of message DC001001.

OFF
Terminates the generation of message DC001001.

The system default option is OFF, unless overridden by a DEADLOCK_ DETAILS parameter included in the SYSIDMS file.

Note: To display the deadlock details current setting, see A.5, “DCMT DISPLAY DEADLOCK” on page A-9.

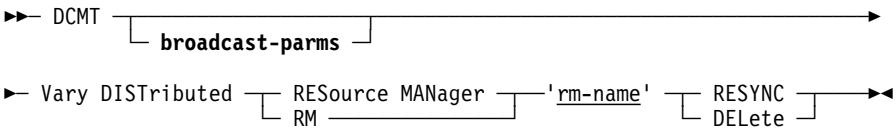
A.15.3 Usage

Generating additional deadlock information: If you vary deadlock details ON, the deadlock detector provides additional information in the form of DC001001 messages during the processing of a deadlock. This information can prove useful in researching the cause of a deadlock situation because it identifies the programs and subschemas involved. However, it also increases the overhead of detecting deadlocks. In an active system in which waits for resources are common, it is recommended that the generation of deadlock details only be initiated when researching a specific deadlock situation.

A.16 DCMT VARY DISTRIBUTED RESOURCE MANAGER

This new command initiates resynchronization with or deletes the specified resource manager.

A.16.1 Syntax



A.16.2 Parameters

broadcast-parms
Indicates to execute the DCMT command on all or a list of data sharing group members. Refer to A.31, “How to Broadcast System Tasks” on page A-49 for more information on broadcasting and **broadcast-parms** syntax.

RESource MANager
Valid values are '*rm-name*' and spaces. If '*rm-name*' is not specified, a list of all known resource managers is displayed.

rm-name
Specifies the name of the resource manager to display. The *rm-name* value must be enclosed in single quotes using the format 'xxxxxxxx::yyyyyyyy'. The *rm-name* value must match a value that appears on the summary display.

RESYNC
Specifies that resynchronization be performed on the named resource manager.

DELete
Specifies that the named resource manager and any interests associated with it be deleted.

A.16.3 Example

The following example illustrates the use of the DCMT VARY DISTRIBUTED RESOURCE MANAGER command to initiate resynchronization with the SYSTEM74::DSI_CLI resource manager.

```
DCMT V DIST RM 'SYSTEM74::DSI_CLI' RESYNC

Resource manager SYSTEM74::DSI_CLI  RESYNC successfully initiated.
```

A.16.4 Usage

Resource manager limitations: Not all resource managers support resynchronization initiated through a DCMT VARY DISTRIBUTED RESOURCE MANAGER command. This is the case for CICS resource managers and resource managers whose name ends with "DSI_SRV". Resynchronization with such resource managers can be initiated only from the associated front-end system. An error message is displayed if the specified resource manager does not support resynchronization through this command.

Deleting resource managers: When a resource manager is deleted, all record of that resource manager is eliminated from the system. The DCMT VARY RESOURCE MANAGER DELETE command should only be used when the resource manager no longer exists. For example, when a DC/UCF system is removed from the network. By deleting the resource manager, no further attempt is made to resynchronize with that resource manager at startup.

Note: Only resource managers whose name ends in "DSI_CLI" or "DSI_SRV" can be deleted.

For resource managers whose name ends in "DSI_SRV": Use the DCMT DISPLAY DISTRIBUTED RESOURCE MANAGER command to determine if the resource manager has associated interests, before deleting the resource manager. If the resource manager's name ends in "DSI_SRV" the delete request fails if there are outstanding interests. Use the DCMT VARY DISTRIBUTED TRANSACTION command to manually complete each transaction before deleting the resource manager.

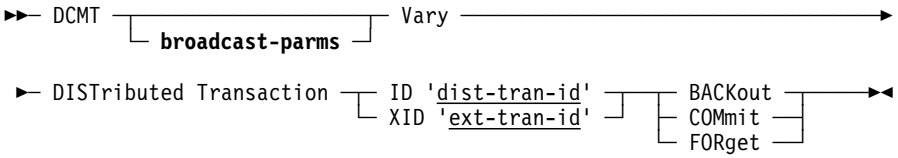
For resource managers whose name ends in "DSI_CLI": If the resource manager's name ends in "DSI_CLI", its associated interests are deleted automatically as part of deleting the resource manager. After deleting the resource manager, use the DCMT VARY DISTRIBUTED TRANSACTION command to complete any transactions whose interests were deleted. Since no further attempt will be made to communicate with the deleted transaction manager, the transactions will now be able to complete.

A.17 DCMT VARY DISTRIBUTED TRANSACTION

This new command forces the completion of a distributed transaction that either:

- Is pending resynchronization
- Has no associated or task or user session

A.17.1 Syntax



A.17.2 Parameters

broadcast-parms
Indicates to execute the DCMT command on all or a list of data sharing group members. Refer to A.31, “How to Broadcast System Tasks” on page A-49 for more information on broadcasting and **broadcast-parms** syntax.

DISTributed Transaction
Provides a list of distributed transactions. If no other options are entered, a list of all distributed transactions is displayed. Valid options are:

ID dist-tran-id
Provides detailed information about the distributed transaction that is assigned to this ID. The *dist-tran-id* value must be enclosed in single quotes, using the format 'xxxxxxxx::yyyyyyyyyyyyyyyy'. The *dist-tran-id* value must match a value that appears on the summary display.

XID ext-tran-id
Provides detailed information about the distributed transaction assigned to this ID. The *ext-tran-id* value is the hexadecimal value of an XA XID or a RRS URID. The *ext-tran-id* value must be enclosed in single quotes.

BACkout
Specifies that the transaction should be backed out. BACkout can only be specified *if* the transaction's state is InDoubt or InBackout.

COMmit
Specifies that the transaction should be committed. COMmit can only be specified *if* the transaction's state is InDoubt or InCommit.

FORget
Specifies that the transaction should be forgotten. FORget can only be specified if the transaction's state is InBackout or InCommit.

A.17.3 Example

The example below illustrates the use of the DCMT VARY DISTRIBUTED TRANSACTION command to complete a distributed transaction whose state is InDoubt.

```
DCMT V DIST TR ID 'SYSTEM74::01650D6EDFB1AB93' COMMIT

Transaction COMMIT  initiated.
```

A.17.4 Usage

Completing transactions manually: Only distributed transactions that are pending resynchronization or have no task or user session can be completed manually using a DCMT VARY DISTRIBUTED TRANSACTION command. The need for issuing this command is extremely rare and only as a result of a resynchronization failure. For more information regarding resynchronization and the need for completing transactions manually, see 3.3, “Two-Phase Commit Support Within Advantage CA-IDMS” on page 3-8.

When a DCMT command is used to force an InDoubt transaction to commit or backout, the transaction branch is flagged as being heuristically committed or backed out and its outcome is HC or HR respectively. Heuristically completed transactions must be explicitly forgotten by:

- Issuing a DCMT command

Or

- Allowing the coordinator to direct that the branch be forgotten

The coordinator should be given the chance to do so, unless it is permanently disabled or its journal files (in the case of Advantage CA-IDMS) were prematurely formatted thereby eliminating the information required to complete the transaction.

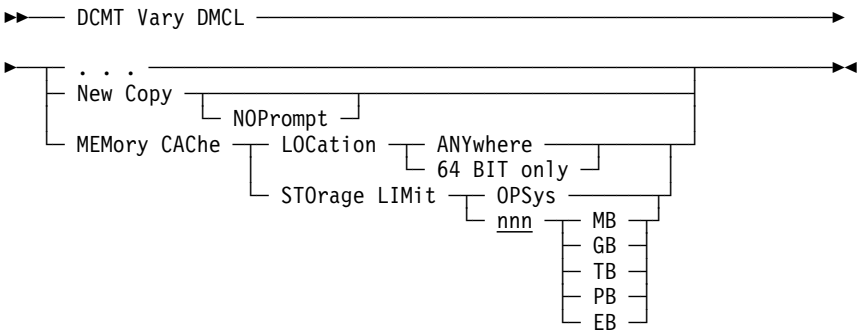
A.18 DCMT VARY DMCL

DCMT VARY DMCL is extended to provide the following enhancements:

- Memory cache option to prevent memory cache overflow
- Noprompt option to inhibit prompting for permission to proceed with changes

This section describes only the new parameters of this command. For more information, see *Advantage CA-IDMS System Tasks and Operator Commands*.

A.18.1 Syntax



A.18.2 Parameters

NOPrompt
Inhibits prompting for permission to proceed. If NOPROMPT is specified, the changes are displayed and then immediately implemented.

MEMORY CACHE
Indicates global options for memory cache:

LOCATION
Indicates where to allocate the storage for memory cache:

ANYWHERE
Memory cache storage is allocated from 64-bit storage; if no or not enough 64-bit storage is available, dataspace storage is acquired.

64 BIT only
Memory cache storage is allocated from 64-bit storage; if no or not enough 64-bit storage is available, memory caching fails.

STORAGE LIMIT
Controls the amount of storage used for memory caching:

OPSYS
Memory cache storage can be acquired until the operating system limit is reached. For 64-bit storage, the operating system limit is set through the MEMLIMIT parameter; for dataspace storage, the limit is optionally imposed by an operating system exit.

nnn **MB, GB, TB, PB, EB**
Advantage CA-IDMS controls the amount of memory cache storage if the value specified is smaller than the operating system limit. nnn must be a positive value between 1 and 32767. MB, GB, TB, PB, EB indicate the unit in which nnn is expressed. The abbreviations stand for Mega Byte (2**20), Giga Byte (2**30), Tera Byte (2**40), Peta Byte (2**50), and Exa Byte (2**60).

Note: A dynamic change to memory caching through DCMT VARY DMCL applies only to files that are opened AFTER the DCMT VARY DMCL command was issued.

A.18.3 Examples

The following examples illustrate the use of the DCMT VARY DMCL command to set where and how much memory cache storage can be allocated.

DCMT VARY DMCL MEMORY CACHE LOCATION ANYWHERE

```
DCMT V DMCL MEMORY CACHE LOCATION ANYWHERE
DMCL MEMORY CACHE LOCATION ANYWHERE
```

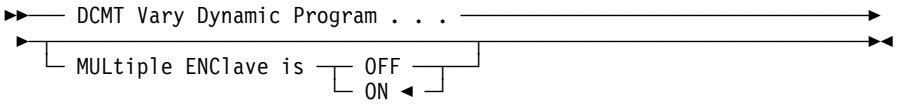
DCMT VARY DMCL MEMORY CACHE STORAGE LIMIT OPSYS

```
DCMT V DMCL MEMORY CACHE STORAGE LIMIT OPSYS
DMCL MEMORY CACHE STORAGE LIMIT OPSYS
```

A.19 DCMT VARY DYNAMIC PROGRAM

DCMT VARY DYNAMIC PROGRAM defines programs to the system at system run time (that is, dynamically). The system uses information supplied in the DCMT VARY DYNAMIC PROGRAM command to build a program definition element (PDE) for the program. Programs defined in this way exist only for the duration of system execution and have no effect on the system definition stored in the data dictionary. This command was modified to support the multiple enclave feature.

A.19.1 Syntax



A.19.2 Parameters

MULtiple ENClave is

Specifies whether the system allows the same language enclave as other LE programs. This parameter is only meaningful for COBOL programs.

ON

Specifies that this program can participate in a multiple program LE enclave. This is the default.

Note: This value is effective only if MULTIPLE ENLAVE is ON is specified on the SYSTEM statement in the sysgen.

OFF

Specifies that this program cannot participate in a multiple program LE enclave.

A.19.3 For More Information

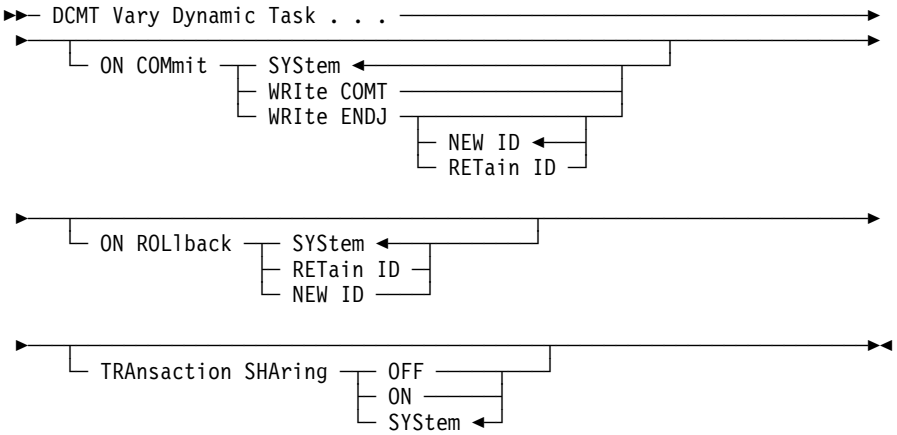
About program definition at system generation time and PDEs, refer to documentation of the PROGRAM statement in *Advantage CA-IDMS System Generation Guide.*

A.20 DCMT VARY DYNAMIC TASK

This command allows the user to define tasks at system runtime, thereby allowing tasks that are not defined in the sysgen to be used. DCMT VARY DYNAMIC TASK is extended in Release 16.0 to include parameters that control transaction sharing and commit and rollback behavior.

This section describes only the new parameters of this command. For more information, see *Advantage CA-IDMS System Tasks and Operator Commands*.

A.20.1 Syntax



A.20.2 Parameters

ON COMmit

Specifies options that control commit behavior. These options apply only to commit operations in which the database session remains active.

SYStem

Specifies that the commit behavior for the task should default to that specified for the system.

WRIte COMT

Specifies that a COMT journal record should be written.

WRIte ENDJ

Specifies that an ENDJ journal record should be written.

NEW ID

Specifies that a new local transaction ID should be assigned to the next transaction started by the database session.

RETain ID

Specifies that the current local transaction ID should be assigned to the next transaction started by the database session.

ON ROLlback
Specifies options that control rollback behavior. These options apply only to rollback operations in which the database session remains active.

SYStem
Specifies that the rollback behavior for the task should default to that specified for the system.

RETain ID
Specifies that the current local transaction ID should be assigned to the next transaction started by the database session.

NEW ID
Specifies that a new local transaction ID should be assigned to the next transaction started by the database session.

TRAnsaction SHaring
Specifies the setting for the transaction sharing option.

ON
Specifies that transaction sharing should be initially enabled for any task of this type.

OFF
Specifies that transaction sharing should be initially disabled for any task of this type.

SYStem
Specifies that the transaction sharing option for a task of this type is based on the system default established in the sysgen or by a DCMT VARY TRANSACTION SHARING command.

A.20.3 Example

The following example defines a new task code 'FOU'. Transaction sharing is initially enabled for the task.

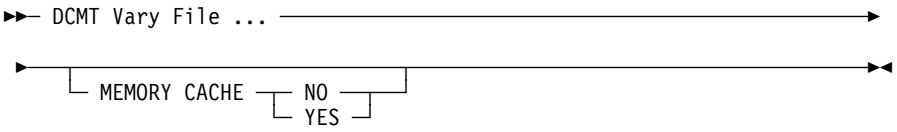
```
DCMT V D T FOU INVOKES MYPROG TRANSACTION SHARING ON
IDMS DC273001 V73 USER:JSMITH Task FOU Added
```

A.21 DCMT VARY FILE

This command changes the status of a specified file. This command is extended in Release 16.0 to let you specify if the file is cached in memory.

This section describes only the new parameters of this command. For more information, see *Advantage CA-IDMS System Tasks and Operator Commands*.

A.21.1 Syntax



A.21.2 Parameters

MEMORY
Specifies if the file is cached in memory.

NO
Specifies that the file is not cached in memory.

YES
Specifies that the file is cached in memory.

A.21.3 Example

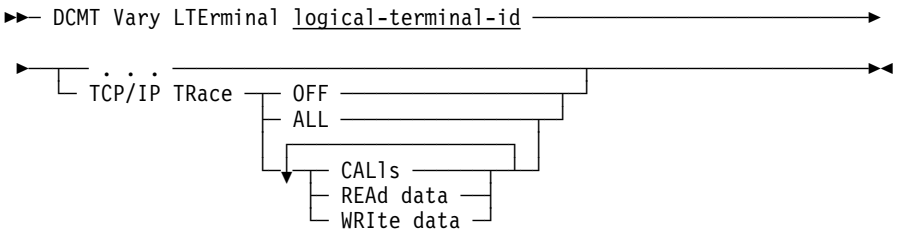
The following example causes file EMPDEMO.EMPDEMO to be cached in memory:

```
DCMT V FILE EMPDEMO.EMPDEMO MEMORY CACHE YES

-------- Data File -------- Mode Stat Pg-Size  Fl-Type M-Cache S-Cache DD-Name
EMPDEMO.EMPDEMO              Ret    0   4276  non-VSAM   Yes      No   EMPDEMO
  Pre-fetch: Not-Allowed (DMCL)       Pages per Track   11      DISP=SHR (DMCL)
  DSname: (DMCL).. DBDC.SYSTEM71.EMPDEMO.EMPDEMO
  DSname: (DMCL).. DBDC.SYSTEM71.EMPDEMO.EMPDEMO

---------- Area -----------   Lock    Lo-Page    Hi-Page #Ret  #Upd #Tret #Ntfy
EMPDEMO.EMP-DEMO-REGION        Ret      75001      75100    0     0     0     0
 Stamp: 1999-11-16-08.17.07.104886  Pg grp: 0      NoShare  NoICVI  NoPerm
```

A.22 DCMT VARY LTERM

This command has been enhanced to support tracing for TCP/IP socket programs.

A.22.1 Syntax



A.22.2 Parameters

OFF
Terminates TCP/IP tracing.

CALLS
Activates TCP/IP function tracing. A record is output to the LOG.

READ data
Specifies an entry is written to the log when data is read through the TCP/IP function.

WRITE data
Specifies an entry is written to the log when data is written using the TCP/IP function.

ALL
Combines CALLS, READ data and WRITE data.

Note: Use the TCP/IP trace facility with care because it can generate a lot of output.

A.22.3 Example

This command activates the TCP/IP trace.

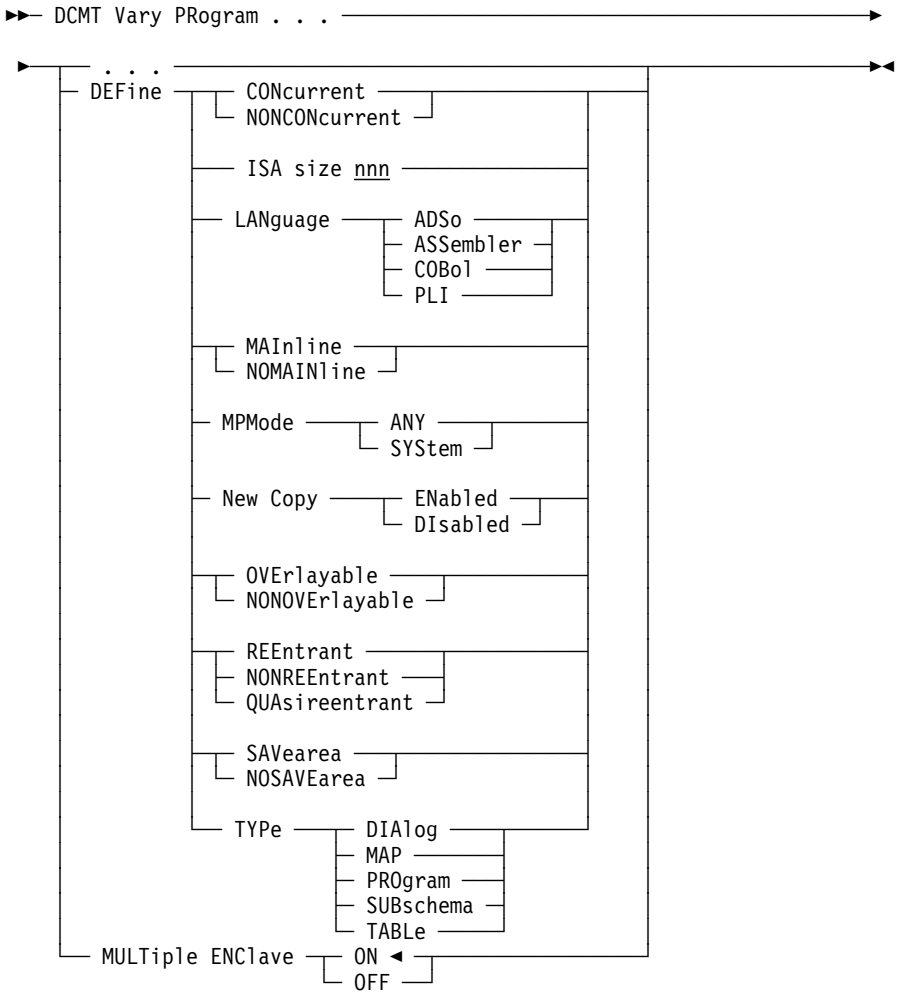
```
DCMT V LTE IPLTE012 TCP/IP TRACE ALL

IDMS DC267016 Vnnnn USER:***  LTERM IPLTE012 TCP/IP TRACE VARIED TO ALL
```

A.23 DCMT VARY PROGRAM

This command changes attributes in the program definition element of an existing program. VARY PROGRAM is extended in Release 16.0 to allow you to modify any program attribute that is specified in the sysgen.

This section describes only the modified syntax of this command. For more information, see *Advantage CA-IDMS System Tasks and Operator Commands*.

A.23.1 Syntax



Note: The following restrictions apply:

- The only parameter that cannot be changed is the RESIDENT parameter.
- You can change the LANGUAGE or TYPE parameters of a program only if the program is DISABLED.

A.23.2 Parameters

CONcurrent
Specifies that the program can be used by multiple tasks at the same time. If the program is reentrant or quasi-reentrant, one copy of the program is used to process all requests. If the program is nonreentrant, as many copies of the program are used as necessary to process requests concurrently.

NONCONcurrent
Specifies that the program can be used by only one task.

ISA size nnn
For Assembler and PL/I programs only, specifies the amount of storage, in bytes, allocated for the program's initial storage area (ISA). If an ISA is specified, GET STORAGE statements are not required in the program because the system automatically allocates the requested storage when the program begins executing. The storage address is passed in register 11. *nnn* represents an integer in the range 0 through 2,147,483,647.

LANguage
Identifies the language in which the program is written:

- ADSo — Advantage CA-IDMS dialog
- ASSembler — an assembler program
- COBol — COBOL program
- PLI — PL/I program

MAInline
For Advantage CA-IDMS dialogs only, indicates the dialog is a mainline dialog. Dialogs defined as MAINLINE are entry points into applications. The names of mainline dialogs are eligible for display on the Advantage CA-IDMS menu screen as allowed by ADSO statement specifications.

If you specify MAINLINE, the dialog must be generated with the MAINLINE attribute but it does not have to be assigned a task code during system generation.

NOMAINline
For Advantage CA-IDMS dialogs only, indicates the dialog is not a mainline dialog.

MPMode
Specifies the multiprocessing mode (MPMODE) for the program. *SYStem* directs the system to assign an MPMODE to the program at execution time. *ANY* assigns an MPMODE of ANY to the program. ANY is appropriate for reentrant and quasi-reentrant programs that are defined without storage protection.

New Copy
Specifies whether the new copy facility is enabled for the program or subschema. *ENable* enables the new copy facility for the program or subschema. *DIsabled* disables the new copy facility for the program or subschema.

OVErlayable
Specifies that the program can be overlaid in the program pool. You should specify OVERLAYABLE only for executable programs invoked through normal DC mechanisms.

NONOVErlayable
Specifies that the program cannot be overlaid in the program pool. You should specify NONOVERLAYABLE for nonexecutable programs (for example, tables) to prevent such programs from being overwritten in the program pool while they are in use.

REEntrant
Specifies that the program is reentrant. To be declared reentrant, the program must acquire all variable storage dynamically and must not modify its own code.

NONREEntrant
Specifies that the program is nonreentrant. Programs that modify their own code and do not ensure the modified code is returned to its original state when the program is not in control *must* be declared NONREENTRANT.

QUAsireentrant
For COBOL programs only, specifies the program is quasi-reentrant. To be declared quasi-reentrant, a program must not modify its own code unless the program ensures the modified code is returned to its original state when the program is not in control. Quasi-reentrant programs are permitted to use working storage because, each time the program is executed, the system creates a separate copy of its working storage in the storage pool. This technique makes the program, in effect, reentrant.

SAVearea
For Assembler programs only, specifies that the system will acquire a save area automatically before each execution of the program. The save area address is passed to the program in register 13. You should specify SAVEAREA or accept it by default if the program uses normal IBM calling conventions and, at the start of execution, saves registers in the save area.

NOSAVEarea
For Assembler programs only, specifies the system will not acquire a save area for the program automatically.

TYPe
Specifies the program type: DIALog, MAP, PROgram, SUBschema, or TABLe.

MULtiple ENClave is
Specifies whether the system allows the same language enclave as other LE programs. This parameter is only meaningful for COBOL programs.

ON
Specifies that this program can participate in a multiple program LE enclave. This is the default.

Note: This value is effective only if MULTIPLE ENLAVE is ON is specified on the SYSTEM statement in the sysgen.

OFF

Specifies that this program cannot participate in a multiple program LE enclave.

A.23.3 Examples

The following example illustrates using the VARY PROGRAM command to change the language:

```
DCMT VARY PROGRAM TESTPROG DEFINE LANGUAGE ASSEMBLER
IDMS DC262013 V71 USER:JSMITH  PROGRAM TESTPROG CDMSLIB LANGUAGE CHANGED
```

The following example illustrates using the VARY PROGRAM command to change the multiprocessing mode:

```
DCMT VARY PROGRAM TESTPROG DEFINE MPMODE ANY
IDMS DC262012 V71 USER:JSMITH  PROGRAM TESTPROG CDMSLIB VARIED SUCCESSFULLY
```

The following example illustrates using the VARY PROGRAM command to change the save area:

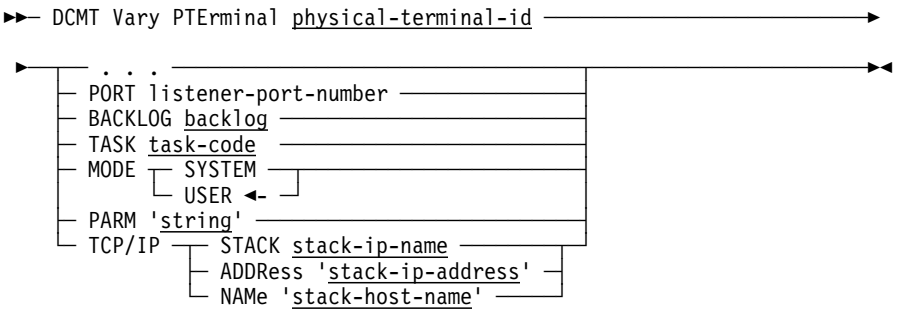
```
DCMT VARY PROGRAM TESTPROG DEFINE NOSAVEAREA
IDMS DC262012 V71 USER:JSMITH  PROGRAM TESTPROG CDMSLIB VARIED SUCCESSFULLY
```

A.24 DCMT VARY PTERM

The DCMT VARY PTERM statement is used to manage generic listeners. Varying a listener PTERM OFFLINE shuts down the generic listener, while varying ONLINE starts the service. The DCMT VARY PTERM command is enhanced so you can dynamically change parameters on a listener PTERM.

Note: Varying a generic listener OFFLINE only affects the listener but it does not affect server tasks that are executing.

A.24.1 Syntax



A.24.2 Parameters

listener-port-number

Number of the listener port. The port number is an integer in the range of 0 through 65535.

backlog

The value defines the maximum length for the queue of pending connections TCP/IP allows before rejecting new connection requests. *backlog* is a positive number between 1 and 1,147,483,647.

The value specified for backlog is not necessarily the value accepted by the LISTEN call. Each TCP/IP implementation has a limit of its own. Advantage CA-IDMS uses the lesser of the implementation's limit and the value specified for the backlog parameter.

stack-ip-name

The job name of the TCP/IP stack. The name is limited to 8 characters.

Specifying *ALL on a multi-homed system (z/OS only) causes listening to all active TCP/IP stacks. Specifying *DEFAULT causes listening to the default TCP/IP stack.

stack-ip-address

IP address of the host. The limit of an IP address depends on whether IPv4 or IPv6 is used: the limit in IPv4 is 15 characters; in IPv6 it is 45 characters.

stack-host-name
Name of the host. The maximum length of the host name is 64 characters.

task-code
Name of the task code to invoke when a connection request is received.

MODE is USER/SYSTEM
Indicates whether the task attached by the listener runs in SYSTEM or USER MODE. MODE is USER is the default. MODE is SYSTEM is only available for application programs written in assembler.

string:
A character string that is passed to the task attached by generic listening. *String* is limited to 80 characters.

A.24.3 Usage

Stack-ip-name, *stack-ip-address* and *stack-host-name* are mutually exclusive. Usually, it is undesirable to specify any of these parameters because doing so might tie a central version to an operating system image. The only situation in which specifying one of the above parameters is useful is when the central version runs on a multi-homed host and listening is to be restricted to a specific TCP/IP stack.

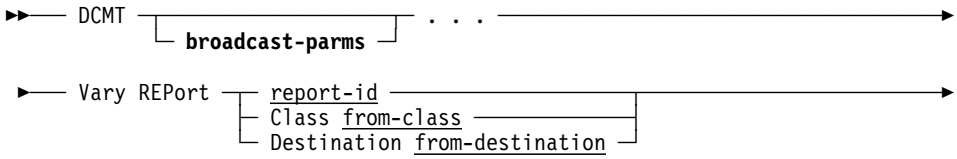
Note: Varying a generic listener OFFLINE only affects the listener but it does not affect server tasks that are executing.

A.25 DCMT VARY REPORT

This command has been enhanced to allow varying of multiple reports.

This section describes only the new parameters of this command. For more information, see *Advantage CA-IDMS System Tasks and Operator Commands*.

A.25.1 Syntax



A.25.2 Parameters

Class
Varies all reports in the specified print class.

from-class
The name of the print class for which reports are to be varied.

Destination
Varies all reports queued to the specified destination.

from-destination
The name of the destination for which queued reports are to be varied.

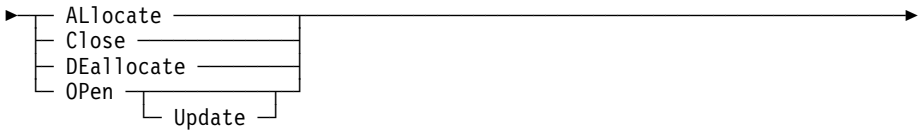
A.26 DCMT VARY SEGMENT

This command has been enhanced to provide allocation or deallocation of all files associated with a specified segment.

This section describes only the new parameters of this command. For more information, see *Advantage CA-IDMS System Tasks and Operator Commands*.

A.26.1 Syntax

Expansion of file-status



A.26.2 Parameters

ALlocate

Dynamically allocates all files associated with the specified segment. The files are allocated using their currently assigned data set name.

Note: This option applies to z/OS or OS/390, z/VM, and BS2000/OSD files only.

DEallocate

Dynamically deallocates all files associated with the specified segment.

Note: This option applies to z/OS or OS/390, z/VM, and BS2000/OSD files only.

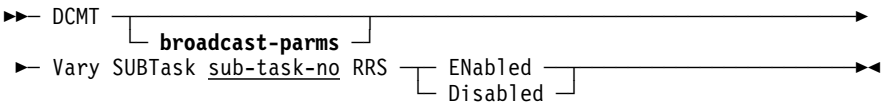
A.26.3 Usage

Dynamic file deallocation: In order to deallocate a segment, all of its areas must be offline.

A.27 DCMT VARY SUBTASK

This new command allows you to enable or disable the ability for a subtask to execute calls to RRS.

A.27.1 Syntax



A.27.2 Parameters

broadcast-parms

Indicates to execute the DCMT command on all or a list of data sharing group members. Refer to A.31, “How to Broadcast System Tasks” on page A-49 for more information on broadcasting and **broadcast-parms** syntax.

SUBTask sub-task-no

Specifies the number of the subtask whose work type is to change. The *sub-task-no* must be a value ranging from 2 through the maximum number of subtasks specified for the system.

Notes:

- Subtask 1 (MAINTASK) can never execute calls to RRS.
- The number of subtasks cannot be changed. Therefore, if a system was started with uni-tasking and without RRS support, the DCMT V SUBTASK command will fail.

RRS

Specifies whether the subtask can execute calls to RRS.

ENabled

Specifies the subtask can execute calls to RRS.

Disabled

Specifies the subtask cannot execute calls to RRS.

A.27.3 Examples

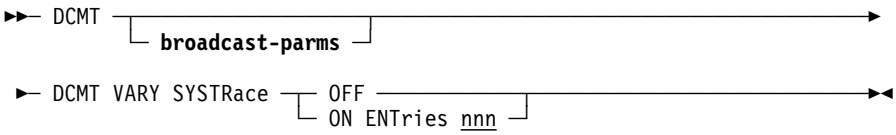
This example illustrates the use of the DCMT VARY SUBTASK command to change the type of work for subtask 2.

```
DCMT V SUBTASK 2 RRS DISABLED

IDMS DC285001 V73 USER:DEMO  Subtask 002 RRS DISABLED
```

A.28 DCMT VARY SYSTRACE

This new command lets you turn the system trace on and off and set the number of entries in the trace table.

A.28.1 Syntax



A.28.2 Parameters

broadcast-parms
Indicates to execute the DCMT command on all or a list of data sharing group members. Refer to A.31, “How to Broadcast System Tasks” on page A-49 for more information on broadcasting and **broadcast-parms** syntax.

OFF
Disables the system trace.

ON
Enables the system trace.

ENTRIES nnn
Specifies the size of the system trace table, where *nnn* is the number of entries in the table.

A.28.3 Examples

The following example illustrates the use of the DCMT VARY SYSTRACE command to turn the system trace off:

```
DCMT VARY SYSTRACE OFF
System trace is OFF
```

The following example illustrates the use of the DCMT VARY SYSTRACE command to set the number of entries in the system trace:

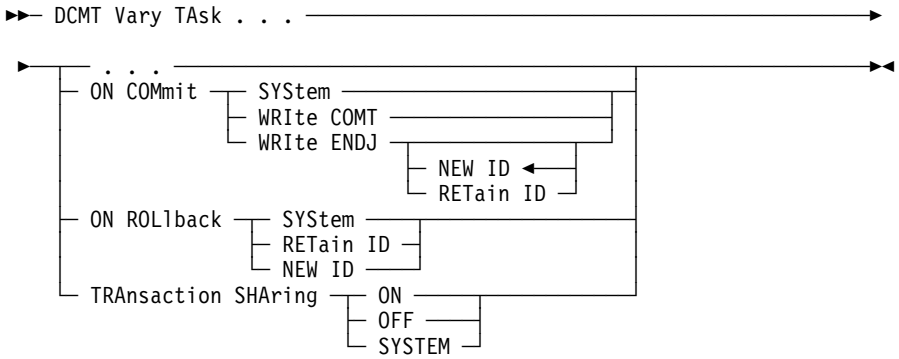
```
DCMT VARY SYSTRACE ON ENTRIES 5000
System trace is ON entries   5000
```

A.29 DCMT VARY TASK

This command changes attributes in the task definition element for a task that already exists. DCMT VARY TASK has been extended in Release 16.0 to include parameters that control transaction sharing and commit and rollback behavior.

This section describes only the new parameters of this command. For more information, see *Advantage CA-IDMS System Tasks and Operator Commands*.

A.29.1 Syntax



A.29.2 Parameters

ON COMmit
Specifies options that control commit behavior. These options apply only to commit operations in which the database session remains active.

SYStem
Specifies that the commit behavior for the task should default to that specified for the system.

WRIte COMT
Specifies that a COMT journal record should be written.

WRIte ENDJ
Specifies that an ENDJ journal record should be written.

NEW ID
Specifies that a new local transaction ID should be assigned to the next transaction started by the database session.

RETain ID
Specifies that the current local transaction ID should be assigned to the next transaction started by the database session.

ON ROLlback
Specifies options that control rollback behavior. These options apply only to rollback operations in which the database session remains active.

SYSTEM
Specifies that the rollback behavior for the task should default to that specified for the system.

RETain ID
Specifies that the current local transaction ID should be assigned to the next transaction started by the database session.

NEW ID
Specifies that a new local transaction ID should be assigned to the next transaction started by the database session.

TRAnsaction SHaring
Specifies the setting for the transaction sharing option.

ON
Specifies that transaction sharing should initially be enabled for any task of this type.

OFF
Specifies that transaction sharing should initially be disabled for any task of this type.

SYStem
Specifies that the transaction sharing option for a task of this type is based on the system default established by the sysgen's SYSTEM statement or by a DCMT VARY TRANSACTION SHARING command.

A.29.3 Example

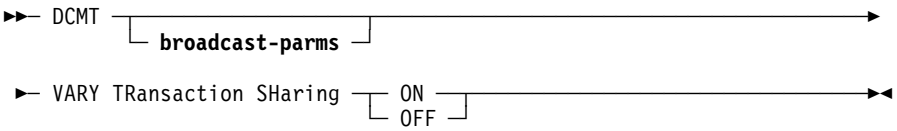
The following example alters task FOU so that it writes an ENDJ journal record on all commit operations:

```
DCMT V TA FOU ON COMMIT WRITE ENDJ
IDMS DC261018 V73 USER:KKK  ON COMMIT varied from SYSTEM to WRITE ENDJ NEW ID
```

A.30 DCMT VARY TRANSACTION SHARING

This new command lets you change the default transaction sharing option for the system.

A.30.1 Syntax



A.30.2 Parameters

broadcast-parms
Indicates to execute the DCMT command on all or a list of data sharing group members. Refer to A.31, “How to Broadcast System Tasks” on page A-49 for more information on broadcasting and **broadcast-parms** syntax.

ON
Specifies that, by default, transaction sharing is enabled for all tasks whose transaction sharing option specifies SYSTEM.

OFF
Specifies that, by default, transaction sharing is disabled for all tasks whose transaction sharing option specifies SYSTEM.

A.30.3 Example

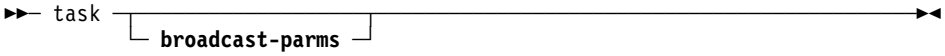
The following example illustrates the use of the DCMT VARY TRANSACTION SHARING command to activate transaction sharing:

```
DCMT V TRANSACTION SHARING ON
Transaction Sharing ON
```

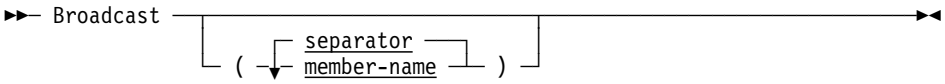
A.31 How to Broadcast System Tasks

If the central version (CV) is a member of a data sharing group (DSG), system tasks DCMT, DCUF and SEND can execute on other central versions that are members of the same DSG. This is called broadcasting. Broadcasting can be done to all the DSG members or a list of DSG members.

A.31.1 Syntax



Expansion of broadcast-parms



A.31.2 Parameters

broadcast-parms
Specifies how to execute the task.

Broadcast
Indicates that the specified task must be executed on one or more members of the data sharing group. If no list of members is given, the task is executed on ALL members.

separator
Separates multiple member names. Use a comma or at least one space.

member-name
Identifies the data sharing member (or a list) on which the specified task is to be executed.

A.31.3 Usage

Authorization: The issuing user must have the authority to execute the command on all members of the group to which it is directed. If the needed authority is not held on a member, the command will not execute on that member, but may on other members.

Output: The output from a broadcast command is segmented by member. All output from one member is displayed before that of another member. When broadcasting to all members, the output for the member on which the command is issued is displayed first. A header indicating the name of the member identifies other member's output.

A.31.3.1 Restrictions on the Broadcastable Tasks

DCMT: All commands can be broadcast, except DCMT ABORT, DCMT SHUTDOWN, DCMT VARY DMCL, DCMT QUIESCE, and DCMT DISPLAY/VARY NUCLEUS.

DCUF: Only the DCUF SHOW USER command can be broadcast.

SEND: All commands can be broadcast. Parameter prompting is not possible when broadcasting.

A.31.4 Examples

DCMT B V SEGMENT EMPDEMO

```
DCMT B V SEGMENT EMPDEMO OFFLINE
---------- Area -----------   Lock    Lo-Page    Hi-Page #Ret  #Upd #Tret #Ntfy
EMPDEMO.EMP-DEMO-REGION        Ofl      75001      75100    0     0     0     0
 Stamp: 2002-11-17-09.55.31.875826  Pg grp: 0      NoShare  NoICVI  NoPerm
EMPDEMO.INS-DEMO-REGION        Ofl      75101      75150    0     0     0     0
 Stamp: 2002-11-17-09.55.31.956231  Pg grp: 0      NoShare  NoICVI  NoPerm
EMPDEMO.ORG-DEMO-REGION        Ofl      75151      75200    0     0     0     0
 Stamp: 2002-11-17-09.55.31.887739  Pg grp: 0      NoShare  NoICVI  NoPerm
===► Output from group member SYSTEM73
---------- Area -----------   Lock    Lo-Page    Hi-Page #Ret  #Upd #Tret #Ntfy
EMPDEMO.EMP-DEMO-REGION        Ofl      75001      75100    0     0     0     0
 Stamp: 1001-08-07-14.58.14.855461  Pg grp: 0      NoShare  NoICVI  NoPerm
EMPDEMO.INS-DEMO-REGION        Ofl      75101      75150    0     0     0     0
 Stamp: 1001-08-07-14.58.14.896650  Pg grp: 0      NoShare  NoICVI  NoPerm
EMPDEMO.ORG-DEMO-REGION        Ofl      75151      75200    0     0     0     0
 Stamp: 1001-08-07-14.58.14.874287  Pg grp: 0      NoShare  NoICVI  NoPerm
```

A.32 Command Codes

The following command codes apply to new and revised DCMT commands available in Release 16.0.

Code	DCMT Command
N009	VARY AREA
N009016	DEALLOCATE
N009022	ALLOCATE
N025	VARY PROGRAM
N025013	VARY PROGRAM MULTIPLE ENCLAVE ON
N025014	VARY PROGRAM MULTIPLE ENCLAVE OFF
N025015	VARY PROGRAM DEFINE *keyword*
N025016	VARY PROGRAM DEFINE LANGUAGE
N025017	VARY PROGRAM DEFINE ISA SIZE
N025018	VARY PROGRAM DEFINE TYPE
N025019	VARY PROGRAM DEFINE MPMODE
N064	DISPLAY or VARY DISTRIBUTED
N064001	DISPLAY DISTRIBUTED TRANSACTION
N064002	DISPLAY DISTRIBUTED TRANSACTION ID/XID
N064006	DISPLAY DISTRIBUTED RESOURCE MANAGER
N064007	DISPLAY DISTRIBUTED RESOURCE MANAGER *rm-name*
N064010	VARY DISTRIBUTED TRANSACTION ID/XID
N064011	VARY DISTRIBUTED RESOURCE MANAGER *rm-name*
N076	DISPLAY SUBTASK/MT
N076005	DISPLAY SUBTASK EFFECTIVENESS
N077	VARY SUBTASK
N077001	VARY SUBTASK nnn RRS ENABLED
N077002	VARY SUBTASK nnn RRS DISABLED
N086	DISPLAY DEADLOCK
N086002	DISPLAY DEADLOCK DETAILS
N087	VARY DEADLOCK

Code	DCMT Command
N087003	VARY DEADLOCK DETAILS OFF
N087004	VARY DEADLOCK DETAILS ON
N089	VARY DMCL
N089006	VARY DMCL MEMORY CACHE STORAGE LIMIT nnn xB
N089007	VARY DMCL MEMORY CACHE STORAGE LIMIT OPSYS
N089008	VARY DMCL MEMORY CACHE LOCATION 64 BIT ONLY
N089009	VARY DMCL MEMORY CACHE LOCATION ANYWHERE
N091	VARY SEGMENT
N091016	DEALLOCATE
N091022	ALLOCATE
N102	DISPLAY or VARY TRANSACTION SHARING
N102000	DISPLAY TRANSACTION SHARING
N102001	VARY TRANSACTION SHARING
N103	DISPLAY or VARY SYSTRACE or DBTRACE
N103001	DISPLAY SYSTRACE
N103002	VARY SYSTRACE OFF
N103003	VARY SYSTRACE ON
N103004	DISPLAY DBTRACE
N103005	VARY DBTRACE OFF
N103006	VARY DBTRACE ON

See the *Advantage CA-IDMS Security Administration* for information on how to use these command codes to secure these DCMT commands.

Appendix B. New and Revised SQL Statements

B.1 User-Defined SQL Function Statements

This section discusses the new SQL statements available in Release 16.0 that let you define external SQL functions. These functions are listed below:

- ALTER FUNCTION
- CREATE FUNCTION
- DISPLAY/PUNCH FUNCTION
- DROP FUNCTION

This section also explains how to invoke these functions.

B.1.1 Function Invocation

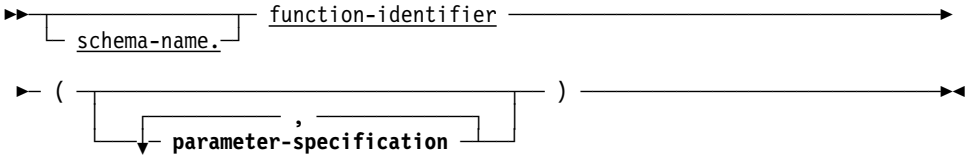
B.1.1.1 Purpose

Represents the invocation of a scalar function through a qualified or unqualified function identifier together with an optional set of parameter values.

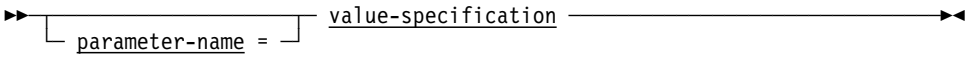
B.1.1.2 Authorization

To invoke a function, you must either own or hold the SELECT privilege on the named function.

B.1.1.3 Syntax



Expansion of parameter-specification:



B.1.1.4 Parameters

schema-name

Specifies the schema with which the function identified by *function-identifier* is associated.

For information on using a schema name to qualify a function, see "Identifying Entities in Schemas" in the *Advantage CA-IDMS Database SQL Option Reference Guide*.

function-identifier
Identifies a function defined in the dictionary.

parameter-specification
Specifies a value to be assigned to a parameter of a function. Both the positional (with NO parameter-name) and the non-positional (with parameter-name) forms of parameter specification can be used in a single function invocation.

If a non-positional parameter specification is used, all remaining parameter specifications in the parameter list MUST be non-positional. Positional parameter specifications are assumed to correspond to the declared parameters of a function in the sequence of their declaration.

parameter-name
Specifies the name of a parameter associated with the function.

value-specification
Any valid expression involving constants, host variables, database columns, and scalar function invocations.

B.1.1.5 Usage

Passing and returning values to a function: During SQL function processing, Advantage CA-IDMS issues a call to the corresponding external routine with the values supplied in the function invocation. Before returning control, the external routine must set a value for the implicitly defined output parameter USER_FUNC; this then becomes the function return value.

For more information about assignment of values to function parameters, see Appendix C, “SQL Functions and SQL Procedure Enhancements.”

CA-supplied versus user-defined functions: If the function invocation contains a schema-name, the target function is the one contained within the schema derived by applying the rules in "Identifying Entities in Schemas" in the *Advantage CA-IDMS Database SQL Option Reference Guide*.

If the function invocation does not contain a schema name, then Advantage CA-IDMS identifies the target function as follows:

- If *function-identifier* matches the identifier of a scalar function distributed with Advantage CA-IDMS, then the target is that function.
- Otherwise, the target function is the function identified by *function-identifier* in the schema derived by applying the rules in "Identifying Entities in Schemas" in the *Advantage CA-IDMS Database SQL Option Reference Guide*.

User-defined function restrictions: You cannot include a user-defined function invocation in the search condition of a table's check constraint.

B.1.1.6 Examples

```
Select emp_id, fin.udf_funbonus(emp_id) from demoempl.employee;

Select power(sqrt(alpha_index), 3)  from prod001.measurement;
```

B.1.2 ALTER FUNCTION Statement

B.1.2.1 Purpose

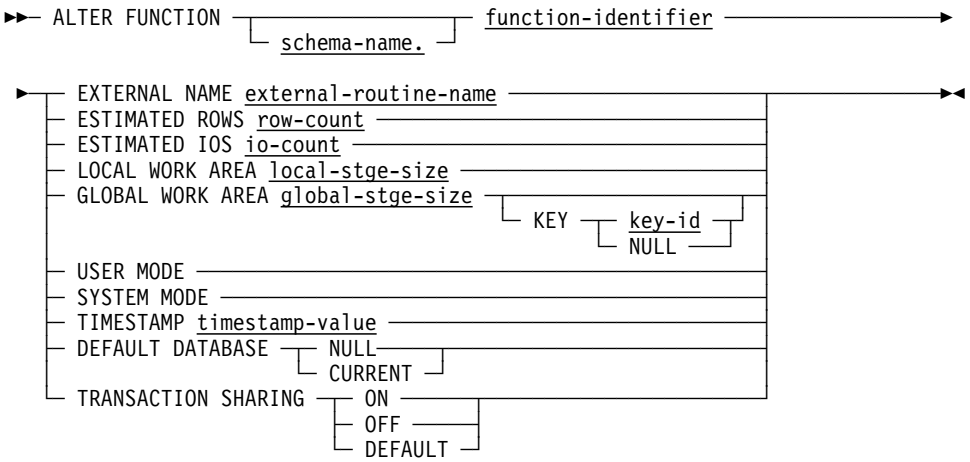
ALTER FUNCTION is a data description statement that modifies the definition of a function in the dictionary. The ALTER FUNCTION statement is an Advantage CA-IDMS extension of ANSI-standard SQL. Using the ALTER FUNCTION statement, you can:

- Revise the estimated row and I/O counts
- Change the external name of the function
- Change the size and characteristics of the work areas passed to the function
- Change the execution mode of the function
- Change the timestamp
- Change the default database
- Change the transaction sharing mode

B.1.2.2 Authorization

To issue an ALTER FUNCTION statement, you must own or hold the ALTER privilege on the function named in the statement.

B.1.2.3 Syntax



B.1.2.4 Parameters

function-identifier

Specifies the name of the function being modified. *function-identifier* must identify a function defined in the dictionary.

schema-name

Identifies the schema associated with the named function. If you do not specify a schema-name, it defaults to:

- The current schema associated with your SQL session if the statement is entered through the Command Facility or executed dynamically
- The schema associated with the access module used at runtime if the statement is embedded in an application program

external-routine-name

Specifies the one- to eight-character name of the program or mapless dialog that Advantage CA-IDMS calls to process function invocations.

row-count

Specifies an integer value, in the range of 0 through 2,147,483,647 that represents the average number of rows that the Advantage CA-IDMS optimizer uses for cost calculation of the function invocation.

io-count

Specifies an integer value, in the range of 0 through 2,147,483,647 that represents the average number of disk accesses that the function generates for a given set of input parameters.

local-stge-size

Specifies an integer value, in the range of 0 through 32,767 that represents the size, in bytes, of a local storage area that Advantage CA-IDMS allocates at runtime and passes to the function on each invocation.

Advantage CA-IDMS allocates a local storage area on each call to a function.

global-stge-size

Specifies an integer value, in the range of 0 through 32,767 that represents the size, in bytes, of a global storage area that Advantage CA-IDMS allocates at runtime and passes to the function on each invocation.

A global storage area is allocated once within a transaction and is retained until the transaction terminates.

key-id

Specifies the one- to four-character identifier for the global storage area. Advantage CA-IDMS passes the same piece of global storage within a transaction to all SQL routines that have the same global storage key.

If you do not specify a storage key, its value remains unchanged if a global storage area was previously associated with the function. To remove a storage key, specify NULL as the key.

USER MODE
Specifies that the function should execute as a user-mode application program within Advantage CA-IDMS. Do not specify USER MODE for functions written as an Advantage CA-ADS mapless dialog.

SYSTEM MODE
Specifies that the function should execute as a system-mode application program. To execute as a SYSTEM MODE application, the program must be written in assembler or COBOL and be fully reentrant or a mapless dialog.

timestamp-value
Specifies the value of the synchronization stamp to be assigned to the function. *timestamp-value* must be a valid external representation of a timestamp.

DEFAULT DATABASE
Specifies whether a default database should be established for database sessions started by the function.

NULL
Specifies that no default database should be established.

CURRENT
Specifies that the database to which the SQL session is connected should become the default for any database session started by the function.

TRANSACTION SHARING
Specifies whether transaction sharing should be enabled for database sessions started by the function. If transaction sharing is enabled for a function's database session, it shares the current SQL session's transaction.

ON
Specifies that transaction sharing should be enabled.

OFF
Specifies that transaction sharing should be disabled.

DEFAULT
Specifies that the transaction sharing setting that is in effect when the function is invoked should be retained.

B.1.2.5 Usage

Specifying a synchronization stamp: When defining or altering a function you can specify a value for the synchronization stamp. If explicitly specified, the synchronization stamp should always be set to a new value following the change so that the change is detectible by the runtime system.

CAUTION:
Care should be exercised when explicitly specifying the synchronization stamp value, since its purpose is to enable the detection of discrepancies between a function and its definition.

If not specified, the synchronization stamp is automatically set to the current date and time.

B.1.2.6 Example

The example below shows the use of ALTER FUNCTION to change the external name of a function.

```
alter function fin.udf_funbonus external name funbon09;
```

B.1.3 CREATE FUNCTION Statement

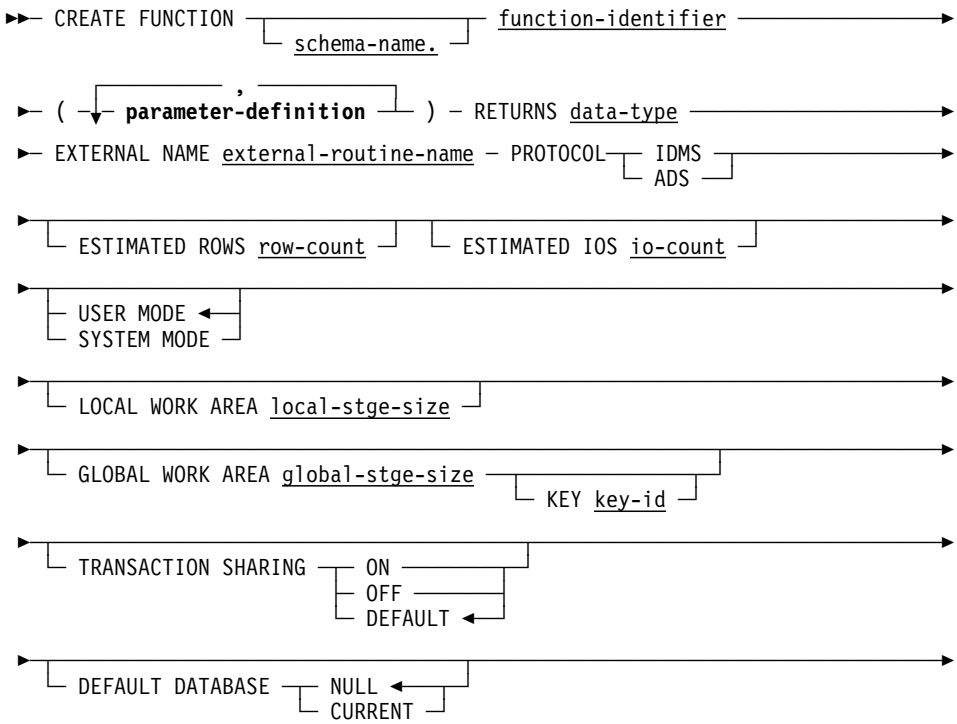
B.1.3.1 Purpose

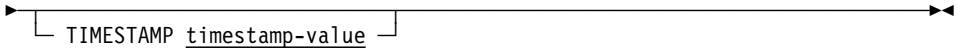
The CREATE FUNCTION statement is a data description statement that stores the definition of a function in the SQL catalog. You can invoke the function in any value-expression of an SQL statement except in the search condition of a table's check constraint. The function invocation results in Advantage CA-IDMS calling the corresponding external routine. Such routines can perform any action and return a single scalar value. You use the formal parameters of a function definition to specify the datatype and format of the data to be passed to the function. Similarly, the datatype of the return value is specified in the function definition.

B.1.3.2 Authorization

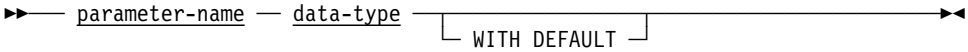
To issue a CREATE FUNCTION statement, you must own the schema in which the function is being defined or hold the CREATE privilege on the named function.

B.1.3.3 Syntax





Expansion of parameter-definition:



B.1.3.4 Parameters

function-identifier

Specifies the 1- to 18-character name of the function that you are creating. Function-identifier must:

- Be unique among the function, procedure, table, table procedure, and view identifiers within the schema associated with the function
- Follow conventions for SQL identifiers

schema-name

Specifies the schema name qualifier to be associated with the function. Schema-name must identify a schema defined in the dictionary. If you do not specify a *schema-name*, it defaults to:

- The current schema associated with your SQL session if the statement is entered through the Command Facility or executed dynamically
- The schema associated with the access module used at runtime if the statement is embedded in an application program

parameter-definition

Defines a parameter to be associated with the function. Parameters are passed to the function in the order in which they are specified. The list of parameters must be enclosed in parentheses. Commas must separate multiple parameter definitions.

Expanded syntax for **parameter-definition** is shown immediately following the CREATE FUNCTION syntax.

RETURNS data-type

Specifies the datatype of the returned value. For more information, see the *Advantage CA-IDMS Database SQL Option Reference Guide*. The returned value is implicitly nullable and can be set to NULL in the external routine. The returned value is accessible to the external routine as an extra parameter with the implicit name USER_FUNC, which comes immediately after the function parameters.

external-routine-name

Specifies the one- to eight-character name of the program or mapless dialog that Advantage CA-IDMS calls to process function invocations.

PROTOCOL

This is a required parameter that specifies the protocol that is used to invoke the function.

IDMS

Use IDMS for SQL functions that are written in COBOL, PL/I, or Assembler and that use the same protocol as in earlier Advantage CA-IDMS releases.

ADS

Use ADS for SQL functions that are written in Advantage CA-ADS. The name of the dialog that is loaded and run when the SQL function is invoked is specified in the *external-routine-name* of the EXTERNAL NAME clause. With the protocol set to ADS, the mode clause must be set to SYSTEM.

row-count

Specifies an integer value, in the range of 0 through 2,147,483,647 that represents the average number of rows that the Advantage CA-IDMS optimizer uses for cost calculation of the function invocation.

io-count

Specifies an integer value, in the range of 0 through 2,147,483,647 that represents the average number of disk accesses that the function generates for a given set of input parameters.

local-stge-size

Specifies an integer value, in the range of 0 through 32,767 that represents the size, in bytes, of a local storage area that Advantage CA-IDMS allocates at runtime and passes to the function on each invocation. Advantage CA-IDMS allocates a local storage area on each call to a function.

Note: If you do not code a LOCAL WORK AREA clause, the default local storage size is 1024 bytes.

global-stge-size

Specifies an integer value, in the range of 0 through 32,767 that represents the size, in bytes, of a global storage area that Advantage CA-IDMS allocates at runtime and passes to the function on each invocation.

A global storage area is allocated once within a transaction and is retained until the transaction terminates.

key-id

Specifies the one- to four-character identifier for the global storage area. Advantage CA-IDMS passes the same piece of global storage within a transaction to all SQL routines that have the same global storage key.

If you do not specify a storage key, Advantage CA-IDMS allocates a unique global storage area for the function.

USER MODE

Specifies that the function should execute as a user-mode application program within Advantage CA-IDMS. Do not specify user mode if the function is a mapless dialog. This is the default.

SYSTEM MODE

Specifies that the function should execute as a system-mode application program. To execute as a system mode application, the program must be written in Assembler or COBOL and be fully reentrant or be a mapless dialog.

If PROTOCOL is set to ADS, you must specify SYSTEM MODE.

timestamp-value
Specifies the value of the synchronization stamp to be assigned to the function. *timestamp-value* must be a valid external representation of a timestamp.

DEFAULT DATABASE
Specifies whether a default database should be established for database sessions started by the function.

NULL
Specifies that no default database should be established.

CURRENT
Specifies that the database to which the SQL session is connected should become the default for any database session started by the function.

TRANSACTION SHARING
Specifies whether transaction sharing should be enabled for database sessions started by the function. If transaction sharing is enabled for a function's database session, it shares the current SQL session's transaction.

ON
Specifies that transaction sharing should be enabled.

OFF
Specifies that transaction sharing should be disabled.

DEFAULT
Specifies that the transaction sharing setting that is in effect when the function is invoked should be retained.

parameter-name
Specifies a 1- to 32-character name of a parameter to be passed to the function. Parameter-name must:

- Be unique within the function that you are defining
- Follow the conventions for SQL identifiers

All parameters are implicitly nullable and thus can be assigned NULL as a parameter value.

data-type
Specifies the datatype of the parameter. For more information, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

WITH DEFAULT
Directs Advantage CA-IDMS to pass a default value for the named parameter if you do not specify a value for the parameter in a function invocation.

The default value for a parameter is based on its data type. For more information, see "CREATE PROCEDURE Statement" in the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.1.3.5 Usage

Specifying a synchronization stamp: When defining or altering a function you can specify a value for the synchronization stamp. If explicitly specified, the synchronization stamp should always be set to a new value following the change so that the change is detectible by the runtime system.

CAUTION:
Care should be exercised when explicitly specifying the synchronization stamp value, since its purpose is to enable the detection of discrepancies between a function and its definition.

If not specified, the synchronization stamp is automatically set to the current date and time.

B.1.3.6 Example

```
CREATE FUNCTION FIN.UDF_FUNBONUS
       ( F_EMP_ID         DECIMAL(4)    )
       RETURNS DECIMAL(10)
       EXTERNAL NAME FUNBONUS PROTOCOL IDMS
       DEFAULT DATABASE CURRENT
       USER MODE
       LOCAL WORK AREA 0 ;
```

B.1.4 DISPLAY/PUNCH FUNCTION Statement

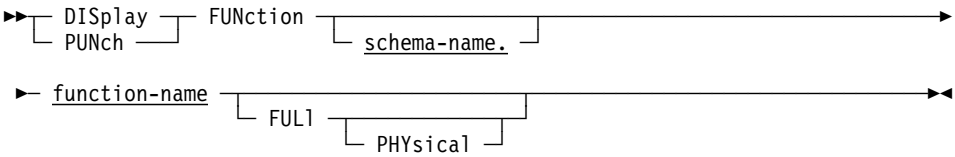
B.1.4.1 Purpose

The DISPLAY/PUNCH FUNCTION statement lets you display or punch a user-defined function definition.

B.1.4.2 Authorization

To issue a DISPLAY/PUNCH FUNCTION statement, you must hold the DISPLAY privilege for the named function.

B.1.4.3 Syntax



B.1.4.4 Parameters

schema-name
Identifies the SQL schema associated with the named function. If you enter the statement through the command facility or execute it dynamically, and if you do not specify *schema-name*, it defaults to the current schema associated with your SQL session.

function-name
Specifies the name of the function to display or punch. *Function-name* must be the name of a function defined in the dictionary.

FULl
Directs Advantage CA-IDMS to display all attributes of the function except physical attributes.

PHYsical
Directs Advantage CA-IDMS to display all attributes of the function including its synchronization timestamp.

WITh
Lists the requested information in addition to the information that is always included, such as the entity occurrence name.

WITHOut
Does *not* list the specified options. *Other* options in effect through the WITH or ALSO WITH clauses in the current DISPLAY statement display.

ALSo WITh
Lists the requested information in addition to the information requested in previously issued DISPLAY WITH and DISPLAY ALSO WITH statements for the named entity.

ALSo WITHOut
Does *not* list the specified options.

ALL
Specifies the display of all the information associated with the requested entity occurrence.

NONe
Specifies the display of the name of the requested entity occurrence. NONE is meaningful only when you specify the WITH clause.

DETails
Specifies the display of entity-specific descriptions, for example, the length of a table.

TIMestamp
Specifies the display of the synchronization timestamp associated with the function.

HIStory
Specifies the display of the chronological account of an entity's existence, including PREPARED/REVISED BY specifications, date created, and date last updated.

KEYs
Specifies the display of all keys associated with the requested function.

AS COMments
Outputs function syntax as comments with the characters *+ preceding the text of the statement. AS COMMENTS is the default.

AS SYNtax
Outputs function syntax that you can edit and resubmit to the command facility.

VERB CREate/DISplay/DROp/PUNch
Specifies the verb with which the entity statement is displayed or punched. For example, if VERB CREATE is specified, the output of the DISPLAY/PUNCH statement is a CREATE statement; if VERB ALTER is specified, the output is an ALTER statement, and so on. The default is VERB CREATE.

B.1.4.5 Example

```
DISPLAY FUNCTION FIN.UDF_FUNBONUS FULL PHYSICAL;
```

B.1.5 DROP FUNCTION

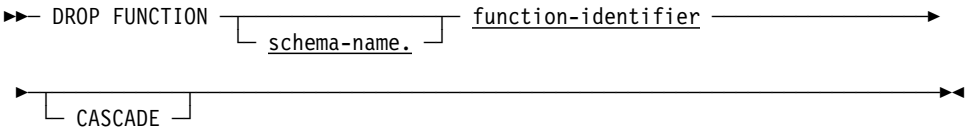
B.1.5.1 Purpose

Deletes the definition of the referenced function from the dictionary. The DROP FUNCTION statement is an Advantage CA-IDMS extension of ANSI-standard SQL.

B.1.5.2 Authorization

To issue a DROP FUNCTION statement, you must own or have the DROP privilege on the function named in the statement.

B.1.5.3 Syntax



B.1.5.4 Parameters

function-identifier

Specifies the name of the function to be dropped. *Function-identifier* must identify a function defined in the dictionary.

schema-name

Identifies the schema associated with the specified function. If you do not specify a *schema-name*, the default value is:

- The current schema associated with your SQL session if the statement is specified through the Command Facility or executed dynamically
- The schema associated with the access module used at runtime if the statement is embedded in an application program

CASCADE

Directs Advantage CA-IDMS to delete any view definition that contains a reference to the function, either directly or nested within another view reference.

B.1.5.5 Example

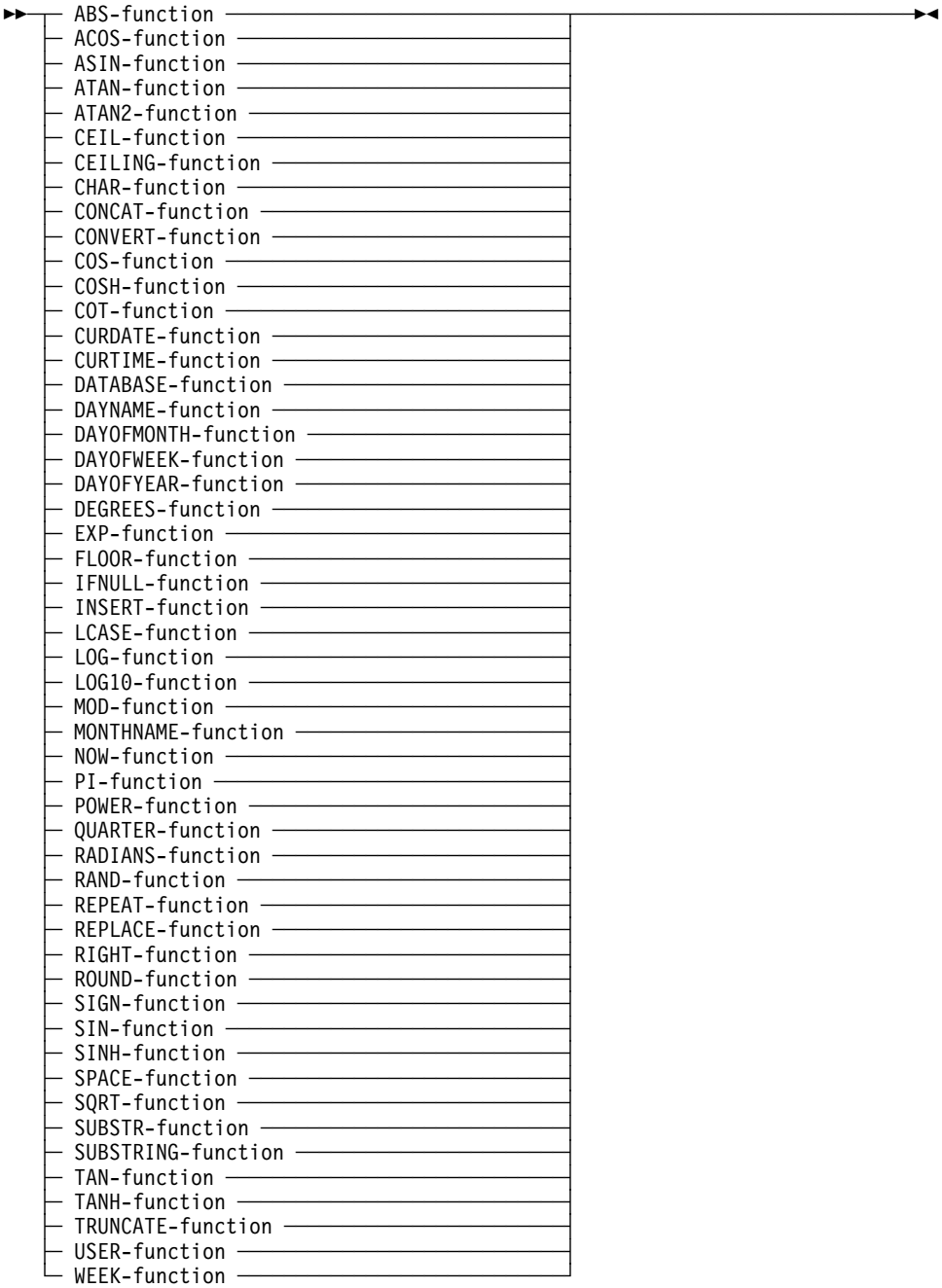
The DROP FUNCTION statement below removes the FIN.UDF_FUNBONUS function from the SQL catalog and drops any view within which the function is invoked.

```
DROP FUNCTION FIN.UDF_FUNBONUS CASCADE;
```

B.2 SQL Scalar Functions

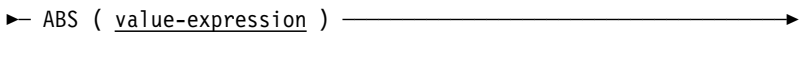
This section provides details about the new SQL scalar functions available in Release 16.0. It also describes the enhancements that have been made to the existing CHAR function.

B.2.1.1 Syntax



Note: All of the previous functions , except LCASE are implemented as user-defined functions in schema SYSCA.

B.2.1.2 ABS-function



ABS returns the absolute value of the *value-expression*.

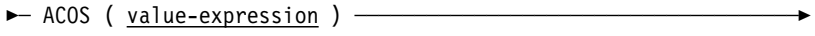
value-expression
A value with a numeric data type.

The result has the same data type as the *value-expression*. If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns 125.

```
SELECT ABS(-125)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.3 ACOS-function



ACOS returns the arccosine of the *value-expression* as an angle expressed in radians. ACOS is the inverse function of the COS function.

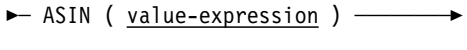
value-expression
Must be a numeric data type and must have a value in the range of -1 to 1. It is converted to a double precision floating-point number for processing by this function.

The result of the function is a double precision floating-point number. If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns 7.9539883018414370E-01:

```
SELECT ACOS(0.7)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.4 ASIN-function



ASIN returns the arcsine of the *value-expression* as an angle expressed in radians ASIN is the inverse function of the SIN function.

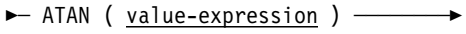
value-expression
The *value-expression* must be of any numeric type and must have a value in the range of -1 to 1. It is converted to a double precision floating-point number for processing by this function.

The result of the function is a double precision floating-point number. If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns 1.5707963267948966E+00:

```
SELECT ASIN(1)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.5 ATAN-function



ATAN returns the arctangent of the *value-expression* as an angle expressed in radians. ATAN is the inverse function of the TAN function.

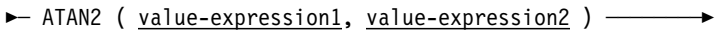
value-expression
Must be of any numeric data type. It is converted to a double precision floating-point number for processing by this function.

The result of the function is a double precision floating-point number. If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns 1.2490457723982544E+00

```
SELECT ATAN(3)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.6 ATAN2-function



ATAN2 returns the arctangent of x and y coordinates, given by *value-expression1* and *value-expression2* respectively, as an angle expressed in radians.

value-expression1
Specifies a numeric *value-expression*.

value-expression2
Specifies a numeric *value-expression*.

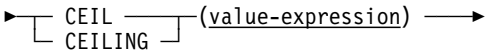
Both *value-expressions* must be of any numeric data type and cannot both be 0. They are converted to double precision floating-point numbers for processing by this function.

The result of the function is a double precision floating-point number. If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns 1.2490457723982544E+00

```
SELECT ATAN2(1,3)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.7 CEIL or CEILING-function



CEILING returns the smallest integer value that is greater than or equal to the *value-expression*. CEIL and CEILING are identical.

value-expression

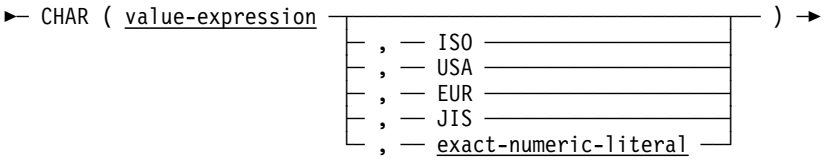
Must be a numeric data type.

The result of the function has the same data type as the *value-expression* except that the scale is 0 if the *value-expression* is of type (UNSIGNED) DECIMAL or (UNSIGNED) NUMERIC. For example, a *value-expression* with a data type of NUMERIC (3,2) results in NUMERIC (3,0). If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns: 13, 2.0000000000000000E+00, -12

```
SELECT CEILING(12.55), CEILING(123.1E-2), CEILING (-12.55)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.8 CHAR-function



CHAR obtains a character string representation from the value in *value-expression*. The syntax and semantics for the CHAR function depends on the data type of *value-expression*.

value-expression

- Data type of value-expression is an exact numeric data: INTEGER, SMALLINT, or LONGINT.

  CHAR returns a fixed-length character string representation of the exact numeric value of *value-expression*. Specifying a second parameter is not allowed. The result is left-justified and contains *n* characters corresponding to the digits of the value of *value-expression* with a preceding minus sign if the *value-expression* is negative. The length of the returned string depends on the data type of *value-expression*:

  – SMALLINT — result length of 6

  – INTEGER — result length of 11

  – LONGINT — a result length of 20.

Example

```
SELECT CHAR(FIXLENGTH), LENGTH(CHAR(FIXLENGTH)) AS LEN_SMALLINT ,
CHAR(NUMROWS) , LENGTH(CHAR(NUMROWS)) AS LEN_INTEGER
    FROM SYSTEM.TABLE WHERE NAME = 'TABLE';

*+ CHAR(FUNCTION)  LEN_SMALLINT  CHAR(FUNCTION)  LEN_INTEGER
*+ --------------  ------------  --------------  -----------
*+ 256                        6  33                       11
*+ 0                          6  0                        11
```

- Data type of *value-expression* is a fixed point, packed or zoned decimal: (UNSIGNED) DECIMAL, (UNSIGNED) NUMERIC.

  CHAR returns a fixed-length character string representation of the value of *value-expression*. Specifying a second parameter is not allowed. If *value-expression* has a precision of *p* and a scale of *s*, the result contains *p+2* characters as follows: a blank or minus sign, depending on the sign of *value-expression*, *p-s* digits followed by a period and finally *s* digits. The result is left-justified.

  Example

```
SELECT VAC_TIME, CHAR(-VAC_TIME), LENGTH(CHAR(VAC_TIME))
       FROM DEMOEMPL.EMP_VACATION WHERE VAC_TIME > 300

*+                VAC_TIME  CHAR(FUNCTION)           (CONST)
*+                --------  --------------           -------
*+                  340.00  -340.0                         33
*+                  396.00  -396.0                         33
*+                  484.00  -484.0                         33
```

- Data type of *value-expression* is a floating-point data type: REAL, FLOAT or DOUBLE PRECISION

  CHAR returns a fixed-length character string representation of the floating point value of *value-expression* Specifying a second parameter is not allowed. The result is left-justified and contains 24 characters.

  Example

```
SELECT AVGROWLENGTH, CHAR(AVGROWLENGTH), LENGTH(CHAR(AVGROWLENGTH)) AS L24
       FROM SYSTEM.TABLE WHERE NAME = 'TABLE';

*+   AVGROWLENGTH  CHAR(FUNCTION)              L24
*+   ------------  --------------              ---
*+   2.5600000E+02  2.56E2                       24
*+   0.0000000E+00  0.0E0                        24
```

- Data type of *value-expression* is a character data type CHAR, VARCHAR.

  CHAR returns a fixed-length character string representation of the value of *value-expression* An *exact-numeric-literal* can be specified as a second parameter, in which case it defines the length of the result. The value of *exact-numeric-literal* must be in the range 0-255. When the:

  – Length of *value-expression* is lower than *exact-numeric-literal* — the result is padded with blanks on the right

  – Length of *value-expression* is larger — truncation occurs, when nonblank characters are truncated, a warning message is issued.

Example

```
SELECT CHAR(NAME,4), LENGTH(CHAR(NAME, 4)) AS LEN
        FROM SYSTEM.TABLE WHERE NAME = 'TABLE';

*+ DB001043 T375 C1M322: String truncation
*+ DB001043 T375 C1M322: String truncation
*+ CHAR(FUNCTION)     LEN
*+ --------------     ---
*+ TABL                 4
*+ TABL                 4
```

- Data type of *value-expression* is DATE, TIME, or TIMESTAMP.

If a format (ISO, USA, EUR, JIS) is not specified for the character string, the result is returned in ISO format or, if the SQL statement is embedded in a program, the format specified in the precompiler options.

►► Refer to the *Advantage CA-IDMS Database SQL Option Programming Guide* for information about specifying precompiler options.

ISO

Specifies that the format of the result should comply with the standard of the International Standards Organization (ISO). Use the following formats when ISO is specified:

Data type	Format	Example
DATE	yyyy-mm-dd	1990-12-15
TIME	hh.mm.ss	16.43.17
TIMESTAMP	yyyy-mm-dd-hh.mm.ss.nnnnnn	1990-12-15-16.43.17.123456

USA

Specifies that the format of the result should comply with the standard of the IBM USA standard. Use the following formats when USA is specified:

Data type	Format	Example
DATE	mm/dd/yyyy	12/15/1990
TIME	hh:mm AM hh:mm PM	4:43 PM
TIMESTAMP	yyyy-mm-dd-hh.mm.ss.nnnnnn	1990-12-15-16.43.17.123456

EUR

Specifies that the format of the result should comply with the standard of the IBM European standard. Use the following formats when EUR is specified:

Data type	Format	Example
DATE	dd.mm.yyyy	15.12.1990

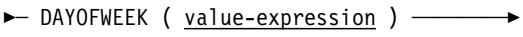
Data type	Format	Example
TIME	hh.mm.ss	16.43.17
TIMESTAMP	yyyy-mm-dd-hh.mm.ss.nnnnnn	1990-12-15-16.43.17.123456

JIS

Specifies that the format of the result should comply with the standard of the Japanese Industrial Standard Christian Era. Use the following formats when JIS is specified:

Data type	Format	Example
DATE	yyyy-mm-dd	1990-12-15
TIME	hh:mm:ss	16:43:17
TIMESTAMP	yyyy-mm-dd-hh.mm.ss.nnnnnn	1990-12-15-16.43.17.123456

B.2.1.9 DAYOFWEEK-function



DAYOFWEEK returns the day of the week where 1 is Sunday and 7 is Saturday.

value-expression

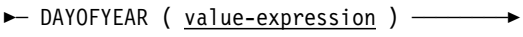
Must be a DATE or TIMESTAMP data type or must be a CHARACTER or VARCHAR data type and represent a valid string representation of a date or timestamp.

The result is an INTEGER data type. The result is null if *value-expression* is null.

Example: The following statement returns 4, which represents Wednesday:

```
SELECT DAYOFWEEK ('2002-12-25')
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.10 DAYOFYEAR-function



DAYOFYEAR returns the day of the year where 1 is January 1.

value-expression

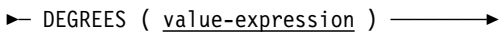
Must be a DATE or TIMESTAMP data type or must be a CHARACTER or VARCHAR data type and represent a valid string representation of a date or timestamp.

The result is an INTEGER data type and in the range of 1 to 366. The result is null if *value-expression* is null.

Example: The following statement returns 365:

```
SELECT DAYOFYEAR ('2002-12-31')
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.11 DEGREES-function



DEGREES returns the number of degrees calculated from the *value-expression* expressed in radians.

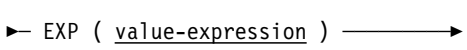
value-expression
Must be of any numeric data type. It is converted to a double precision floating-point number for processing by this function.

The result of the function is a double precision floating-point number. If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns 8.9999999999999985E+01:

```
SELECT DEGREES(PI() / 2)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.12 EXP-function



EXP returns a value that is calculated as the base of the natural logarithm (e), raised to a power specified by the *value-expression*. EXP is the inverse function of LOG.

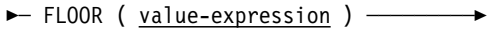
value-expression
Must be a numeric data type. It is converted to a double precision floating-point number for processing by this function.

The result of the function is a double precision floating-point number. If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns 2.7182818284590451E+00:

```
SELECT EXP (1)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.13 FLOOR-function



FLOOR returns the largest integer value that is less than or equal to the *value-expression*.

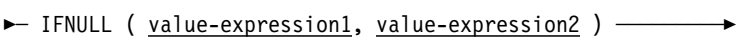
value-expression
Must be a numeric data type.

The result of the function has the same data type as the *value-expression* except that the scale is 0 if the *value-expression* is of type (UNSIGNED) DECIMAL or (UNSIGNED) NUMERIC. For example, a *value-expression* with a data type of NUMERIC (3,2) results in NUMERIC(3,0). If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns: 12, 1.0000000000000000E+00, -13

```
SELECT FLOOR  (12.55), FLOOR  (123.1E-2), FLOOR   (-12.55)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.14 IFNULL-function



IFNULL returns the first *value-expression* that is not null. IFNULL is similar to the VALUE and COALESCE scalar functions with the exception that IFNULL is limited to only two *value-expressions* instead of multiple *value-expressions*.

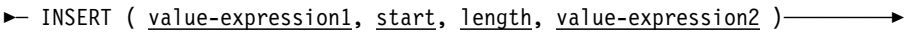
value-expression1
Specifies a *value-expression*

value-expression2
Specifies a *value-expression.*

Example: The following statement shows '**NULL**' for any row with a null value for SEGMENT, otherwise the name of the segment is shown:

```
SELECT SCHEMA, NAME, IFNULL (SEGMENT, '**NULL**')
       FROM SYSTEM.TABLE
```

B.2.1.15 INSERT-function



INSERT returns a string constructed from *value-expression1*, where beginning at *start*, *length* characters are deleted and *value-expression2* is inserted.

value-expression1
Specifies a character string *value-expression*. *value-expression1* specifies the source string and must be a CHARACTER or VARCHAR data type. If the length of *value-expression1* is 0, the result is a null value.

value-expression2
Specifies a character string *value-expression.*

value-expression2 specifies the string to be inserted into *value-expression1*, starting at *start*. The string to be inserted must be a CHARACTER or VARCHAR data type.

start

Specifies a numeric *value-expression*. *Start* must be of any numeric data type, but only the integer part is considered. The integer part of *start* specifies the starting point within *value-expression1* where the deletion of characters and the insertion of *value-expression2* begins. The integer part of *start* must be in the range of 1 to the length of *value-expression1* plus one.

length

Specifies a numeric *value-expression*. *Length* must be of any numeric data type, but only the integer part is considered. The integer part of *length* specifies the number of characters that are to be deleted from *value-expression1*, starting at start. The integer part of *length* must be in the range of 0 to the length of *value-expression1*.

The result is always of VARCHAR data type. The length of the result is given by the following formula:

```
LENGTH(value-expression1) + LENGTH(value-expression2) -
min(length, LENGTH(value-expression1) - start + 1)
```

If both start and length are constants, the maximum length of the result is calculated during compilation of the INSERT invocation using the above formula, otherwise the maximum length is 8000.

The result is null if *value-expression1* or *value-expression2* is null. If the insert cannot be done because of invalid parameters, an exception is raised.

Example 1: The following statement appends the string 'DEF' to the string 'ABC' giving 'ABCDEF':

```
SELECT SUBSTR(INSERT ('ABC', 4  , 0,'DEF')
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

Because the start and length parameters of the INSERT function are constants, the maximum length of the VARCHAR string is 6.

Example 2: The following statement prefixes the string 'DEF' with the string 'ABC' giving 'ABCDEF':

```
SELECT SUBSTR(INSERT ('DEF', 1 *1, 0,'ABC'), 1, 20)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

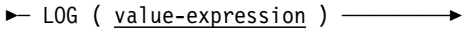
Because the start position is not a constant, but an expression, the maximum length of the VARCHAR string is 8000. The SUBSTR function is used to limit the result to 20 characters.

Example 3: The following statement replaces the character at position 3 in string 'ABCDEF' with the string 'XYZ' returning 'ABXYZDEF':

```
SELECT INSERT ('ABCDEF', 3  , 1,'XYZ')
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

Because both the start and length parameters of the INSERT function are constants, the maximum length of the result VARCHAR string is 8.

B.2.1.16 LOG-function



LOG returns a value that is calculated as the natural logarithm of *value-expression*. LOG is the inverse function of EXP.

value-expression

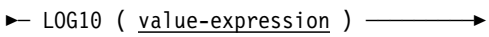
Must be of any numeric data type. It is converted to a double precision floating-point number for processing by this function.

The result of the function is a double precision floating-point number. If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns 1.0986122886681095E+00:

```
SELECT LOG (3)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM'
```

B.2.1.17 LOG10-function



LOG10 returns a value that is calculated as the base 10 logarithm of *value-expression*.

value-expression

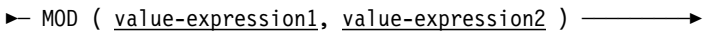
The *value-expression* must be of any numeric data type. It is converted to a double precision floating-point number for processing by this function.

The result of the function is a double precision floating-point number. If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns 3.0000000000000000E+00:

```
SELECT LOG (1000)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM'
```

B.2.1.18 MOD-function



MOD returns the remainder of dividing *value-expression1* by *value-expression2* using the formula:

```
MOD(v1, v2) = v1 - Truncated_Integer(v1/v2) * v2
```

with *Truncated_Integer*(v1 / v2) the truncated integer result of the division.

value-expression1
Specifies a numeric *value-expression* and must be of any numeric data type.

value-expression2
Specifies a numeric *value-expression* and must be of any numeric data type. *value-expression2* cannot be zero.

If any of the *value-expressions* are null the result is a null value. If a data error occurs, an exception is raised.

The data type of the result follows these rules:

- Both *value-expressions* are INTEGER or SMALLINT — the result is INTEGER.
- One of the *value-expressions* is LONGINT and the other is INTEGER, SMALLINT, or LONGINT — the result is LONGINT.
- One *value-expression* is an INTEGER, SMALLINT, or LONGINT and the other is an (UNSIGNED) DECIMAL or (UNSIGNED) NUMERIC — the result is (UNSIGNED) DECIMAL or (UNSIGNED) NUMERIC with the same precision and scale as the (UNSIGNED) DECIMAL or (UNSIGNED) NUMERIC *value-expression*.
- Both *value-expressions* are (UNSIGNED) DECIMAL or (UNSIGNED) NUMERIC — the result is equal to the data type of *value-expression1*. The precision and scale of the result are given by the following formulas:

  Prec. result = min(prec.1-scale.1, prec.2-scale.2) + max(scale.1, scale.2)
  Scale.result = max(scale1, scale2)

- Either *value-expression* is a floating-point number, REAL, FLOAT, or DOUBLE PRECISION — the result is double precision floating-point.

The processing of this function is always done in floating-point. Both *value-expressions* are converted to double precision floating-point numbers.

Example 1: The following statement returns 1:

```
SELECT MOD(10, 3    )
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

Example 2: The following statement returns 1.0000000000000000E+00:

```
SELECT MOD(10E0, 3    )
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

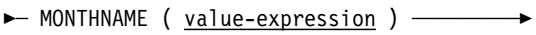
Example 3: The following statement returns 1.0:

```
SELECT MOD(10.0, 3    )
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

Example 4: The following statement returns 1.00:

```
SELECT MOD(10.00  , 3    )
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.19 MONTHNAME-function



MONTHNAME returns a character string containing the English name of the month specified by *value-expression*.

value-expression

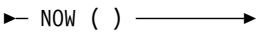
Must be a DATE or TIMESTAMP data type or must be a CHARACTER or VARCHAR data type and represent a valid string representation of a date or timestamp.

The result is of CHARACTER(12) data type. The result is null if *value-expression* is null.

Example: The following statement returns the names of all months from now to now + 11 months: January, February, March, April, May, June, July, August, September, October, November, December.

```
SELECT MONTHNAME(NOW() + 0  MONTH),
       MONTHNAME(NOW() + 1  MONTH),   MONTHNAME(NOW() + 2  MONTH),
       MONTHNAME(NOW() + 3  MONTH),   MONTHNAME(NOW() + 4  MONTH),
       MONTHNAME(NOW() + 5  MONTH),   MONTHNAME(NOW() + 6  MONTH),
       MONTHNAME(NOW() + 7  MONTH),   MONTHNAME(NOW() + 8  MONTH),
       MONTHNAME(NOW() + 9  MONTH),   MONTHNAME(NOW() + 10 MONTH),
       MONTHNAME(NOW() + 11 MONTH),   MONTHNAME(NOW() + 12 MONTH)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM'
```

B.2.1.20 NOW-function

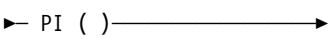


NOW is equivalent to the special-register CURRENT TIMESTAMP. For more information, see "Expansion of Special-register" in the *Advantage CA-IDMS Database SQL Option Reference Guide*.

Example: The following statement returns the current date and time twice:

```
SELECT NOW(), CURRENT TIMESTAMP
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.21 PI-function

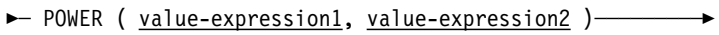


PI returns the constant value of pi as a floating point value. The value returned is 3.141592653589793238.

Example: The following statement returns 3.1415926535897933E+00:

```
SELECT PI()
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.22 POWER-function



POWER returns the value of *value-expression1* to the power of *value-expression2*.

value-expression1
Specifies a numeric *value-expression* and must be of any numeric data type.

value-expression2
Specifies a numeric *value-expression* and must be of any numeric data type.

The internal processing of this function is done using double precision floating-point arithmetic.

The data type of the result of the function depends on the data types of *value-expression1* and *value-expression2*. The result is:

- INTEGER — when *value-expression1* and *value-expression2* are SMALLINT or INTEGER
- LONGINT — when one of the *value-expressions* is LONGINT and the other LONGINT, INTEGER or SMALLINT,
- DOUBLE PRECISION — all other cases

If *value-expression1* or *value-expression2* is null the result is a null value. If a data error occurs a data exception is raised.

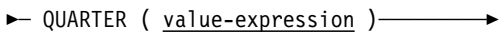
Example 1: The following statement returns the value 625:

```
SELECT POWER(25,2)
       FROM SYSTEM.TABLE WHERE NAME = 'SCHEMA'
```

Example 2: The following statement returns the value 625: The following SELECT returns the value 6.2500000000000000E+02:

```
SELECT POWER(25.0,2)
       FROM SYSTEM.TABLE WHERE NAME = 'SCHEMA'
```

B.2.1.23 QUARTER-function



QUARTER returns the quarter of the year in which the date, specified by *value-expression*, occurs.

value-expression
Must be a DATE or TIMESTAMP data type or must be a CHARACTER or VARCHAR data type and represent a valid string representation of a date or timestamp.

The result is an INTEGER data type and is in the range of 1 to 4. The result is null if *value-expression* is null.

Example: The following statement returns 4 because December is in the last quarter of the year:

```
SELECT QUARTER('2002-12-31')
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM'
```

B.2.1.24 RADIANS-function

RADIANS returns the number of radians corresponding to the number of degrees specified by *value-expression*.

value-expression

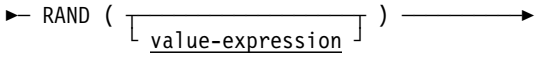
The *value-expression* must be of any numeric data type. It is converted to a double precision floating-point number for processing by this function.

The result of the function is a double precision floating-point number. If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns 3.1415926535897931E+00, which is an approximate value of PI:

```
SELECT RADIANS(180)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.25 RAND-function



RAND returns a random floating-point value between 0 and 1. *value-expression* is optional and specifies a seed value. If no seed value is specified, 1 is used as seed value.

value-expression

If specified, the *value-expression* must be of any numeric data type. It is converted to an INTEGER number for processing by this function.

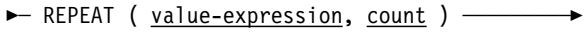
The result of the function is a double precision floating-point number. If a data exception occurs an exception is raised.

Within the context of an Advantage CA-IDMS task, the optional seed value is only evaluated once during the very first call of the random generator with a seed value. The series of generated random numbers is equal for equal seed values when executed under different Advantage CA-IDMS tasks.

Example: The following statement returns random floating-point numbers between 0 and 1:

```
SELECT RAND (200), RAND()
       FROM SYSTEM.SCHEMA;
```

B.2.1.26 REPEAT-function



REPEAT returns a string constructed as count times *value-expression* repeated.

value-expression
Specifies the string to be repeated and must be a CHARACTER or CHAR data type.

count
An expression of any numeric data type, but only the integer part is considered. The integer part of *count* specifies the number of times to repeat *value-expression*.

The result of the function is VARCHAR. The length of the result is the length of *value-expression* multiplied by *count*. If the actual length of the result string exceeds the maximum for the return type, an error occurs. If *count* is a constant, the maximum length of the result is calculated during compilation of the REPEAT function invocation, otherwise the maximum is 16000. The result is null if *value-expression* or *count* is null. If the insert cannot be done an exception is raised.

Example 1: The following statement returns 'ABCDABCDABCDABCD':

```
SELECT SUBSTR(REPEAT('ABCD', 4)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

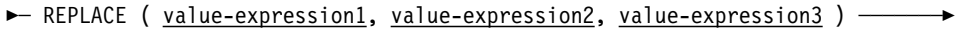
Example 2: The following statement returns a string with length 0:

```
SELECT REPEAT('ABCD', 0)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

Example 3: The following statement returns <null> because count is negative:

```
SELECT REPEAT('ABCD', -2)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.27 REPLACE-function



REPLACE substitutes all occurrences of *value-expression2* in *value-expression1* with *value-expression3*. If *value-expression2* was not found in *value-expression1*, *value-expression1* is returned unchanged.

value-expression1
Specifies a character string *value-expression*. *value-expression1* is a non-null expression that specifies the source string.

value-expression2
Specifies a character string *value-expression*. *value-expression2* is a non-null expression that specifies the string to be replaced in the source string.

value-expression3

Specifies a character string *value-expression*. *value-expression3* is an expression that specifies the replacement string. A null value causes *value-expression1* to be returned unchanged.

The arguments must all have data types that are compatible with VARCHAR, that is CHARACTER or VARCHAR. The actual length of each string must be less than or equal to 8000. The data type of the result is VARCHAR and the resulting length must be less than or equal to 8000. The length of the result is given by the following formula, where n is the number of occurrences of *value-expression2* in *value-expression1*:

```
LENGTH(value-expression1) + (n * (LENGTH(value-expression3)
- LENGTH(value-expression2)))
```

The result is null if *value-expression1*, *value-expression2*, or *value-expression3* is null. If the replace cannot be done an exception is raised.

Example 1: Replace all characters '*' in the string '**123.0**99' with '$$'. In this example, the result is '$$$$123.0$$$$99'.

```
SELECT REPLACE('**123.0**99', '*', '$$')
       FROM SYSTEM.SCHEMA WHERE NAME ='SYSTEM'
```

Example 2: List the departments of the EMPSCHM.DEPARTMENT table in alphabetical order, but ignore any spaces when sorting. The REPLACE function removes all spaces in the SORT_NAME column of the result.

```
SELECT *, REPLACE
(DEPT_NAME_0410, ' ', '') SORT_NAME
       FROM EMPSCHM.DEPARTMENT
       ORDER BY SORT_NAME;
```

Example 3: Replace string 'FOO' in the string 'LOTS OF FOOLISH TALK' with '**FOO**'.

```
SELECT REPLACE('LOTS OF FOOLISH TALK', 'FOO', '**FOO**')
       FROM SYSTEM.SCHEMA WHERE NAME ='SYSTEM'
```

B.2.1.28 RIGHT-function

RIGHT returns a string constructed from the specified number of rightmost *count* characters of *value-expression*.

value-expression

Specifies the string from which the result is constructed and must be a CHARACTER or VARCHAR data type.

count

Any numeric data type, but only the integer part is considered. The integer part of *count* specifies the length of the result. The integer part of *count* must be an integer between 0 and *n*, where *n* is the length of *value-expression*.

The result is null if *value-expression* or *count* is null. If *count* is larger than the length of *value-expression* an exception is raised.

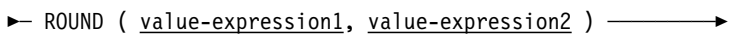
Example 1: The following statement returns the string 'CD':

```
SELECT RIGHT ('ABCD', 2)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

Example 2: The following statement returns a string with length 0:

```
SELECT RIGHT ('ABCD', 0)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.29 ROUND-function



ROUND returns *value-expression1* rounded to *value-expression2* places to the right of the decimal point if *value-expression2* is positive, or to the left of the decimal point if *value-expression2* is zero or negative.

value-expression1
Specifies a numeric *value-expression* and must be of any numeric data type.

value-expression2
Specifies a numeric *value-expression* and must be of any numeric data type. The *value-expression2* must be of any numeric data type but is converted internally to INTEGER.

The integer value of *value-expression2* specifies the number of places to the right of the decimal point for the result if *value-expression2* is not negative. If *value-expression2* is negative, *value-expression1* is rounded to 1 + the absolute integer value of *value-expression2* number of places to the left of the decimal point. If the absolute integer value of *value-expression2* is larger than the number of digits to the left of the decimal point, the result is 0.

If *value-expression1* is positive, rounding is to the next higher positive number. If *value-expression1* is negative, rounding is to the next lower negative number.

The result of the function has the same data type and attributes the *value-expression1* except that the precision is increased by one if *value-expression1* is of (UNSIGNED) DECIMAL or (UNSIGNED) NUMERIC data type and the precision is less than 31. If any of the *value-expressions* are null, the result is a null value. If a data error occurs an exception is raised.

Example 1:

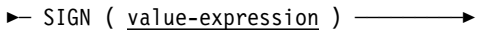
The following statement returns: 627.46380. 627.46400, 627.46000, 50000, 627.00000, 630.00000, 600.00000, 1000.00000, 0.00000:

```
SELECT  ROUND(627.46381, 4) ,                ROUND(627.46381, 3) ,
       ROUND(627.46381, 2) ,                   ROUND(627.46381, 1) ,
       ROUND(627.46381, 0) ,                   ROUND(627.46381,-1) ,
       ROUND(627.46381,-2) ,                   ROUND(627.46381,-3) ,
       ROUND(627.46381,-4)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

Example 2: The following statement returns:-627.46380, -627.46400, -627.46000, -627.50000, -627.00000, -630.00000, -600.00000

```
SELECT  ROUND(-627.46381, 4) ,               ROUND(-627.46381, 3) ,
       ROUND(-627.46381, 2) ,                  ROUND(-627.46381, 1) ,
       ROUND(-627.46381, 0) ,                  ROUND(-627.46381,-1) ,
       ROUND(-627.46381,-2)          FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.30 SIGN-function



SIGN returns an indicator of the sign of *value-expression*. The possible values for the indicator are:

- -1 if *value-expression* is less than zero
- 0 if *value-expression* is zero
- 1 if *value-expression* is greater than zero

value-expression

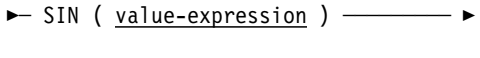
Must be of any numeric data type except (UNSIGNED) DECIMAL or (UNSIGNED) NUMERIC with a scale and precision of 31. The data type and attributes of the result of the function are the same as the *value-expression* except when the *value-expression* is (UNSIGNED) DECIMAL or (UNSIGNED) NUMERIC. The precision is incremented if the *value-expression*'s precision and scale are equal. This is to allow for the return values of the function.

If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns:-1, 0, 1:

```
SELECT SIGN (1 - 10), SIGN ( 0), SIGN (1 +10)
       FROM SYSTEM.TABLE WHERE NAME = 'SYSTEM' ;
```

B.2.1.31 SIN-function



SIN returns the sine of the *value-expression*, which must be an angle expressed in radians. SIN is the inverse function of the ASIN function.

value-expression

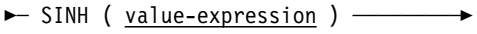
Must be of any numeric data type. It is converted to a double precision floating-point number for processing by this function.

The result of the function is a double precision floating-point number. If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns 1.0000000000000000E+00:

```
SELECT  SIN( PI() / 2)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.32 SINH-function



SINH returns the hyperbolic sine of the *value-expression*, which must be an angle expressed in radians.

value-expression

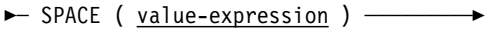
Must be of any numeric data type. It is converted to a double precision floating-point number for processing by this function.

The result of the function is a double precision floating-point number. If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns 1.1548739357257750E+01:

```
SELECT  SIN( PI())
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.33 SPACE-function



SPACE returns a character string that consists of *value-expression* number of blanks.

value-expression

Any numeric data type, but only the integer part is considered. The integer part specifies the number of blanks that makes up the result, and it must be between 0 and 30000.

The result is of VARCHAR data type. The length of the result is the integer part of *value-expression*.

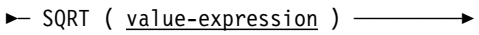
If *.value-expression*. is a constant, the maximum length of the result is calculated during compilation of the SPACE function invocation, otherwise the maximum is 30000.

The result is null if *value-expression* is null. An error occurs if value-expression is larger than 30000.

Example: The following statement returns 10 blanks:

```
SELECT SPACE (10)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.34 SQRT-function



SQRT returns the square root of the *value-expression*.

value-expression
Must be of any numeric data type. It is converted to a double precision floating-point number for processing by this function.

The result of the function is a double precision floating-point number. If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

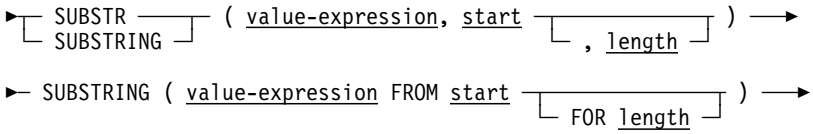
Example 1: The following statement returns 4.0000000000000000E+00:

```
SELECT  SQRT(16)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

Example 2: The following statement returns <null> because the square root of a negative number does not exist:

```
SELECT  SQRT(-16)
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.35 SUBSTR or SUBSTRING-function



SUBSTR or SUBSTRING obtains a substring of the value in *value-expression*. In 16.0 SUBSTRING allows the same syntax as SUBSTR.

value-expression
Must be a character or graphics string.

start
Specifies the position of the first character of the result. *Start* is a value expression that must be an integer less than or equal to the length of the string in *value-expression*. If *start* is null, the result of the function is null.

length
Specifies the length of the result. *Length* is a value expression that must be an integer not less than one. The sum of *length* and *start* must not exceed 1 + the length of the string in *value-expression*. (The length of a value with a data type of VARCHAR or VARGRAPHIC is its maximum length.) When:

- The substring is less than the specified *length* — Advantage CA-IDMS pads the result with blanks

- *length* is not specified — the substring begins at *start* and ends at the end of the string
- *length* is null — the result of the function is null

The result of the SUBSTR function is a character string when *value-expression* is a character string; the result is a graphics string when *value-expression* is a graphics string.

B.2.1.36 TAN-function

TAN returns the tangent of the *value-expression*, which must be an angle expressed in radians. TAN is the inverse function of the ATAN function.

value-expression
Must be of any numeric data type. It is converted to a double precision floating-point number for processing by this function.

The result of the function is a double precision floating-point number. If the *value-expression* is null, the result is a null value. If a data error occurs, an exception is raised.

Example: The following statement returns 1.0000000000000000E+00:

```
SELECT TAN ( PI()/4 )
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.37 TANH-function

TANH returns the hyperbolic tangent of the *value-expression*, which must be an angle expressed in radians.

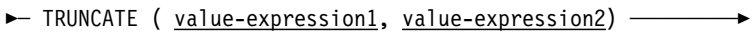
value-expression
Must be any numeric data type. It is converted to a double precision floating-point number for processing by this function.

The result of the function is a double precision floating-point number. If the *value-expression* is null the result is a null value. If a data error occurs an exception is raised.

Example: The following statement returns 6.5579420263267255E-01:

```
SELECT TANH ( PI()/4 )
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.38 TRUNCATE-function



TRUNCATE returns *value-expression1* truncated to *value-expression*2 places to the right of the decimal point if *value-expression*2 is positive or 0. If *value-expression*2 is negative, *value-expression1* is truncated to the absolute value of *value-expression*2 places to the left of the decimal point. If the absolute value of *value-expression*2 is not smaller than the number of digits to the left of the decimal point, the result is 0.

value-expression1

Specifies a numeric *value-expression* and must be of any numeric data type.

value-expression2

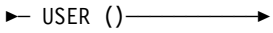
Specifies a numeric *value-expression* and must be of any numeric data type. *value-expression*2 must be of any numeric data type but is internally converted to INTEGER.

The result of the function has the same data type and attributes as *value-expression1*. The result is null if *value-expression1* or *value-expression2* is null. If an error occurs an exception is raised.

Example: The following statement returns: 627.46380, 627.46300, 627.46000, 627.40000, 627.00000, 620.00000, 600.00000, 0.00000, 0.00000

```
SELECT  TRUNCATE(627.46381, 4) ,
                        TRUNCATE(627.46381, 3) ,
                        TRUNCATE(627.46381, 2) ,
                        TRUNCATE(627.46381, 1) ,
                        TRUNCATE(627.46381, 0) ,
                        TRUNCATE(627.46381,-1) ,
                        TRUNCATE(627.46381,-2) ,
                        TRUNCATE(627.46381,-3) ,
                        TRUNCATE(627.46381,-4)
          FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.39 USER-function

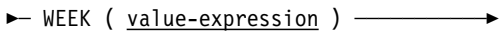


USER is equivalent to the special-register USER. For more information, see *Advantage CA-IDMS Database SQL Option Reference Guide*.

Example: The following statement returns JSMITH, the user executing the SELECT statement:

```
SELECT USER()
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

B.2.1.40 WEEK-function



WEEK returns the week of the year for the specified *value-expression*. The function uses the ISO definition: a week starts with Monday and comprises 7 days. Week 1 is the first week of the year that contains a Thursday (or the first week that contains January 4).

value-expression

Must be a DATE or TIMESTAMP data type or must be a CHARACTER or VARCHAR data type and represent a valid string representation of a date or timestamp.

The result is an INTEGER data type and is in the range of 1 to 53. The result is null if *value-expression* is null.

Example: The following statement returns 52,1:

```
SELECT WEEK ('2000-01-01'), WEEK('2000-01-03')
       FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM'
```

B.3 Revised SQL Statements

The SQL statements in this section have been revised for Release 16.0.

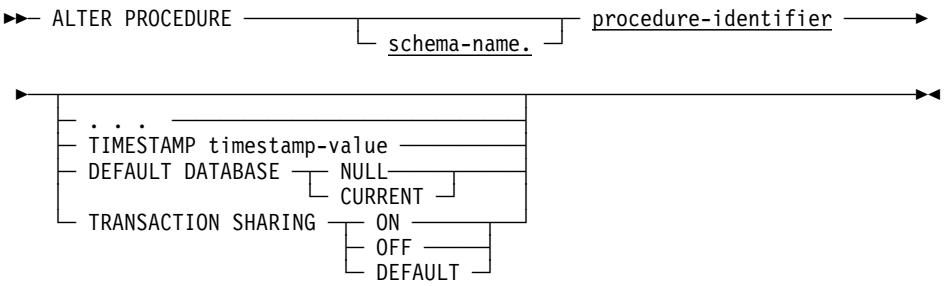
B.3.1 ALTER PROCEDURE Statement

The ALTER PROCEDURE statement is extended in Release 16.0 allowing you to:

- Update the procedure's synchronization timestamp
- Change the procedure's default database option
- Change the procedure's transaction sharing option

For a complete description of the syntax and parameters for the ALTER PROCEDURE statement, see the*Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.1.1 Syntax



B.3.1.2 Parameters

timestamp-value

Specifies the value of the synchronization stamp to be assigned to the procedure. *Timestamp-value* must be a valid external representation of a timestamp.

DEFAULT DATABASE

Specifies whether a default database should be established for database sessions started by the procedure.

NULL

Specifies that no default database should be established.

CURRENT

Specifies that the database to which the SQL session is connected should become the default for any database session started by the procedure.

TRANSACTION SHARING

Specifies whether transaction sharing should be enabled for database sessions started by the procedure. If transaction sharing is enabled for a procedure's database session, it shares the current SQL session's transaction.

ON
Specifies that transaction sharing should be enabled.

OFF
Specifies that transaction sharing should be disabled.

DEFAULT
Specifies that the transaction sharing setting that is in effect when the procedure is invoked should be retained.

B.3.1.3 Usage

Specifying a synchronization stamp: When defining or altering a procedure you can specify a value for the synchronization stamp. If explicitly specified, the synchronization stamp should always be set to a new value following the change so that the change is detectible by the runtime system.

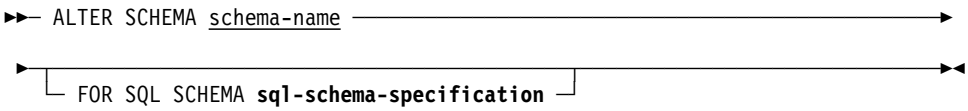
CAUTION:
Care should be exercised when explicitly specifying the synchronization stamp value, since its purpose is to enable the detection of discrepancies between a procedure function and its definition.

If not specified, the synchronization stamp is automatically set to the current date and time.

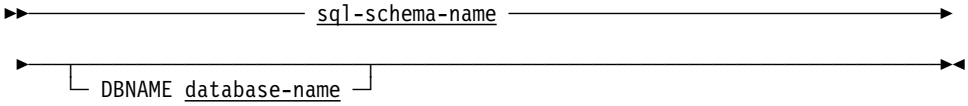
B.3.2 ALTER SCHEMA Statement

The ALTER SCHEMA statement is extended in Release 16.0 to allow you to update the referenced SQL schema. For a complete description of the syntax and parameters for the ALTER SCHEMA statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.2.1 Syntax



Expansion of sql-schema-specification:



B.3.2.2 Parameters

sql-schema-specification
Identifies an existing SQL-defined schema to which the new SQL schema refers. Expanded syntax for **sql-schema-specification** appears immediately following the statement syntax.

sql-schema-name
Names the referenced SQL-defined-schema. This named schema must not reference another schema.

DBNAME database-name
Identifies the database containing the data described by the referenced SQL-defined schema. Database-name must be a database name that is defined in the database name table or a segment name defined in the DMCL.

B.3.2.3 Usage

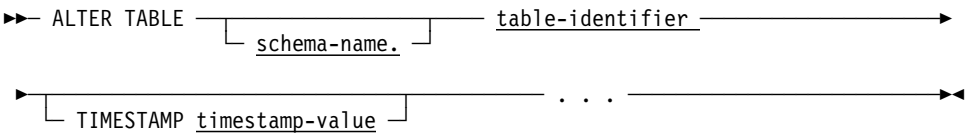
Restricted changes: You cannot alter the type of a schema, meaning that you cannot change a non-referencing schema to a referencing schema and vice versa, nor can you change the type of schema being referenced (from SQL to non-SQL and vice versa).

Changing referenced SQL schema information: If you change the name of the SQL schema that is referenced, you must drop and recreate all views that reference tables in the referencing schema (that is, the schema being altered). To determine which views are affected, use the DISPLAY ALL VIEW statement with the REFERENCED selection criteria. Before dropping the view, display its syntax by using the DISPLAY or PUNCH VIEW statement.

B.3.3 ALTER TABLE Statement

The ALTER TABLE statement is extended in Release 16.0 to allow you to update a table's synchronization timestamp with a user-specified value. For a complete description of the syntax and parameters for the ALTER TABLE statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.3.1 Syntax



B.3.3.2 Parameters

TIMESTAMP timestamp-value

Specifies the value of the synchronization stamp to be assigned to the table. *timestamp-value* must be a valid external representation of a timestamp.

B.3.3.3 Usage

Specifying a synchronization stamp: When defining or altering a table you can specify a value for the synchronization stamp. If explicitly specified, the synchronization stamp should always be set to a new value following the change so that the change is detectible by the runtime system.

CAUTION:
Care should be exercised when explicitly specifying the synchronization stamp value, since its purpose is to enable the detection of discrepancies between a table and its definition.

If not specified, the synchronization stamp is automatically set to the current date and time.

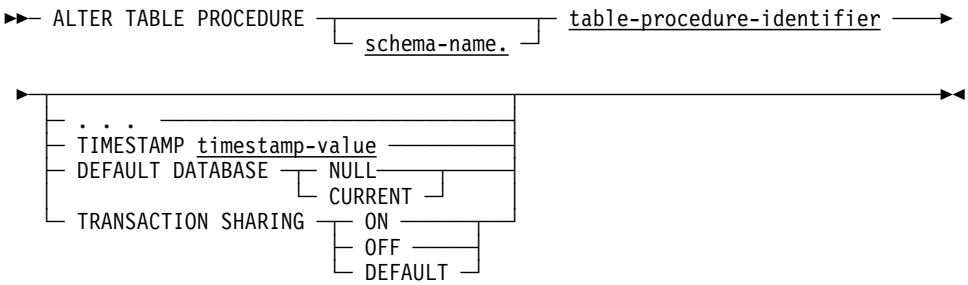
B.3.4 ALTER TABLE PROCEDURE Statement

The ALTER TABLE PROCEDURE statement is extended in Release 16.0 to allow you to:

- Update the table procedure's synchronization timestamp
- Change the table procedure's default database option
- Change the table procedure's transaction sharing option

For a complete description of the syntax and parameters for the ALTER TABLE PROCEDURE statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.4.1 Syntax



B.3.4.2 Parameters

timestamp-value
Specifies the value of the synchronization stamp to be assigned to the table procedure. *timestamp-value* must be a valid external representation of a timestamp.

DEFAULT DATABASE
Specifies whether a default database should be established for database sessions started by the table procedure.

NULL
Specifies that no default database should be established.

CURRENT
Specifies that the database to which the SQL session is connected should become the default for any database session started by the table procedure.

TRANSACTION SHARING
Specifies whether transaction sharing should be enabled for database sessions started by the table procedure. If transaction sharing is enabled for a table procedure's database session, it shares the current SQL session's transaction.

ON
Specifies that transaction sharing should be enabled.

OFF
Specifies that transaction sharing should be disabled.

DEFAULT
Specifies that the transaction sharing setting that is in effect when the procedure is invoked should be retained.

B.3.4.3 Usage

Specifying a synchronization stamp: When defining or altering a table procedure you can specify a value for the synchronization stamp. If explicitly specified, the synchronization stamp should always be set to a new value following the change so that the change is detectible by the runtime system.

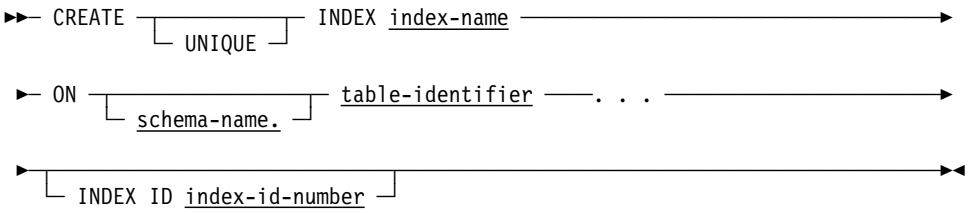
CAUTION:
Care should be exercised when explicitly specifying the synchronization stamp value, since its purpose is to enable the detection of discrepancies between a table procedure and its definition.

If not specified, the synchronization stamp is automatically set to the current date and time.

B.3.5 CREATE INDEX Statement

The CREATE INDEX statement is extended in Release 16.0 to allow you to assign an index ID value to the index being created. For a complete description of the syntax and parameters for the CREATE INDEX statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.5.1 Syntax



B.3.5.2 Parameters

INDEX ID index-id-number

Assigns an index ID value for the index being created. The *index-id-number* must be in the range of 1 through 32, 767.

B.3.5.3 Usage

Specifying an INDEX ID: When defining an index, you can specify a value for its numeric index identifier. If explicitly specified, it must be unique across all other indexes residing in the same database area.

If not specified, the index's numeric identifier is automatically set to the next available number in the range 1 through 32,767.

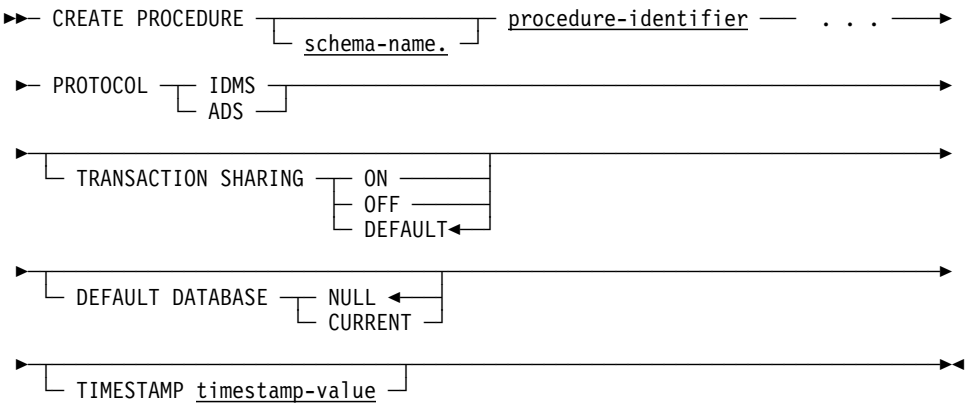
B.3.6 CREATE PROCEDURE Statement

The CREATE PROCEDURE statement is extended in Release 16.0 to allow you to:

- Specify the procedure's synchronization timestamp
- Specify the procedure's default database option
- Specify the procedure's transaction sharing option
- Specify ADS as a protocol in addition to IDMS

For a complete description of the syntax and parameters for the CREATE PROCEDURE statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.6.1 Syntax



B.3.6.2 Parameters

PROTOCOL

This is a required parameter that specifies the protocol that is used to invoke the procedure.

IDMS

Use IDMS for SQL procedures that are written in COBOL, PL/I, or Assembler and that use the same protocol as in earlier Advantage CA-IDMS releases.

ADS

Use ADS for SQL functions that are written in Advantage CA-ADS. The name of the dialog that is loaded and run when the SQL function is invoked is specified in the *external-routine-name* of the EXTERNAL NAME clause. With the protocol set to ADS, the mode clause must be set to SYSTEM.

TRANSACTION SHARING

Specifies whether transaction sharing should be enabled for database sessions started by the procedure. If transaction sharing is enabled for a procedure's database session, it shares the current SQL session's transaction.

ON

Specifies that transaction sharing should be enabled.

OFF

Specifies that transaction sharing should be disabled.

DEFAULT

Specifies that the transaction sharing setting that is in effect when the procedure is invoked should be retained.

DEFAULT DATABASE

Specifies whether a default database should be established for database sessions started by the procedure.

NULL

Specifies that no default database should be established.

CURRENT
Specifies that the database to which the SQL session is connected should become the default for any database session started by the procedure.

TIMESTAMP timestamp-value
Specifies the value of the synchronization stamp to be assigned to the procedure. *Timestamp-value* must be a valid external representation of a timestamp.

B.3.6.3 Usage

Specifying a synchronization stamp: When defining or altering a procedure you can specify a value for the synchronization stamp. If explicitly specified, the synchronization stamp should always be set to a new value following the change so that the change is detectible by the runtime system.

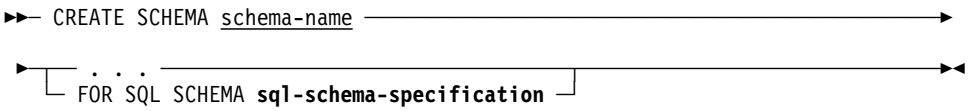
CAUTION:
Care should be exercised when explicitly specifying the synchronization stamp value, since its purpose is to enable the detection of discrepancies between a procedure and its definition.

If not specified, the synchronization stamp is automatically set to the current date and time.

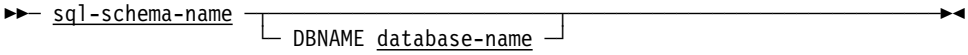
B.3.7 CREATE SCHEMA

The CREATE SCHEMA statement is extended in Release 16.0 to allow you to reference an SQL schema as an alternative to a non-SQL schema. For a complete description of the syntax and parameters for the CREATE SCHEMA statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.7.1 Syntax



Expansion of sql-schema-specification:



B.3.7.2 Parameters

sql-schema-specification
Identifies an existing SQL-defined schema to which the new SQL schema refers. Expanded syntax for **sql-schema-specification** appears immediately following the statement syntax.

sql-schema-name
Names the referenced SQL-defined-schema. This named schema must not itself reference another schema.

DBNAME database-name

Identifies the database containing the data described by the referenced SQL-defined schema. *Database-name* must be a database name that is defined in the database name table or a segment name defined in the DMCL.

If you do not specify DBNAME, no database name is included in the definition of *schema-name*. At runtime, the database to which the SQL session is connected must include segments containing the areas described by the referenced SQL-defined schema.

B.3.7.3 Usage

Creating a referencing schema: If a FOR NONSQL SCHEMA or a FOR SQL SCHEMA clause is specified, the new SQL-defined schema that is being created references the specified schema and itself becomes a referencing schema. If a non-SQL defined schema is specified, then creation of a referencing schema enables SQL access to a non-SQL defined database described by the referenced schema. Similarly, if the referenced schema is SQL-defined, then the creation of a referencing schema enables SQL access to an SQL-defined database described by the referenced schema.

In either case, if a DBNAME is specified, the referencing schema provides access to the database instance identified by *database-name*. If no DBNAME is specified, the referencing schema is **unbound** and the instance of the database to be accessed is determined at runtime. Access modules that reference tables through an unbound referencing schema can therefore be used to access more than one instance of a database.

You cannot define either a table or a view in a referencing schema; however, you can define a view in another schema that references a table through a referencing schema.

Specifying DBNAME: When you create a referencing schema, you use the DBNAME parameter to specify the name of the database containing the data. The name specified can be either the name of a database name defined in the database name table or the name of a segment included in the DMCL.

If you do not specify a database name, the database to which your SQL session is connected when accessing the data through the referencing schema must include the segments containing the data.

B.3.7.4 Example

Defining a referencing schema for an SQL-defined schema:

```
CREATE SCHEMA EMPDEMO1 FOR SQL SCHEMA DEMO DBNAME USERDB;
```

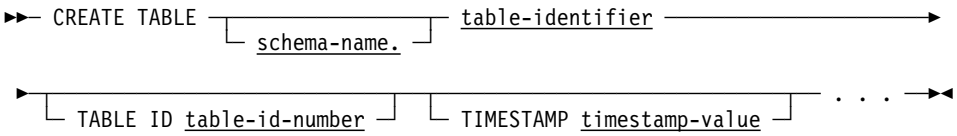
B.3.8 CREATE TABLE Statement

The CREATE TABLE statement is extended in Release 16.0 to allow you to:

- Specify the table's synchronization timestamp
- Specify the table's identification number

For a complete description of the syntax and parameters for the CREATE TABLE statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.8.1 Syntax



B.3.8.2 Parameters

TABLE ID table-id-number
Assigns a table ID value for the table being created. The *table-id-number* must be in the range of 1024 through 4095.

TIMESTAMP timestamp-value
Specifies the value of the synchronization stamp to be assigned to the table. *Timestamp-value* must be a valid external representation of a timestamp.

B.3.8.3 Usage

Specifying a synchronization stamp: When defining or altering a table you can specify a value for the synchronization stamp. If explicitly specified, the synchronization stamp should always be set to a new value following the change so that the change is detectible by the runtime system.

CAUTION:
Care should be exercised when explicitly specifying the synchronization stamp value, since its purpose is to enable the detection of discrepancies between a table and its definition.

If not specified, the synchronization stamp is automatically set to the current date and time.

Specifying a TABLE ID: When defining a table, you can specify a value for its numeric table identifier. If explicitly specified, the TABLE ID must be unique across all other tables whose rows are stored in the same database area.

If the TABLE ID is not specified, the table's numeric identifier is automatically set to the next available number in the range 1024 through 4095.

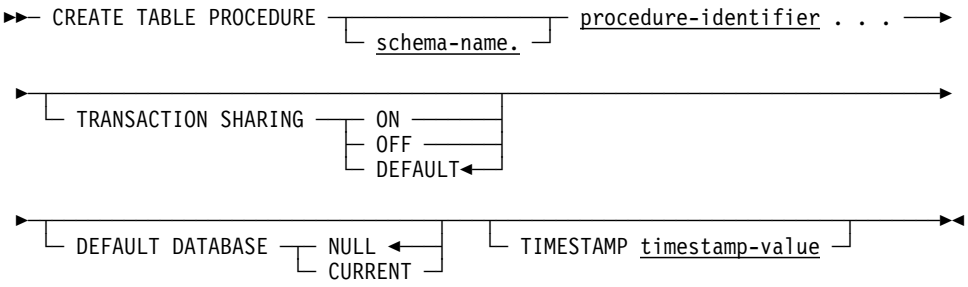
B.3.9 CREATE TABLE PROCEDURE Statement

The CREATE TABLE PROCEDURE statement is extended in Release 16.0 to allow you to:

- Specify the table procedure's synchronization timestamp
- Specify the table procedure's default database option
- Specify the table procedure's transaction sharing option

For a complete description of the syntax and parameters for the CREATE TABLE PROCEDURE statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.9.1 Syntax



B.3.9.2 Parameters

DEFAULT DATABASE
Specifies whether a default database should be established for database sessions started by the table procedure.

NULL
Specifies that no default database should be established.

CURRENT
Specifies that the database to which the SQL session is connected should become the default for any database session started by the table procedure.

TRANSACTION SHARING
Specifies whether transaction sharing should be enabled for database sessions started by the table procedure. If transaction sharing is enabled for a table procedure's database session, it shares the current SQL session's transaction.

ON
Specifies that transaction sharing should be enabled.

OFF
Specifies that transaction sharing should be disabled.

DEFAULT
Specifies that the transaction sharing setting that is in effect when the procedure is invoked should be retained.

TIMESTAMP timestamp-value

Specifies the value of the synchronization stamp to be assigned to the table procedure. *Timestamp-value* must be a valid external representation of a timestamp.

B.3.9.3 Usage

Specifying a synchronization stamp: When defining or altering a table procedure you can specify a value for the synchronization stamp. If explicitly specified, the synchronization stamp should always be set to a new value following the change so that the change is detectible by the runtime system.

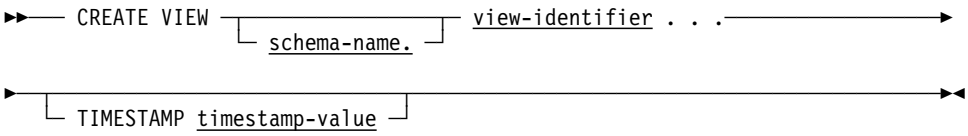
CAUTION:
Care should be exercised when explicitly specifying the synchronization stamp value, since its purpose is to enable the detection of discrepancies between a table procedure and its definition.

If not specified, the synchronization stamp is automatically set to the current date and time.

B.3.10 CREATE VIEW Statement

The CREATE VIEW statement is extended in Release 16.0 to allow you to specify a view's synchronization timestamp. For a complete description of the syntax and parameters for the CREATE VIEW statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.10.1 Syntax



B.3.10.2 Parameters

TIMESTAMP timestamp-value

Specifies the value of the synchronization stamp to be assigned to the view. *Timestamp-value* must be a valid external representation of a timestamp.

B.3.10.3 Usage

Specifying a synchronization stamp: When defining or altering a view you can specify a value for the synchronization stamp. If explicitly specified, the synchronization stamp should always be set to a new value following the change so that the change is detectible by the runtime system.

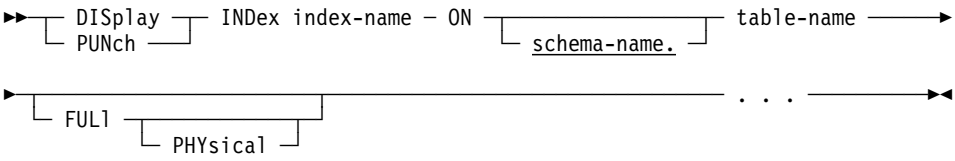
CAUTION:
Care should be exercised when explicitly specifying the synchronization stamp value, since its purpose is to enable the detection of discrepancies between a view and its definition.

If not specified, the synchronization stamp is automatically set to the current date and time.

B.3.11 DISPLAY/PUNCH INDEX Statement

The DISPLAY/PUNCH INDEX statement is extended in Release 16.0 to allow you to view the index ID. For a complete description of the syntax and parameters for the DISPLAY/ PUNCH INDEX statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.11.1 Syntax



B.3.11.2 Parameters

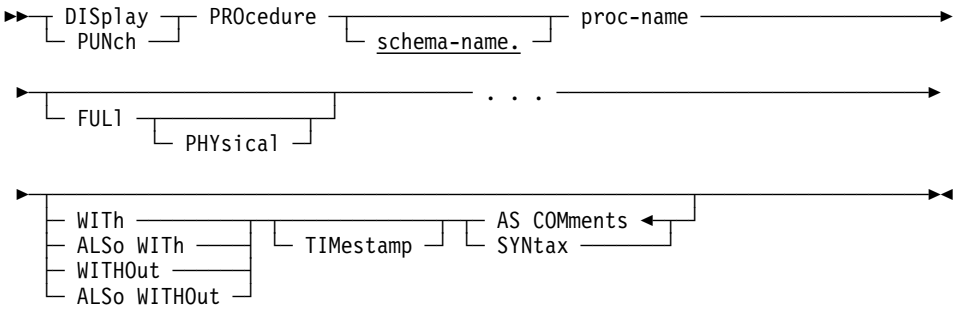
FULl
Directs Advantage CA-IDMS to display all attributes of the index except physical attributes.

PHYsical
Directs Advantage CA-IDMS to display all attributes of the index including its physical attributes. This includes the internal index ID.

B.3.12 DISPLAY/PUNCH PROCEDURE Statement

The DISPLAY/PUNCH PROCEDURE statement is extended in Release 16.0 to allow you to view the procedure's synchronization timestamp. For a complete description of the syntax and parameters for the DISPLAY/ PUNCH PROCEDURE statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.12.1 Syntax



B.3.12.2 Parameters

FULl
Directs Advantage CA-IDMS to display all attributes of the procedure except physical attributes.

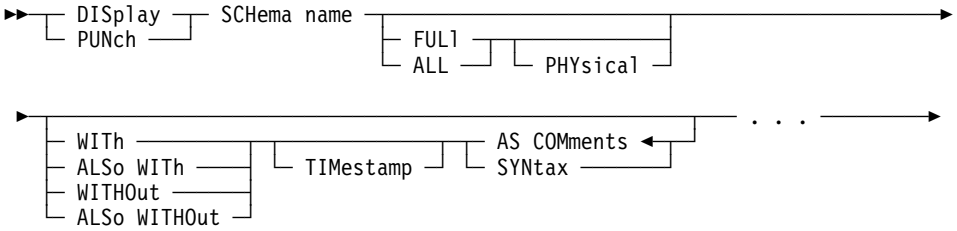
PHYsical
Directs Advantage CA-IDMS to display all attributes of the index including its physical attributes. This includes the procedure's synchronization timestamp.

TIMestamp
Specifies the display of the synchronization timestamp for the procedure.

B.3.13 DISPLAY/PUNCH SCHEMA Statement

The DISPLAY/PUNCH SCHEMA statement is extended in Release 16.0 to allow you to view the physical attributes of a schema's entities. For a complete description of the syntax and parameters for the DISPLAY/PUNCH SCHEMA statement, see the :*Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.13.1 Syntax



B.3.13.2 Parameters

ALL or FULl
Directs Advantage CA-IDMS to display all attributes of the schema except physical attributes.

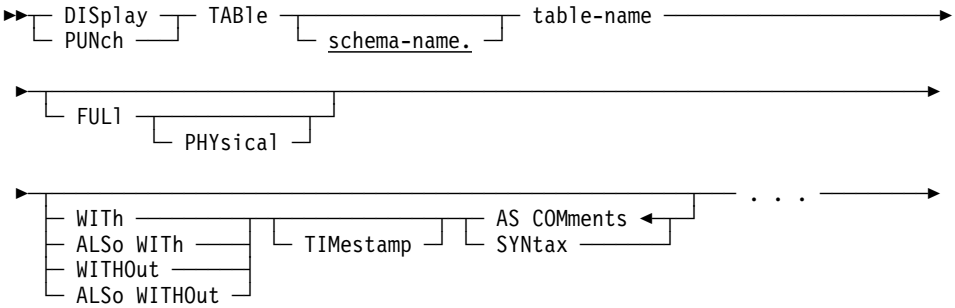
PHYsical
Directs Advantage CA-IDMS to display all attributes of the schema including its physical attributes. This includes table IDs, index IDs, and synchronization timestamps for functions, procedures, tables, table procedures, and views.

TIMestamp
Specifies the display of the synchronization timestamps for the schema entities.

B.3.14 DISPLAY/PUNCH TABLE Statement

The DISPLAY/PUNCH TABLE statement is extended in Release 16.0 to allow you to view the table's synchronization timestamp and table ID. For a complete description of the syntax and parameters for the DISPLAY/PUNCH TABLE statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.14.1 Syntax



B.3.14.2 Parameters

FULl
Directs Advantage CA-IDMS to display all attributes of the table except physical attributes.

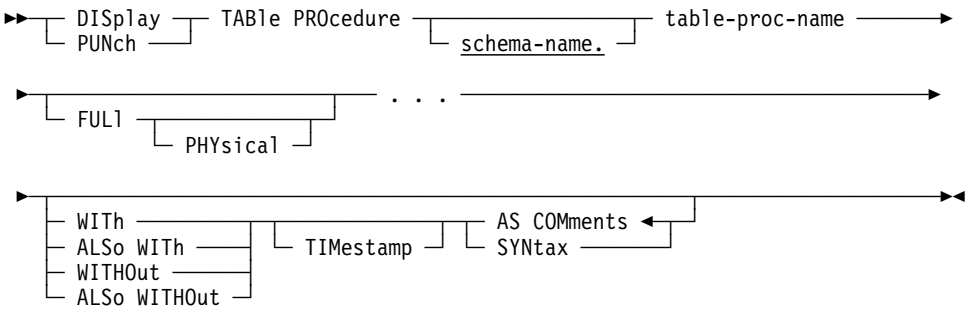
PHYsical
Directs Advantage CA-IDMS to display all attributes of the table including its physical attributes. This includes the table's synchronization timestamp and table ID

TIMestamp
Specifies the display of the synchronization timestamp for the table.

B.3.15 DISPLAY/PUNCH TABLE PROCEDURE Statement

The DISPLAY/PUNCH TABLE PROCEDURE statement is extended in Release 16.0 to allow you to view the table procedure's synchronization timestamp. For a complete description of the syntax and parameters for the DISPLAY/ PUNCH TABLE PROCEDURE statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.15.1 Syntax



B.3.15.2 Parameters

FULl
Directs Advantage CA-IDMS to display all attributes of the table procedure except physical attributes.

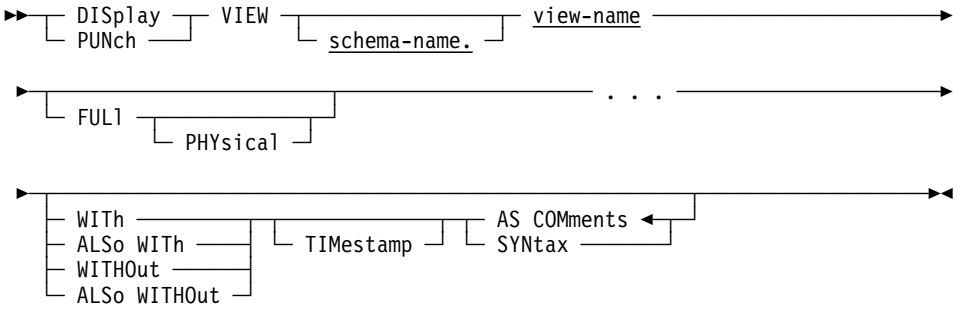
PHYsical
Directs Advantage CA-IDMS to display all attributes of the table procedure including its physical attributes. This includes the table procedure's synchronization timestamp.

TIMestamp
Specifies the display of the synchronization timestamp for the table procedure.

B.3.16 DISPLAY/PUNCH VIEW Statement

The DISPLAY/PUNCH VIEW statement is extended in Release 16.0 to allow you to display the view's synchronization timestamp. For a complete description of the syntax and parameters for the DISPLAY/PUNCH VIEW statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.16.1 Syntax



B.3.16.2 Parameters

ALL or FULl
Directs Advantage CA-IDMS to display all attributes of the view except physical attributes.

PHYsical
Directs Advantage CA-IDMS to display all attributes of the view including its physical attributes. This includes the view's synchronization timestamp.

TIMestamp
Specifies the display of the synchronization timestamp for the view.

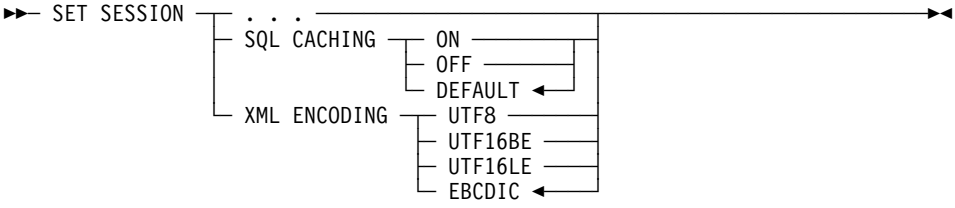
B.3.17 SET SESSION Statement

The SET SESSION statement is extended to provide the following enhancements:

- Controlling dynamic SQL statement caching
- Encoding XML values

For a complete description of the syntax and parameters for the SET SESSION statement, see the *Advantage CA-IDMS Database SQL Option Reference Guide*.

B.3.17.1 Syntax



B.3.17.2 Parameters

SQL CACHING
Specifies dynamic SQL statement caching.

ON
If SQL caching is globally enabled, the session uses caching until the session option is changed or until the caching is disabled at the system level.

OFF
Regardless of the global setting for SQL caching, the session will not use caching until the session option is changed.

DEFAULT
Same as ON.

XML ENCODING

Specifies the type of encoding to use for XML values.

XML ENCODING remains valid until the end of session or until a new SET SESSION command is executed.

UTF8

Specifies UTF-8 Unicode encoding.

UTF16BE

Specifies UTF-16 Big Endian Unicode encoding.

UTF16LE

Specifies UTF-16 Little Endian Unicode encoding.

EBCDIC

Specifies EBCDIC encoding. This is the default.

B.3.17.3 Examples

Example 1: The following example shows EBCDIC encoding:

```
set session XML ENCODING ebcdic ;
select cast(SLICE as BIN (27)) as EBCDIC
  from SYSCA.XMLSLICE
 where SLICESIZE = 27 and XMLVALUE =
 XMLCOMMENT('  0123456789ABCDEF  ');
```

The result looks like this:

```
*+ EBCDIC
*+ ------
*+ 4C5A60604040F0F1F2F3F4F5F6F7F8F9C1C2C3C4C5C6404060606E
```

Example 2: The following example shows UTF-8 encoding:

```
set session XML ENCODING UTF8 ;
*+ Status = 0        SQLSTATE = 00000
select cast(SLICE as BIN (27)) as "UTF-8"
  from SYSCA.XMLSLICE
 where SLICESIZE = 27 and XMLVALUE =
 XMLCOMMENT('  0123456789ABCDEF  ');
```

The result looks like this:

```
*+ UTF-8
*+ -----
*+ 3C212D2D2020303132333435363738394142434445462020 2D2D3E
```

Example 3: The following example shows UTF-16 Big Endian encoding:

```
set session XML ENCODING UTF16BE ;
*+ Status = 0        SQLSTATE = 00000
select cast(SLICE as BIN (27)) as "UTF-16 BE"
  from SYSCA.XMLSLICE
 where SLICESIZE = 27 and XMLVALUE =
 XMLCOMMENT('  0123456789ABCDEF  ');
```

The result looks like this:

```
++ UTF-16 BE
++ ---------
++ 003C0021002D002D00200020003000310032003300340035003600
++ 37003800390041004200430044004500460020002000 2D002D003E
```

Example 4: The following example shows UTF-16 Little Endian encoding:

```
set session XML ENCODING UTF16LE ;
++ Status = 0        SQLSTATE = 00000
select cast(SLICE as BIN (27)) as "UTF-16 LE"
  from SYSCA.XMLSLICE
 where SLICESIZE = 27 and XMLVALUE =
XMLCOMMENT('  0123456789ABCDEF  ');
```

The result looks like this:

```
++ UTF-16 LE
++ ---------
++ 3C0021002D002D00200020003000310032003300340035003600 37
++ 3800390041004200430044004500460020002000 2D002D003E00
```

B.4 SQL/XML Functions and Table Procedure

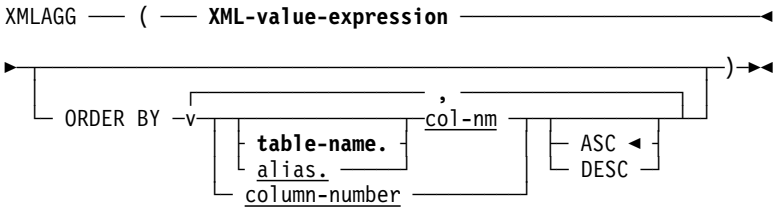
This section contains the SQL/XML functions and table procedure supplied in the XML Publishing enhancement.

►► For more information on using XML Publishing, see 4.6, “XML Publishing” on page 4-33.

B.4.1 XMLAGG-function

Returns an XML value that is computed from a collection of rows. The result is the XML concatenation of a list of XML elements, aggregated in the statement containing the XMLAGG-function.

B.4.1.1 Syntax



B.4.1.2 Parameters

ORDER BY

Before the aggregation takes place, the XML elements, specified by **XML-value-expression**, are sorted in ascending or descending order by the values in the specified columns. XML elements are ordered first by the first column specified, then by the second column specified within the ordering established by the first column, then by the third column specified, and so on.

col-nm

Specifies the name of column.

table-name.

Specifies the table, view, procedure, or table procedure that includes the named column.

alias

Specifies the *alias* associated with the table, view, procedure, or table procedure that includes the named column. The *alias* must be defined in the FROM parameter of the subquery, query specification, or SELECT statement that includes the XMLAGG function.

column-number

Specifies a column number. You can specify from 1 through 254 columns. Multiple columns must be separated by commas.

B.4.1.3 Examples

Example 1: Use of the XMLAGG function to display all employees belonging to each department.

```
SELECT XMLSERIALIZE(CONTENT
                        XMLELEMENT(NAME "dept",
                                   XMLATTRIBUTES(e.DEPT_ID AS "id"),
                                   XMLAGG(XMLELEMENT(NAME "lname",
                                                     e.EMP_LNAME)))
                        AS VARCHAR(256)) AS "EMP_NAME_COL"
     FROM DEMOEMPL.EMPLOYEE e GROUP BY DEPT_ID ;
```

The result is similar to the following. Note that the content of the EMP_NAME_COL column has been formatted for convenience.

```
EMP_NAME_COL
------------
<dept id="1100">
  <lname>Fordman</lname>
  <lname>Halloran</lname>
  <lname>Hamel</lname>
</dept>
<dept id="1110">
  <lname>Widman</lname>
  <lname>Alexander</lname>
</dept>
<dept id="1120">
  <lname>Umidy</lname>
  <lname>White</lname>
  <lname>Johnson</lname>
</dept>
```

Example 2: Use of the XMLAGG function to display all employees belonging to each department. For each employee, the positions and jobs are included. This example shows that the use of the XMLAGG function together with the ability to specify subqueries as arguments for the SQL/XML functions allows creating very complex XML structures.

```
select xmlpointer (
 xmlelement
 ( Name "Employees"
 , xmlagg
   ( xmlelement
     ( NAME "Department"
     , xmlattributes(DEPT_ID as "DeptId")
     ,  xmlelement
        ( NAME "EmployeesInDepartment"
        , select xmlagg
          ( xmlelement
            ( name "Employee"
            , xmlattributes(EMP_ID as "EmpId")
            , EMP_FNAME
            , EMP_LNAME
            , xmlelement
```

```
                        ( name "Address"
                        , XMLFOREST
                          ( e.STREET     as  "Street"
                          , e.CITY       as  "City"
                          , e.STATE      as  "State"
                          )
                        )
                      , xmlelement
                        ( name "Positions"
                        , select xmlagg
                          ( xmlelement
                            ( name "Position"
                            , xmlattributes
                              ( p.JOB_ID      as "JobId" )
                            , JOB_TITLE
                            , SALARY_AMOUNT
                            , BONUS_PERCENT
                            )
                          )
                           from DEMOEMPL.POSITION p, DEMOEMPL.JOB j
                          where p.EMP_ID = e.EMP_ID
                            and p.JOB_ID = j.JOB_ID
                        )
                      )
                    )
                     from DEMOEMPL.EMPLOYEE e
                    where d.DEPT_ID = e.DEPT_ID
                  )
                )
              )
            )
          )
        )from DEMOEMPL.DEPARTMENT d
```

The result is similar to the following. It has been formatted and displayed with an "XML-enabled" Web browser that allows collapsing and expanding XML elements in an XML tree.

```
- <Employees>
+ <Department DeptId="">1120">
- <Department DeptId="5000">
- <EmployeesInDepartment>
- <Employee EmpId="3449">
  Cynthia Taylor
- <Address>
  <Street>201 Washington St</Street>
  <City>Concord</City>
  <State>MA</State>
  </Address>
- <Positions>
  <Position JobId="4023">Accountant 74776.0</Position>
  </Positions>
  </Employee>
+ <Employee EmpId="5103">
```

```
  ⋮
  </EmployeesInDepartment>
  </Department>
+ <Department DeptId="4500">
  </Employees>
```

Example 3: Use of the XMLAGG function and subqueries to display part of an SQL catalog as an XML document.

```
select xmlpointer (
 xmlelement
 ( Name "Catalog"
 , xmlagg
   ( xmlelement
     ( Name "Schema"
     , xmlattributes
       ( s.NAME as "Name"
       , s.TYPE as "Type"
       )
     , 'Referencing SQL Schema:'
     , s.REFDSQLSCHEMA
     , 'Referencing Non SQL Schema:'
     , s.NTWKSCHEMA
     , select
         xmlagg
         ( xmlelement
           ( Name "TablesInSchema"
           , xmlattributes
             ( t.NAME    as "Name"
             , t.TYPE    as "Type"
             , t.LENGTH  as "Length"
             )
           , SEGMENT
           , '.'
           , 'AREA'
           , xmlelement
             ( Name "TableStats"
             , xmlattributes
               ( t.NUMCOLS        as "NumCols"
               , t.NUMINDEXES     as "NumIndexes"
               , t.NUMREFERENCING as "NumReferencing"
               , t.NUMROWS        as "NumRows"
               , t.NUMPAGES       as "NumPages"
               , t.NUMSYNTAX      as "NumSyntax"
               , t.ESTROWS        as "EstRows"
               )
             )
           , select
               xmlagg
               ( xmlelement
                 ( Name "ColumnsInTable"
                 , xmlattributes
                   ( c.NAME   as "Name"
```

```
                                , c.NUMBER as "Nr"
                                )
                              , TYPE
                              , xmlelement
                                ( Name "DataTypeDetails"
                                , xmlattributes
                                  ( c.TYPECODE  as "Code"
                                  , c.PRECISION as "Precision"
                                  , c.SCALE     as "Scale"
                                  )
                                )
                              , xmlelement
                                ( Name "OtherDetails"
                                , xmlattributes
                                  ( c.NULLS     as "Null"
                                  , c.DEFAULT   as "Default"
                                  , c.VOFFSET   as "VOffset"
                                  , c.VLENGTH   as "VLength"
                                  , c.NOFFSET   as "NOffset"
                                  , c.NLENGTH   as "NLength"
                                  , c.NUMVALUES as "NumValues"
                                  )
                                )
                            )
                           )
                           from SYSTEM.COLUMN c
                          where c.TABLE  = t.NAME
                            and c.SCHEMA = t.SCHEMA
                      )
                     )
                     from SYSTEM.TABLE t
                    where t.SCHEMA = s.NAME
                      and TYPE = 'T'
                )
              )
             )
            )from SYSTEM.SCHEMA s
```

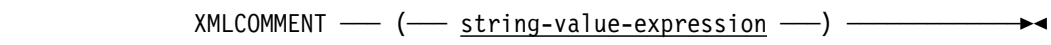
The result is similar to the following. It has been formatted and displayed with an "XML-enabled" Web browser that allows collapsing and expanding XML elements in an XML tree.

```
- <Catalog>
  <Schema Name="EMPSCHM" Type="N">
   Referencing SQL Schema: Referencing Non SQL Schema:EMPSCHM</Schema>
- <Schema Name="DEMOEMPL" Type="R">
   Referencing SQL Schema:Referencing Non SQL Schema:
+ <TablesInSchema Name="DEPARTMENT" Type="T" Length="68">
+ <TablesInSchema Name="DIVISION" Type="T" Length="56">
+ <TablesInSchema Name="EMPL_MANAGER_INFO" Type="T" Length="56">
+ <TablesInSchema Name="EMPLOYEE" Type="T" Length="204">
   SQLDEMO .AREA
  <TableStats NumCols="15" NumIndexes="4" NumReferencing="2" NumRows="55"
   NumPages="40" NumSyntax="1" EstRows="0" />
- <ColumnsInTable Name="DEPT_ID" Nr="5">
   UNSIGNED NUMERIC
  <DataTypeDetails Code="128" Precision="4" Scale="0" />
  <OtherDetails Null="N" Default="N" VOffset="49" VLength="4" NOffset="0"
   NLength="0" NumValues="14" />
  </ColumnsInTable>
- <ColumnsInTable Name="EMP_FNAME" Nr="3">
   CHARACTER
  <DataTypeDetails Code="1" Precision="0" Scale="0" />
  <OtherDetails Null="N" Default="N" VOffset="9" VLength="20" NOffset="0"
   NLength="0" NumValues="0" />
  </ColumnsInTable>
- <ColumnsInTable Name="EMP_ID" Nr="1">
   UNSIGNED NUMERIC
  <DataTypeDetails Code="128" Precision="4" Scale="0" />
  <OtherDetails Null="N" Default="N" VOffset="0" VLength="4" NOffset="0"
   NLength="0" NumValues="0" />
  </ColumnsInTable>

  ⋮
  </TablesInSchema>

  ⋮
+ <TablesInSchema Name="INSURANCE_PLAN" Type="T" Length="168">
+ <TablesInSchema Name="JOB" Type="T" Length="188">
+ <TablesInSchema Name="POSITION" Type="T" Length="64">
   </Schema>

  ⋮
   </Catalog>
```

B.4.2 XMLCOMMENT-function

Returns an XML value that is an XML comment, generated from *string-value-expression*. The XML value consists of an XML root information item with one child, an XML comment information item whose [content] property is *string-value-expression*.

B.4.2.1 Syntax



B.4.2.2 Parameters

string-value-expression

Specifies a character string **value-expression**, that is a **value-expression** that returns a value of type character.

If *string-value-expression* is NULL, XMLCOMMENT returns a NULL value. *string-value-expression* cannot contain a hyphen (--) sequence of characters and cannot end with a hyphen (-) character.

B.4.2.3 Example

The following statement returns a single XML comment:

```
SELECT XMLSERIALIZE(CONTENT XMLCOMMENT('My personal opinion')
                    AS CHAR(80)) as "Comment Only"
 FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

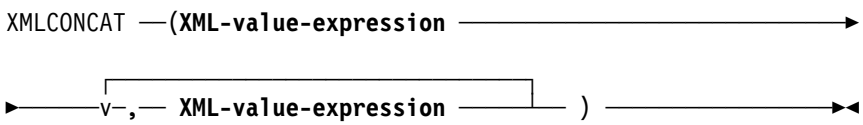
The result is similar to the following:

```
Comment Only
------------
<!--My personal opinion-->
```

B.4.3 XMLCONCAT-function

Returns an XML value that is the concatenation of all the **XML-value-expressions**. If all the **XML-value-expressions** are NULL or empty, a NULL value is returned.

B.4.3.1 Syntax



B.4.3.2 Example

Use of the XMLCONCAT function to concatenate two XML elements defined using the XMLELEMENT function.

```
SELECT e.EMP_ID,
       XMLSERIALIZE(CONTENT
                    XMLCONCAT(XMLELEMENT(NAME "fname",
                                         e.EMP_FNAME),
                              XMLELEMENT(NAME "lname",
                                         e.EMP_LNAME))
                    AS VARCHAR(64)) AS "EMP_NAME_COL"
  FROM DEMOEMPL.EMPLOYEE e WHERE EMP_ID < 1500 ;
```

The result is similar to the following:

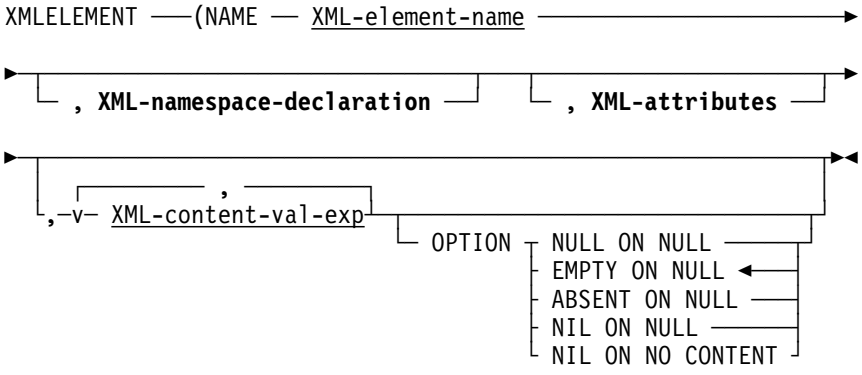
```
EMP_ID  EMP_NAME_COL
------  ------------
  1003  <fname>James</fname><lname>Baldwin</lname>
  1034  <fname>James</fname><lname>Gallway</lname>
  1234  <fname>Thomas</fname><lname>Mills</lname>
```

B.4.4 XMLELEMENT-function

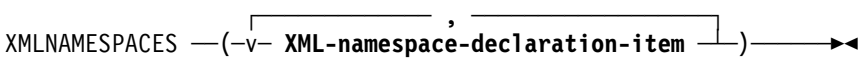
Returns an XML value that is a single XML element information item as a child of its XML root information. Provided are an XML element name, an optional list of XML namespace declarations, an optional list of attributes, and an optional list of values as the content of the new element.

The XMLATTRIBUTES pseudo function can be used to specify XML attributes in an XML element. The XMLNAMESPACES pseudo function can be used to declare XML namespace in an XML element.

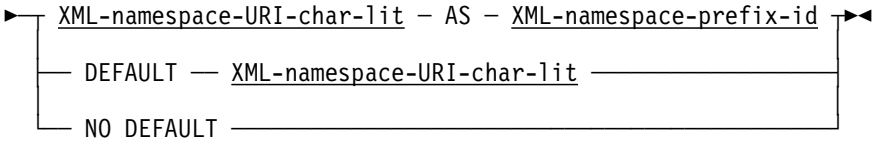
B.4.4.1 Syntax



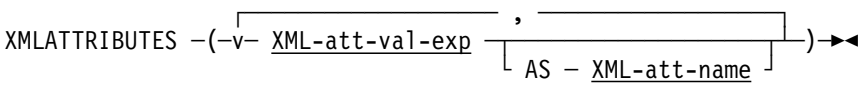
Expansion of XML-namespace-declaration



Expansion of XML-namespace-declaration-item



Expansion of XML-attributes



B.4.4.2 Parameters

XML-element-name

Specifies an identifier that is used as an XML element name. This name must be an XML QName. If the name is qualified, the namespace prefix must be declared within the scope. The maximum length of the identifier is 128 characters.

XML-content-val-exp

Specifies a **value-expression** or an **XML-value-expression** that after mapping according to 4.6.4.3, “Mapping SQL Data Type Values to XML Schema Data Type Values” on page 4-37, is used as the content of the generated XML element.

XML-namespace-URI-char-lit

Specifies a character string literal of an XML namespace through a URI. For example, http://www.w3.org/2001/XMLSchema. This character string literal can be empty when used with the DEFAULT option only.

XML-namespace-prefix-id

Specifies an identifier that is used as a namespace prefix that is bound to the XML namespace given by *XML-namespace-URI-char-lit*. The maximum length of the identifier is 128 characters.

This identifier must be an XML NCName. It cannot be equal to "xml" or "xmlns", and it cannot start with the characters "xml" (in any combination). Be sure that no duplicate namespace prefixes are declared in the same XMLNAMESPACES function call.

XML-att-name

Specifies an identifier that is used as the XML attribute name. The maximum length of the identifier is 128 characters. The attribute name must be an XML QName. It cannot be equal to "xmlns" or start with "xmlns:". Be sure that no duplicate attribute names are declared in the same XMLATTRIBUTES function call.

XML-att-val-exp

Specifies a *value-expression* that is used as the value of the XML attribute. The *value-expression* can be of any type except GRAPHIC or VARGRAPHIC. The length of the value is limited to 512 characters.

If *XML-att-name* is not specified, the attribute name is derived from the *XML-att-val-exp* value. The *XML-att-val-exp* value must be a valid SQL column name, optionally qualified with **table-name** or *alias* The fully escaped mapping is applied on the SQL column name to create the attribute name.

OPTION

Specifies the processing of null values for *XML-content-val-exp* as follows:

ABSENT ON NULL

The returned value is never null, but element is completely absent when all *XML-content-val-exp* are NULL.

EMPTY ON NULL

The returned value is never null. Element has no content for each NULL value of *XML-content-val-exp* that is NULL. This is the default.

NIL ON NO CONTENT

The returned value is not null. When the [children] property of the element does not contain at least one XML element information item or at least one XML character information item, the element is:

```
<XML-element-name xsi:nil="true"/>
```

NIL ON NULL
The returned value is not null, but when all *XML-content-val-exp* are NULL, element is

```
<XML-element-name xsi:nil="true"/>
```

with implicit xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance.

NULL ON NULL
The returned value is NULL when all *XML-content-val-exp* are NULL.

Notes:

- XMLATTRIBUTES look like any other SQL/XML function, but it is not an SQL function. It is a function-like construct that can only be used in an XMLELEMENT invocation.
- An *XML-namespace-declaration* can have only one *XML-namespace-declaration-item* containing the literal DEFAULT or NO DEFAULT.
- *XML-element-name* is an identifier in the SQL language. Some valid XML names, that is, all XML QNames with a non-null prefix, require this identifier to be delimited by double quotes.

B.4.4.3 Examples

Example 1: The following SELECT statement produces a row for each employee with one column representing an 'emp' XML element containing the employee's last name. The datatype of the result column is VARCHAR(64).

```
SELECT XMLSERIALIZE(CONTENT
                    XMLELEMENT(NAME "emp", EMP_LNAME)
                    AS VARCHAR(64)) AS "EMP_NAME_COL"
  FROM DEMOEMPL.EMPLOYEE ;
```

The result is similar to the following:

```
EMP_NAME_COL
------------
<emp>Baldwin</emp>
<emp>Gallway</emp>
<emp>Mills</emp>
```

Example 2: Same as Example 1, but this example includes a first column containing the employee ID, which uses the "e" alias for the table name and a WHERE clause on the SELECT statement.

```
SELECT e.EMP_ID,
       XMLSERIALIZE(CONTENT
                    XMLELEMENT(NAME "emp", e.EMP_LNAME)
                    AS VARCHAR(64)) AS "EMP_NAME_COL"
  FROM DEMOEMPL.EMPLOYEE e WHERE EMP_ID < 1500 ;
```

The result is similar to the following:

```
EMP_ID  EMP_NAME_COL
------  ------------
  1003  <emp>Baldwin</emp>
  1034  <emp>Gallway</emp>
  1234  <emp>Mills</emp>
```

Example 3: Same as Example 2, but this example also includes the employee ID as an attribute within the <emp> tag.

```
SELECT e.EMP_ID,
       XMLSERIALIZE(CONTENT
                    XMLELEMENT(NAME "emp",
                               XMLATTRIBUTES(e.EMP_ID AS "id"),
                               e.EMP_LNAME)
                    AS VARCHAR(64)) AS "EMP_NAME_COL"
  FROM DEMOEMPL.EMPLOYEE e WHERE EMP_ID < 1500 ;
```

The result is similar to the following:

```
EMP_ID  EMP_NAME_COL
------  ------------
  1003  <emp id="1003">Baldwin</emp>
  1034  <emp id="1034">Gallway</emp>
  1234  <emp id="1234">Mills</emp>
```

Example 4: Same as Example 3, but this example includes a second attribute within the <emp> tag with the employee's first name.

```
SELECT e.EMP_ID,
    XMLSERIALIZE(CONTENT
                 XMLELEMENT(NAME "emp",
                            XMLATTRIBUTES(e.EMP_ID    AS "id",
                                          e.EMP_FNAME AS "fname"),
                            e.EMP_LNAME)
                 AS VARCHAR(64)) AS "EMP_NAME_COL"
  FROM DEMOEMPL.EMPLOYEE e WHERE EMP_ID < 1500 ;
```

The result is similar to the following. Note that the content of the EMP_NAME_COL column has been formatted for convenience.

```
EMP_ID  EMP_NAME_COL
------  ------------
  1003  <emp id="1003" fname="James">Baldwin</emp>
  1034  <emp id="1034" fname="James">Gallway</emp>
  1234  <emp id="1234" fname="Thomas">Mills</emp>
```

Example 5: Same as Example 4, but this example removes the attributes and uses the employee's first name and last name as sub-elements of <emp>.

```
SELECT e.EMP_ID,
       XMLSERIALIZE(CONTENT
                    XMLELEMENT(NAME "emp",
                         XMLELEMENT(NAME "fname", e.EMP_FNAME),
                         XMLELEMENT(NAME "lname", e.EMP_LNAME))
                    AS VARCHAR(64)) AS "EMP_NAME_COL"
  FROM DEMOEMPL.EMPLOYEE e WHERE EMP_ID < 1500 ;
```

The result is similar to the following. Note that the content of the EMP_NAME_COL column has been formatted for convenience.

```
EMP_ID  EMP_NAME_COL
------  ------------
  1003  <emp>
          <fname>James</fname>
          <lname>Baldwin</lname>
        </emp>
  1034  <emp>
          <fname>James</fname>
          <lname>Gallway</lname>
        </emp>
  1234  <emp>
          <fname>Thomas</fname>
          <lname>Mills</lname>
        </emp>
```

Example 6: Same as Example 5, but this example concatenates the employee's first name and last name into one sub-element of <emp>.

```
SELECT e.EMP_ID,
       XMLSERIALIZE(CONTENT
         XMLELEMENT(NAME "emp",
                    XMLELEMENT(NAME "name",
                               e.EMP_FNAME ||' '|| e.EMP_LNAME))
          AS VARCHAR(64)) AS "EMP_NAME_COL"
       FROM DEMOEMPL.EMPLOYEE e WHERE EMP_ID < 1500 ;
```

The result is similar to the following. Note that the content of the EMP_NAME_COL column has been formatted for convenience.

```
EMP_ID  EMP_NAME_COL
------  ------------
  1003  <emp>
          <name>James Baldwin</name>
        </emp>
  1034  <emp>
          <name>James Gallway</name>
        </emp>
  1234  <emp>
          <name>Thomas Mills</name>
        </emp>
```

Example 7: Same as Example 6, but this example uses XMLNAMESPACES.

```
SELECT e.EMP_ID,
       XMLSERIALIZE(CONTENT
         XMLELEMENT(NAME "emp",
              XMLNAMESPACES(
                         DEFAULT 'http://ca.com/hr/globalxml',
                         'http://ca.com/hr/frenchxml' AS "fr" ),
                    XMLELEMENT(NAME "fr:nom",
                               e.EMP_FNAME ||' '|| e.EMP_LNAME))
          AS VARCHAR(64)) AS "EMP_NAME_COL"
       FROM DEMOEMPL.EMPLOYEE e WHERE EMP_ID < 1500 ;
```

The result is similar to the following. Note that the content of the EMP_NAME_COL column has been formatted for convenience.

```
EMP_ID  EMP_NAME_COL
------  ------------
1003    <emp xmlns="http://ca.com/hr/globalxml"
             xmlns:fr="http://ca.com/hr/frenchxml">
          <fr:nom>James Baldwin</fr:nom>
        </emp>
1034    <emp xmlns="http://ca.com/hr/globalxml"
             xmlns:fr="http://ca.com/hr/frenchxml">
          <fr:nom>James Gallway</fr:nom>
        </emp>
1234    <emp xmlns="http://ca.com/hr/globalxml"
             xmlns:fr="http://ca.com/hr/frenchxml">
          <fr:nom>Thomas Mills</fr:nom>
        </emp>
```

Example 8: This example illustrates the use of a subquery as an argument of XMLELEMENT.

```
SELECT
  XMLSERIALIZE
  ( CONTENT
    XMLELEMENT
    ( NAME "Employee"
    ,XMLATTRIBUTES(e.EMP_ID as "Id")
    , e.EMP_FNAME
    , e.EMP_LNAME
    , SELECT
        XMLELEMENT
        ( NAME "Manager"
        , XMLATTRIBUTES(m.EMP_ID as "MgrId")
        , m.EMP_FNAME || m.EMP_LNAME
        )
       FROM DEMOEMPL.employee m
      WHERE e.MANAGER_ID = m.EMP_ID
  ) AS VARCHAR(120)) AS "EmployeeManager"
FROM DEMOEMPL.EMPLOYEE e
```

The result is similar to the following:

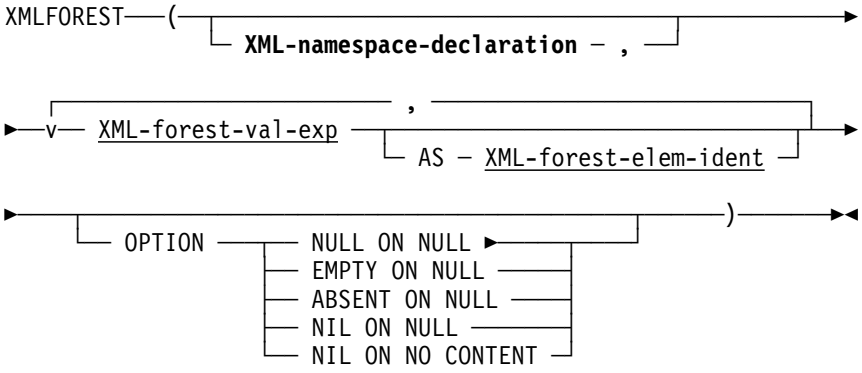
```
EmployeeManager
---------------
<Employee Id="5008">Timothy Fordman
   <Manager MgrId="2246">Marylou Hamel</Manager></Employee>
<Employee Id="4703">Martin Halloran
   <Manager MgrId="2246">Marylou Hamel</Manager></Employee>
```

B.4.5 XMLFOREST-function

Returns an XML value that is a list of XML element information items as the children of its XML root information. An XML element is produced from each *XML-forest-val-exp*, using the column name or, if provided, the *XML-forest-elem-ident* as the XML element name and the *XML-forest-val-exp* as the element content. The value of *XML-forest-val-exp* can be any value that has a mapping to an XML value.

The XMLNAMESPACES pseudo function can be used to declare XML namespace in an XML element.

B.4.5.1 Syntax



Expansion of XML-namespace-declaration: See B.4.4, "XMLELEMENT-function" on page B-69.

B.4.5.2 Parameters

XML-forest-elem-ident

Specifies an identifier that is used as an XML element name. The identifier must be an XML QName. If a namespace prefix is used, it must have been declared in the scope of the element. The maximum length of the identifier is 128 characters.

XML-forest-val-exp

Specifies a *value-expression* that is used as the element content of an XML element. If *XML-forest-elem-ident* is not specified, the forest element name is derived from the *XML-forest-val-exp* value. The *XML-forest-val-exp* value must be a valid SQL column name, optionally qualified with **table-name** or *alias*. The fully escaped mapping is applied on the SQL column name to create the forest element name.

OPTION

Specifies the processing of null values for *XML-forest-val-exp* as follows:

ABSENT ON NULL

The returned value is never null, but element is completely absent from the list when *XML-forest-val-exp* is NULL.

EMPTY ON NULL
The returned value is never null. Element has no content for each NULL value of *XML-forest-val-exp*.

NIL ON NO CONTENT
The returned value is not null. When the [children] property of element does not contain at least one XML element information item or at least one XML character information item, the element is:

```
<XML-forest-element-name xsi:nil="true"/>.
```

NIL ON NULL
The returned value is not null, but when an *XML-forest-val-exp* is NULL, element becomes

```
<XML-forest-element-name xsi:nil="true"/>
```

with implicit xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance.

NULL ON NULL
The returned value is NULL when all *XML-forest-val-exp* are NULL. This is the default.

Note: *XML-element-name* is an identifier in the SQL language. Some valid XML names, that is, all XML QNames with non-null namespace prefix, requires this identifier to be delimited by double quotes.

B.4.5.3 Example

Similar to “Example 5” on page B-72 in B.4.4, “XMLELEMENT-function,” but the use of two XMLELEMENT invocations to declare two sub-elements of <emp> is replaced by a single XMLFOREST invocation.

```
SELECT e.EMP_ID,
       XMLSERIALIZE(CONTENT
                    XMLELEMENT(NAME "emp",
                               XMLFOREST(e.EMP_FNAME AS "fname",
                                         e.EMP_LNAME AS "lname"))
                    AS VARCHAR(64)) AS "EMP_NAME_COL"
  FROM DEMOEMPL.EMPLOYEE e WHERE EMP_ID < 1500 ;
```

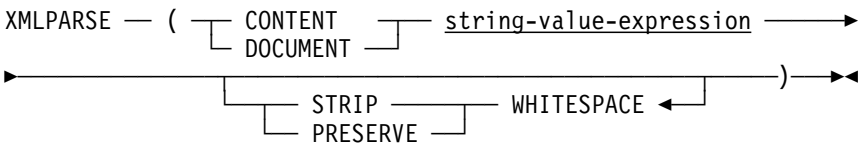
The result is similar to the following. Note that the content of the EMP_NAME_COL column has been formatted for convenience.

```
EMP_ID  EMP_NAME_COL
------  ------------
  1003  <emp>
          <fname>James</fname>
          <lname>Baldwin</lname>
        </emp>
  1034  <emp>
          <fname>James</fname>
          <lname>Gallway</lname>
        </emp>
  1234  <emp>
          <fname>Thomas</fname>
          <lname>Mills</lname>
        </emp>
```

B.4.6 XMLPARSE-function

Returns an XML value as the result of performing a non-validating parse of a character string. Parsing is the inverse operation of serializing.

B.4.6.1 Syntax



B.4.6.2 Parameters

string-value-expression

Specifies a character string **value-expression**, that is a **value-expression** that returns a value of type character. If *string-value-expression* is NULL, XMLPARSE returns a NULL value.

Note: The DOCUMENT and STRIP WHITESPACE options are not functional in this feature. This means that CONTENT and PRESERVE WHITESPACE should always be specified.

B.4.6.3 Example

The following statement causes an SQL statement exception because the XML is not completely serialized. The serialization is truncated after 20 characters.

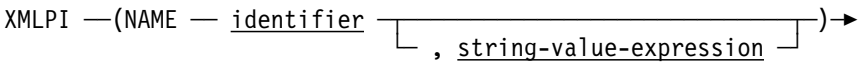
```
SELECT
  XMLPARSE
  ( CONTENT
    XMLSERIALIZE
    ( CONTENT
      XMLELEMENT
      ( NAME "EMP", EMP_LNAME
      ) AS CHAR(20)
    )
  )
 FROM DEMOEMPL.EMPLOYEE ;

*+ Status = -4       SQLSTATE = 38000        Messages follow:
*+ DB001075 C-4M321: Procedure IDMSQFUX exception 38000 XMLPARSE: Premature end
*+ of data in tag EMP line 1
```

B.4.7 XMLPI-function

Returns an XML value that is an XML processing instruction (PI). The XML value consists of:

- An XML root information item with one child
- An XML processing information item whose [target] property is the partially escaped mapping of *identifier* to an XML Name, and whose [content] property is *string-value-expression*, trimmed of leading blanks.

B.4.7.1 Syntax



B.4.7.2 Parameters

Identifier
Specifies the target in the processing instruction. It must be a valid NCName. The maximum length of the *identifier* is 128 characters.

string-value-expression
Specifies a character string **value-expression**, that is a **value-expression** that returns a value of type character. It can be NULL or empty, but if present, it cannot contain the "?>" sequence.

A processing instruction takes the following syntactical form in XML 1.0:

```
<?target data?>
```

Processing instructions instruct applications to perform some type of extra processing on a given document.

An example of a processing instruction, which is supported by most Web browsers is:

```
<?xml-stylesheet href="mystyle.xsl" type="text/xsl"?>
```

When the browser loads an XML document and recognizes the processing instruction, it performs a transformation using the specified XSLT file and displays the result of the transformation instead of the raw XML file. This processing instruction has been accepted as a W3C recommendation (for more information, see http://www.w3.org/TR/xml-stylesheet.)

Another example of a processing instruction accepted as a W3C recommendation is the use of the XML declaration:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

The pseudo-attribute *version* currently must have value "1.0". The pseudo-attribute *standalone* specifies whether any markup declarations are defined in separate documents. Finally, the pseudo-attribute *encoding* specifies the encoding of the XML document. XML parsers are required to support at least encoding UTF-8 and UTF-16.

B.4.7.3 Example

The following statement returns only a single processing instruction:

```
 SELECT XMLSERIALIZE(CONTENT XMLPI (NAME "xml" ,
      '   version="1.0" encoding="UTF-8" standalone="yes"')
                      AS CHAR(80)) as "PI Instruction"
  FROM SYSTEM.SCHEMA WHERE NAME = 'SYSTEM';
```

The result is similar to the following:

```
PI Instruction
--------------
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

B.4.8 XMLPOINTER-function

Returns a BINARY(4) value that is a pointer to a LOB (Large Object) that holds the serialized value of **XML-value-expression**.

The XMLPOINTER function is used in programs that need to process serialized XML values. The structure of the LOB is a variable-length storage object. It starts with a signed integer of 32 bit and contains the LOB data length (max 2 GB), followed by the LOB data.

B.4.8.1 Syntax

Notes:

- If **XML-value-expression** is NULL or empty, XMLPOINTER returns a NULL value.
- The storage object of the LOB is allocated from an Advantage CA-IDMS CV storage pool or from the batch address space for local mode programs. The

storage object is only addressable in client programs that run in the same Advantage CA-IDMS CV as the database server or in batch local mode programs.

- The program invoking the XMLPOINTER function must free the storage of the LOB when it is no longer needed. If no free storage is done, the storage associated with the LOB is freed at task termination.
- Client programs that cannot access the LOB returned by XMLPOINTER can use XMLSERIALIZE (returns a maximum of 30,000 characters) or the table procedure SYSCA.XMLSLICE to process XML values.
- XMLPOINTER is not part of the SQL/XML ISO standard. It is an Advantage CA-IDMS extension to facilitate access to XML LOBs.

B.4.8.2 Example

The following statement returns pointers to XML LOB objects:

```
SELECT XMLPOINTER(XMLFOREST(NAME as "Name"
                          , SCHEMA as "Schema")
                 ) AS "PointerToLob"
 FROM  SYSTEM.TABLE
```

where schema = 'DEMOPROJ'

The result is similar to the following:

```
**
** PointerToLob
** ------------
** 20003008
** 20003088
```

B.4.9 XMLROOT-function

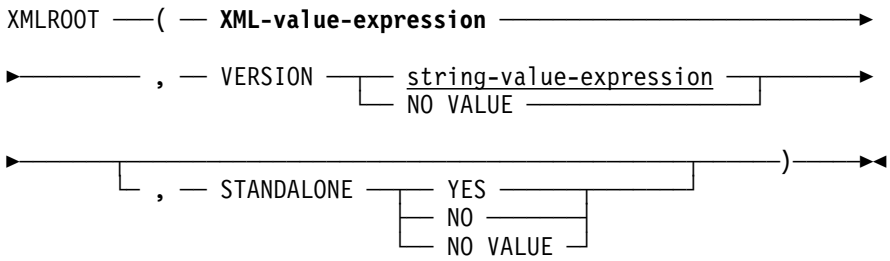
Returns an XML value by modifying the properties of the XML root information item of another XML value.

All XML documents must have at least one well-formed root element. The root element, often called the document tag, must follow the prolog (XML declaration plus DTD) and must be a non-empty tag that encompasses the entire document.

Notes:

- The Encoding property cannot be specified in the XMLROOT function. However, the XMLROOT function sets the value of the property by using the value from the XML ENCODING parameter in the SQL SET SESSION statement, unless the given XML value already has a non-null value for its Encoding property.
- If the input **XML-value-expression** is NULL or empty, XMLROOT returns a NULL value.

B.4.9.1 Syntax



B.4.9.2 Parameters

string-value-expression

Specifies a character string **value-expression**, that is a **value-expression** that returns a value of type character.

B.4.9.3 Example

Use of the XMLROOT function to generate the XML declaration in an XML document.

```
set session XML ENCODING UTF8;

select
  XMLSERIALIZE(CONTENT
    XMLROOT(
      XMLELEMENT(NAME "Employee",
        XMLATTRIBUTES(EMP_ID AS "Id",
                      DEPT_ID AS "DeptId",
                      MANAGER_ID AS "MgrId"),
        TRIM(EMP_FNAME)||' '||trim(EMP_LNAME),
        ' from ', CITY),
      VERSION '1.0', STANDALONE YES)
                        AS VARCHAR(256)) AS "EmployeeData"
  from DEMOEMPL.EMPLOYEE  where EMP_ID = 1003;
```

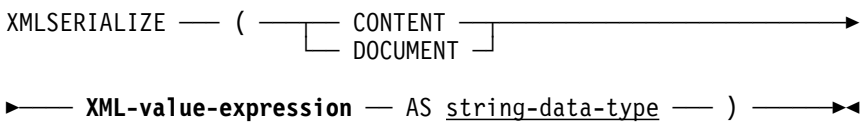
The result is similar to the following. Note that the content of the EmployeeData column has been formatted for convenience:

```
*+ EmployeeData
*+ ------------
*+ <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Employee Id="1003" DeptId="6200">James Baldwin from Boston </Employee>
```

B.4.10 XMLSERIALIZE-function

Returns a value of character string or binary string. Serialization is an operation on an XML value that transforms the XML value in a continous character string representation. Serialization is the inverse operation of parsing.

B.4.10.1 Syntax



B.4.10.2 Parameters

string-data-type

Must be one of the character data types of data type: CHAR(n), CHARACTER(n), VARCHAR(n), CHAR VARYING(n).

Note: The DOCUMENT option is not functional in this feature. This means that CONTENT should always be specified.

B.4.10.3 Example

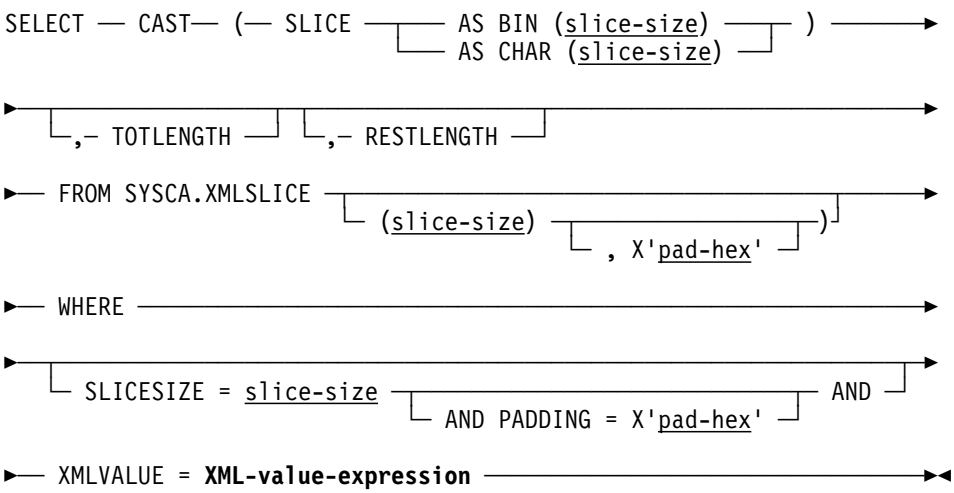
Use of XMLSERIALIZE to serialize an XML value as a character string of 50 characters.

```
 SELECT NAME
      , XMLSERIALIZE(CONTENT
                     XMLELEMENT(NAME "Schema"
                              , XMLATTRIBUTES (NAME AS "Name"
                                             , CUSER AS "User"))
                     AS CHAR(50)) AS "Serialized XML"
  FROM SYSTEM.SCHEMA WHERE NAME IN ('SYSTEM', 'SYSDICT') ;

NAME        Serialized XML
----        --------------
SYSTEM      <Schema Name="SYSTEM" User="DEFJE01"</Schema>
SYSDICT     <Schema Name="SYSDICT" User="DEKDO01"</Schema>
```

B.4.11 XMLSLICE Table Procedure

SYSCA.XMLSLICE is a table procedure used to retrieve character or binary slices of equal length from the serialization of an XML value. It is an Advantage CA-IDMS extension that has been made available to allow any client program to process large serialized XML values. It is used when neither XMLSERIALIZE (limited to 30,000 characters) nor XMLPOINTER (requires client to run in same address space as Advantage CA-IDMS) can be used.

B.4.11.1 Syntax



B.4.11.2 Parameters

TOTLENGTH
The total length of the serialized XML value, without padding characters.

RESTLENGTH
The XML data length, without padding characters, that have not been returned yet.

SYSCA.XMLSLICE
Specifies a table procedure that slices **XML-value-expression**, after serialization, into slices of equal size, specified by *slice-size*. Each row returned by SYSCA.XMLSLICE represents a slice.

slice-size
Specifies an integer **value-expression**, with a positive value <= 8192. A *slice-size* must always be present, either as the first positional parameter of the table procedure or as the right operand in the equal predicate for SLICESIZE.

pad-hex
Specifies an optional two-byte hexadecimal literal that is used to pad the last slice of the serialized XML value. The default depends on the XML ENCODING parameter of the SQL SET SESSION statement. Two spaces are used for EBCDIC (X'4040') and UTF8 (X'2020'); one space is used for UTF16LE (X'2000') and UTF16BE (X'0020').

If **XML-value-expression** is specified as a **subquery**, it must be enclosed in parentheses.

The content of the slice is available in the SLICE column of the table procedure. Optionally, you can specify additional columns in the SELECT statement.

B.4.11.3 Examples

Example 1: In the following SELECT statement, all the employees in DEMOEMP.EMPLOYEE are aggregated in one XML value. This XML value is serialized, and each row returned is a 40-character slice of the serialized XML value.

```
select cast(slice as char(40)) from SYSCA.XMLSLICE
where slicesize = 40
  and xmlvalue =
(
  select xmlelement(name "employee",
                  xmlagg(xmlelement(name "Name",
                                    E.EMP_LNAME)))
  from DEMOEMPL.EMPLOYEE e
)
```

The result is similar to the following:

```
** (EXPR)
** ------
** <employee><Name>Albertini           </Na
** me><Name>Alexander           </Name><Nam
** e>Anderson            </Name><Name>Baldw
** in             </Name><Name>Bennett
**         </Name><Name>Bradley
**  </Name><Name>Brooks              </Name
** ><Name>Carlson             </Name><Name>
** Catlin              </Name><Name>Clark
**              </Name><Name>Courtney
**       </Name><Name>Crane               <
** /Name><Name>Cromwell            </Name><
** Name>Dexter              </Name><Name>Do
** nelson            </Name><Name>Ferguson
**            </Name><Name>Ferndale
**     </Name><Name>Fordman             </N
** ame><Name>Gallway             </Name><Na
** me>Griffin            </Name><Name>Hall
```

Example 2: This example shows the z/OS JCL for the batch command facility IDMSBCF and the SQL statements to create the z/OS dataset "CAIDMS.SAMPLE.XML" holding an XML document encoded in Unicode UTF-16 Little Endian.

The XML document contains the id and name of all employees as present in the DEMOEMPL.EMPLOYEE table.

With a binary file transfer, the dataset can be copied to other platforms for further processing. The use of the XMLSLICE table procedure allows for creating XML documents up to 2 GB.

```
//BCFLOCAL EXEC PGM=IDMSBCF,REGION=7500K
//STEPLIB  DD   DSN=CAIDMS.R160.LOADLIB,DISP=SHR
  ⋮
//SYSLST DD SYSOUT=A
//SYSOUT DD SYSOUT=A
//OUTPUT   DD  DSN=CAIDMS.SAMPLE.XML,DISP=(NEW,CATLG),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=16000),
//             UNIT=SYSDA,SPACE=(TRK,(10,10))
//SYSIPT  DD *
set options OUTPUT to OUTPUT;    -- redirects output of BCF
set session XML Encoding utf16LE;-- requests UTF-16 LE encoding
select cast(slice as char(80))
  from sysca.xmlslice(80, X'2000')-- defines slices of 80 bytes
                                  -- padding with space in UTF-16 LE
 where xmlvalue =
(select xmlroot(xmlelement(name "AllEmployees"
                          , xmlagg(xmlelement(name "Emp"
                                 , xmlattributes(EMP_ID as "Id")
                                 , EMP_FNAME ||EMP_LNAME)))
              , version '1.0')
  from DEMOEMPL.EMPLOYEE
)
/*
```

Appendix C. SQL Functions and SQL Procedure Enhancements

C.1 Overview

This appendix:

- Describes the procedures for developing user-defined SQL functions
- Provides examples of SQL functions and procedures written in Advantage CA-ADS
- Provides an example of an SQL function written in COBOL

C.2 When To Use a User-Defined Function

You can use a user-defined SQL function (currently only of scalar type) just as you would use any SQL scalar function. A scalar function is a function whose argument includes zero or more value expressions on which the function operates. The result of a scalar function is a single value. This value is derived from the expression or expressions forming the arguments to the function.

To use this feature, follow the steps below:

- Define the function using the new CREATE FUNCTION statement.
- Write the function in COBOL, PL/I, Assembler, or Advantage CA-ADS following the guidelines outlined below. You may also be able to use an existing program as a function.
- If necessary, define the program to an Advantage CA-IDMS system.
- Invoke the SQL function. You invoke the SQL function in a way very similar to the way in which you invoke built-in functions. The only restriction is that you cannot invoke a user-defined function from within a table's check constraint.

C.3 Defining a Function

You define a function using the CREATE FUNCTION statement. An example is shown below:

```
CREATE FUNCTION JSMITH.UDF_FUNBONUS
       ( EMP_ID         DECIMAL(4)    )
         RETURNS DECIMAL(10)
         EXTERNAL NAME FUNBONUS PROTOCOL IDMS
         DEFAULT DATABASE CURRENT
         USER MODE
         LOCAL WORK AREA 0
         ;
```

Similarly, use the new ALTER FUNCTION and DROP FUNCTION statements to modify and delete the definition of existing functions.

C.3.1.1 For More Information

For information:

- About the syntax and parameters used in defining functions, see CREATE FUNCTION, ALTER FUNCTION, and DROP FUNCTION in Appendix B, “New and Revised SQL Statements.”
- About detailed examples of using CREATE FUNCTION, see C.6, “Advantage CA-ADS SQL Function and Procedure Examples” on page C-10 and C.7, “COBOL SQL Function Example” on page C-14.

C.4 Invoking a Function

User-defined SQL functions are invoked using the user function invocation syntax. See the User Function Invocation topic in Appendix B, “New and Revised SQL Statements.”

Access to user-defined functions is controlled the same way as procedures. GRANT and REVOKE statements on a resource type of TABLE are used to give and remove SELECT or DEFINE privileges on a function.

C.5 Writing a Function

You can write the program associated with a function in COBOL, PL/I, Assembler, or Advantage CA-ADS. When called, the program is passed a fixed parameter list consisting of the parameters specified in the function definition, as well as additional parameters used for communication between Advantage CA-IDMS and the function.

Whenever a function is invoked, Advantage CA-IDMS calls the program associated with the function to service the request. The function responds by processing the input parameters. By setting SQLSTATE appropriately you can optionally indicate an error condition.

Advantage CA-IDMS performs transaction and session management automatically in response to requests that the originating application issues. Changes to the database made by a function are committed or rolled out together with other changes made within the SQL transaction. No special action is required of the function in order to ensure that this occurs.

The following section discusses writing a function in detail.

For an example of a function written in COBOL, see C.7, “COBOL SQL Function Example” on page C-14. For an example of a function written in Advantage CA-ADS, see C.6, “Advantage CA-ADS SQL Function and Procedure Examples” on page C-10.

C.5.1 Calling Arguments

The following sets of arguments are passed when a function is called:

- One argument for each of the parameters specified on the function definition, passed in the order in which the parameters were declared. These arguments vary from function to function; they are used to pass values to the function.
- One argument to contain the return value of the function. The implicit name for this argument is USER_FUNC.
- One argument for each null indicator associated with a parameter specified in the procedure definition, passed in the order in which the parameters were declared. These arguments vary from function to function; they are used to pass values to the function.
- One argument for the null indicator associated with the return value of the function (the null indicator for the USER_FUNC parameter).
- A set of common arguments used for communications between Advantage CA-IDMS and the function. This set of arguments, shown in the table below, is the same for all functions.

Argument	Contents
Result Indicator (fullword)	Not used
SQLSTATE (CHAR (5))	Status code returned by the procedure: The initial value is always 00000. 00000 — Indicates success 01Hxx — Indicates a warning 02000 — Indicates no more rows 38xxx — Indicates an error
Function Name (CHAR (18))	Name of the function
Explicit Name	Not used
Message Text (CHAR (80))	Message text returned by the function and displayed by Advantage CA-IDMS in the event of an error or warning
SQL Command Code (fullword)	Always 16, indicating a Fetch SQL request
SQL Operation Code (fullword)	Always 16, indicating a "next row" request
Instance Identifier (fullword)	Not meaningful for functions
Local Work Area (user-defined)	A user-defined working storage area
Global Work Area (user-defined)	A user-defined storage area that can be shared by one or more functions or by other SQL routines

C.5.2 Parameter Arguments

On entry to the function, the values of the arguments corresponding to the parameters defined in the CREATE FUNCTION statement are as follows:

- Non-null parameters contain one of the following:
  - The parameter values specified on the function reference
  - The datatype-specific default value if WITH DEFAULT was specified in the function definition, and no value was specified in the function invocation
- All other parameters contain nulls (that is, the null indicator for the parameter is negative).

On exit, the function is expected either to have set the value of the parameter USER_FUNC and the corresponding indicator appropriately or to have set an SQLSTATE value indicating no-more-rows. If the indicator parameter is set to -1, Advantage CA-IDMS ignores the value of the USER_FUNC parameter.

C.5.3 Local Work Area

Another parameter passed on each call to a function is a local work area.

Advantage CA-IDMS allocates the local work area prior to calling the function and frees it immediately after the function exits. When the local work area is allocated, it is initialized to binary zeros.

C.5.4 Global Work Area

A global work area is a storage area that can be shared across one or more functions, or other SQL routines, within a transaction. Each global work area has an associated key that is either:

- The four-character identifier specified on the GLOBAL WORK AREA clause
- The fully-qualified name of the function if no identifier was specified All SQL routines executing within a transaction and having the same global storage key share the same global work area.

C.6 Advantage CA-ADS SQL Function and Procedure Examples

In Release 16.0, an SQL procedure or function can be coded as an Advantage CA-ADS mapless dialog.

C.6.1 Function Example

This example invokes the SQL function ASIND, which returns the arcsine in degrees of the supplied value. The SQL function is implemented using an Advantage CA-ADS dialog that invokes the Advantage CA-ADS built-in function ARCSINE-DEGREES().

C.6.1.1 Function Definition

The SQL function definition is shown below:

```
CREATE FUNCTION JSMITH.ASIND
  (  ARG            DOUBLE PRECISION)
     RETURNS DOUBLE PRECISION
     EXTERNAL NAME ASIND
     PROTOCOL ADS
     SYSTEM MODE
     LOCAL WORK AREA 0
     GLOBAL WORK AREA 0
     ;
```

C.6.1.2 Work Records

To access the function parameters, the Advantage CA-ADS dialog should include these work records:

- <schema>.<function_name>

  Note: This record does not reside in the dictionary; it is built automatically by the Advantage CA-ADS dialog compiler (ADSC or ADSOBCOM) when the dialog is compiled.

- ADSO-SQLPROC-COM-AREA — A system-supplied record. See Chapter 4, “SQL Features” for the record layout.

These work records are included in the ASIND mapless dialog:

- JSMITH.ASIND
- ADSO-SQLPROC-COM-AREA

C.6.1.3 Premap Process

The premap process performs the actions required for the SQL function. The code for the function is provided below:

```
ADD MODULE NAME IS ASIND-PROC VERSION IS 1
LANGUAGE PROCESS
PROCESS SOURCE FOLLOWS
IF ARG LE 1.0
    THEN
      DO.
        MOVE 0 TO USER_FUNC-I
        MOVE ARCSINE-DEGREES(ARG) TO USER_FUNC
      END.
     ELSE
       D0.
         MOVE '38099' TO SQLPROC-SQLSTATE.
         MOVE 'Arg must be <= 1.0' to SQLPROC-MSG-TEXT.
       END.
      LEAVE ADS.
MSEND;
```

C.6.1.4 Invoking the Function

The SELECT clause is used to invoke the function. The first example illustrates a correctly executing function:

```
SELECT JSMITH.ASIND (1)
  FROM SYSTEM.TABLE WHERE NAME = 'ASIND'
*+
*+                        USER_FUNC
*+                        ---------
*+  9.0000000000000000E+01
*+
*+ 1 row processed
```

The second example illustrates a function that results in an error message:

```
SELECT JSMITH.ASIND (2)
 FROM SYSTEM.TABLE WHERE NAME = 'ASIND'
*+ Status = -4    SQLSTATE = 38000      Messages follow:
*+ DB001075 C-4M321: Table Procedure ASIND exception 38099 ARG MUST BE <= 1.0
```

C.6.2 Procedure Example

The following SQL procedure, GET_PROC_AREA, writes any supplied message to a global area. The contents of the global area are shown when no input is supplied. The procedure definition is given below:

```
CREATE PROCEDURE JSMITH.GET_PROC_AREA
  ( IN_AREA            CHARACTER (25),
    GLOBAL_AREA        CHARACTER (25)
  )
    EXTERNAL NAME GETPAREA
    PROTOCOL ADS
    SYSTEM MODE
    LOCAL WORK AREA 0
    GLOBAL WORK AREA 25 KEY GGLA
    ;
```

C.6.2.1 Work Records

To access the procedure parameters, the Advantage CA-ADS dialog should include these work records:

- <schema>.<function_name>

  Note: This record does not reside in the dictionary; it is built automatically by the Advantage CA-ADS dialog compiler (ADSC or ADSOBCOM) when the dialog is compiled.

- A global work area similar to the one listed below:

```
ADD RECORD NAME GETPAREA-SQLPROC-GLOBAL-AREA.
    03 AREA-C                    PIC   X OCCURS 25.
```

The work records included in the mapless dialog GETPAREA are given below:

- JSMITH.GET_PROC_AREA
- GETPAREA-SQLPROC-GLOBAL-AREA

C.6.2.2 Premap Process

The premap process performs the actions of the SQL procedure. The premap process for the sample procedure is given below:

```
ADD
 PROCESS NAME IS GETPAREA_PROC VERSION IS 1
     PUBLIC ACCESS IS ALLOWED FOR ALL
     PROCESS SOURCE FOLLOWS
IF IN-AREA-I GE 0
  THEN
    DO.
       MOVE 0 TO GLOBAL-AREA-I.
       MOVE IN-AREA TO GLOBAL-AREA.
       MOVE IN-AREA TO GETPAREA-SQLPROC-GLOBAL-AREA.
       MOVE 'WRITING TO GLOBAL AREA' TO IN-AREA.
    END.
   ELSE
     DO.
       M0VE 0 TO IN-AREA-I.
       MOVE 'READING FROM GLOBAL-AREA' TO IN-AREA.
       M0VE 0 TO GLOBAL-AREA-I.
       MOVE GETPAREA-SQLPROC-GLOBAL-AREA TO GLOBAL-AREA.
     END.
LEAVE ADS.
```

```
MSEND
.

 .
```

C.6.2.3 Procedure Invocation

The GET_PROC_AREA invocation is given below. The first example illustrates writing to the global area:

```
CALL JSMITH.GET_PROC_AREA ('HELLO FROM ADS DIALOG');
*+
*+ IN_AREA                      GLOBAL_AREA
*+ -------                      -----------
*+ WRITING TO GLOBAL AREA       HELLO FROM ADS DIALOG
*+
*+ 1 row processed
```

The second example illustrates reading from the global area:

```
CALL JSMITH.GET_PROC_AREA ();
*+
*+ IN_AREA                      GLOBAL_AREA
*+ -------                      -----------
*+ READING FROM GLOBAL_AREA     HELLO FROM ADS DIALOG
*+
*+ 1 row processed
```

C.7 COBOL SQL Function Example

This section contains:

- A sample SQL function definition.
- A sample SQL function written in COBOL. It requires the employee demo database and assumes use of a VS COBOL II compiler.
- An example of the SQL function invocation.

C.7.1 Function Definition

The example below illustrates an SQL function definition:

```
CREATE FUNCTION FIN.UDF_FUNBONUS
  ( F_EMP_ID                          DECIMAL(4)
  )
    RETURNS DECIMAL(10)
    EXTERNAL NAME FUNBONUS PROTOCOL IDMS
    DEFAULT DATABASE CURRENT
    USER MODE
    LOCAL WORK AREA 0
    ;
```

C.7.2 Sample COBOL Code

The sample program shown below is included on the Advantage CA-IDMS installation tape. This program requires the SQL employee demo database.

```
*COBOL PGM SOURCE FOR FUNBONUS
*RETRIEVAL
*DMLIST
 IDENTIFICATION DIVISION.
 PROGRAM-ID.                    FUNBONUS.
 AUTHOR.                        JSMITH.
 INSTALLATION.                  SYSTEM71.
 DATE-WRITTEN.                  01/02/2003.
*-----------------------------------------------------------------*
*                                                                 *
* CA-IDMS SQL                   16.0                              *
*                                                                 *
* FUNBONUS implements the SQL function FUNBONUS                   *
*                                                                 *
*-----------------------------------------------------------------*
 ENVIRONMENT DIVISION.
*
 CONFIGURATION SECTION.
*SOURCE-COMPUTER.               IBM WITH DEBUGGING MODE.
*
 DATA DIVISION.
*
 WORKING-STORAGE SECTION.
*-----------------------------------------------------------------*
*                                                                 *
*-----------------------------------------------------------------*
```

```
 LINKAGE SECTION.
  EXEC SQL BEGIN DECLARE SECTION END-EXEC.
  EXEC SQL
    INCLUDE TABLE FIN.UDF_FUNBONUS NO STRUCTURE
  END-EXEC.
  EXEC SQL END DECLARE SECTION END-EXEC.
  77 RESULT-IND                          PIC S9(04) COMP SYNC.
  01 FUN-SQLSTATE.
     02 FUN-SQLSTATE-CLASS               PIC X(02).
     02 FUN-SQLSTATE-SUBCLASS            PIC X(03).
*-----------------------------------------------------------------*
 PROCEDURE DIVISION USING F-EMP-ID
                        , USER-FUNC
                        , F-EMP-ID-I
                        , USER-FUNC-I
                        , RESULT-IND
                        , FUN-SQLSTATE.
  0000-MAINLINE.
     IF F-EMP-ID-I NOT < 0
       THEN
         EXEC SQL
           SELECT SUM(BONUS_AMOUNT) INTO :USER-FUNC
             FROM DEMOEMPL.BENEFITS
            WHERE EMP_ID = :F-EMP-ID
         END-EXEC

                IF SQLSTATE NOT = '00000'
            MOVE -1 TO USER-FUNC-I
            MOVE '38901' TO FUN-SQLSTATE
         ELSE
            MOVE 0 TO USER-FUNC-I
      ELSE
        MOVE -1 TO USER-FUNC-I
        MOVE '38902' TO FUN-SQLSTATE.
  EXIT PROGRAM.
  STOP RUN.
```

C.7.3 Invoking the Function

The example below illustrates invoking the SQL function defined earlier:

```
SELECT EMP_ID, FIN.UDF_FUNBONUS (EMP_ID   )
  FROM DEMOEMPL.EMPLOYEE
WHERE EMP_ID = 3411
*+
*+ EMP_ID     USER_FUNC
*+ ------     ---------
*+   3411          5100
*+
*+ 1 row processed
```

Appendix D. SQL ROWID Examples

D.1 Overview

This appendix provides several examples of how to use ROWID.

D.2 ROWID in a Simple SELECT

This example illustrates using ROWID in a simple SELECT statement:

```
SELECT ROWID, OFFICE_CODE_0450, OFFICE_CITY_0450
  FROM EMPSCHM.OFFICE;
*+
*+     ROWID     OFFICE_CODE_0450  OFFICE_CITY_0450
*+     --------  ----------------  ----------------
*+  X'01259701'  002               BOSTON
*+  X'0125A001'  001               SPRINGFIELD
*+  X'0125A301'  005               GLASSTER
*+  X'0125A601'  012               CAMBRIDGE
*+  X'0125A901'  008               WESTON
*+
*+ 5 rows processed
```

D.3 ROWID in a Searched UPDATE

This example illustrates using ROWID in a WHERE clause of an UPDATE statement:

```
UPDATE EMPNSQL.EMPLOYEE SET EMP_CITY = 'BRUSSELS'
  WHERE ROWID = X'0124FF01';
*+ Status = 0        SQLSTATE = 00000
*+ 1 row processed
```

D.4 ROWID in a SELECT Using a Join

These examples illustrate using a SELECT statement that uses a join between a base table and a view. They show that the ROWID of a view is the ROWID of the first component in the view.

D.4.1 Example 1

In this example, the returned ROWID for the view is the ROWID of the EMPSCHM.OFFICE base table:

```
DROP   VIEW JSMITH.EMPOFFV;
*+ Status = 0        SQLSTATE = 00000
CREATE VIEW JSMITH.EMPOFFV
 AS SELECT EV.*, O.*
  FROM EMPSCHM.OFFICE O, JSMITH.EMPLOYEEV EV
 WHERE "OFFICE-EMPLOYEE";
*+ Status = 0        SQLSTATE = 00000
SELECT EOV.ROWID, D.ROWID, D.*, EMP_ID, OFFICE_CODE_0450
  FROM JSMITH.EMPOFFV EOV, EMPSCHM.DEPARTMENT D
 WHERE "DEPT-EMPLOYEE" AND EMP_ID < 5;
*+
*+     ROWID        ROWID      DEPT_ID_0410
*+     --------      --------  ------------
*+  X'0125A001'   X'0125BD01'           100
*+  X'0125A001'   X'0125BC01'          3100
*+  X'0125A001'   X'0125AB01'          3200
*+
*+ DEPT_NAME_0410                                  DEPT_HEAD_ID_0410  EMP_ID
*+ --------------                                  -----------------  ------
*+ EXECUTIVE ADMINISTRATION                                       30       1
*+ INTERNAL SOFTWARE                                               3       3
*+ COMPUTER OPERATIONS                                             4       4
*+
*+ OFFICE_CODE_0450
*+ ----------------
*+ 001
*+ 001
*+ 001
*+
*+ 3 rows processed
```

D.4.2 Example 2

In the following example, the returned ROWID for the view is the ROWID of the EMPSCHM.EMPLOYEE base table:

```
DROP   VIEW JSMITH.EMPOFFV;
*+ Status = 0        SQLSTATE = 00000
CREATE VIEW JSMITH.EMPOFFV
 AS SELECT EV.*, O.*
  FROM JSMITH.EMPLOYEEV EV, EMPSCHM.OFFICE O
 WHERE "OFFICE-EMPLOYEE";
*+ Status = 0        SQLSTATE = 00000
SELECT EOV.ROWID, D.ROWID, D.*, EMP_ID, OFFICE_CODE_0450
  FROM JSMITH.EMPOFFV EOV, EMPSCHM.DEPARTMENT D
 WHERE "DEPT-EMPLOYEE" AND EMP_ID < 5;
*+
*+     ROWID         ROWID     DEPT_ID_0410
*+     --------      --------  ------------
*+  X'01252801'   X'0125BD01'           100
*+  X'01253B01'   X'0125BC01'          3100
*+  X'01255301'   X'0125AB01'          3200
*+
*+ DEPT_NAME_0410                                  DEPT_HEAD_ID_0410  EMP_ID
*+ --------------                                  -----------------  ------
*+ EXECUTIVE ADMINISTRATION                                       30       1
*+ INTERNAL SOFTWARE                                               3       3
*+ COMPUTER OPERATIONS                                             4       4
*+
*+ OFFICE_CODE_0450
*+ ----------------
*+ 001
*+ 001
*+ 001
*+
*+ 3 rows processed
```

D.5 Searched Update of Records Without Primary Key

This example updates all the COVERAGE records of the employee with EMP_ID = 23:

```
UPDATE EMPSCHM.COVERAGE C
SET SELECTION_YEAR_0400 = 20
 WHERE C.ROWID IN (
          SELECT CI.ROWID
     FROM EMPSCHM.EMPLOYEE E, EMPSCHM.COVERAGE CI
    WHERE "EMP-COVERAGE"
      AND EMP_ID = 23);
*+ Status = 0        SQLSTATE = 00000
*+ 2 rows processed
```

D.6 Searched Delete of Records Without Primary Key

This example deletes all the COVERAGE records of the employee with EMP_ID = 23:

```
DELETE FROM EMPSCHM.COVERAGE C
 WHERE C.ROWID IN (
        SELECT CI.ROWID
  FROM EMPSCHM.EMPLOYEE E, EMPSCHM.COVERAGE CI
 WHERE "EMP-COVERAGE"
   AND EMP_ID = 23);
*+ Status = 0        SQLSTATE = 00000
*+ 2 rows processed
```

Appendix E. SQL Cache Tables

E.1 Overview

This appendix describes the tables (actually table procedures) that are used for displaying and controlling the SQL cache. It also provides some examples of how the DBA can display and control the cache. The SQL cache is used in conjunction with the dynamic SQL statement caching feature. Dynamic SQL statement caching is explained in 4.2, “Dynamic SQL Caching” on page 4-4.

E.2 Tables for Viewing, Monitoring, and Controlling the Cache

SQL is the Application Programming Interface (API) used to view, monitor, and change the cache and the cache configuration. This means that cache administration, configuration, and dynamic SQL cache monitoring is available in any environment that supports Advantage CA-IDMS SQL, such as IDMSBCF, OCF, Advantage CA-IDMS Visual DBA, and Advantage CA-IDMS SQL programs, among others.

This section describes the SYSCA tables (specifically, three table procedures and one view) defined for dynamic SQL cache management.

E.2.1 DSCCACHEOPT

The DSCCACHEOPT table manages the SQL cache options.

Column	Data Type	Description
CACHEMAXCNT	INTEGER	The maximum number of entries that the cache can contain.
DEFAULT	CHAR(4)	Default for caching: ON/OFF. This specifies if caching is enabled or disabled for any connect name that does not appear in the EXCEPTCON column.
EXCEPTCNT	INTEGER	Count of rows in the DSCCACHEOPT relation with non-NULL value for the EXCEPTCON column. It is the number of connect names in the list of exceptions.
EXCEPTCON	CHAR(8)	Connect name that forms an exception to the default caching.

E.2.1.1 Notes

- After startup of central version, DSCCACHEOPT reflects the parameters of the sysgen SQL CACHE statement. In absence of an SQL CACHE statement there are no rows in DSCCACHEOPT and SQL caching is disabled, but can be activated by inserting a DSCACHEOPT row. Updates to the DSCCACHEOPT table have no impact on the CV's sysgen.
- In local mode when no DSCCACHEOPT row exists, a DSCCACHEOPT row is automatically inserted with values derived from the SYSIDMS parameter SQL_CACHE_ENTRIES.
- There can be 0 to n rows in this table. If there are 0 rows, this means that SQL statement caching is not active and not defined to the system. If there are rows, then the first row contains non-NULL values for CACHEMAXCNT, DEFAULT and EXCEPTCNT and a NULL value for EXCEPTCON. The first row contains the main SQL cache parameters. Other rows in the DSCCACHEOPT relation

contain only non-NULL values for the EXCEPTCON column. These rows form the list of exception connect names.

- You can issue select, insert, update and delete commands against DSCCACHEOPT.
- Deleting the first row automatically deletes all other rows and removes all SQL cache structures from the system, effectively disabling caching until a new DSCCACHEOPT row is inserted. Deleting other rows removes exception connect names from the exception list.
- Inserting a row is always possible. When one or more rows exist, an insert can only specify a value for EXCEPTCON, this is the way to add connect names to the list. When no rows exist, the first insert must specify values for CACHEMAXCNT and DEFAULT. Other values are not allowed. A successful insertion of the first row enables SQL caching.
- Updating of CACHEMAXCNT and DEFAULT columns automatically applies to the first row only, so that a WHERE clause is not needed to filter the first row. When CACHEMAXCNT is decreased, the entries in the SQL cache with the highest AGE (see the description of the DSCCACHE table) are removed. Increase CACHEMAXCNT to enlarge the size of the cache. You cannot update EXCEPTCON for the first row. You cannot update EXCEPTCNT as this is automatically calculated.
- The size of the cache is specified in terms of number of entries. Each entry represents a single cached statement. The cache is allocated from the storage pool within a central version and from operating system storage in local mode. By selecting from the DSSCACHECTRL table you can determine the amount of storage being consumed.

E.2.2 DSCCACHECTRL

The DSCCACHECTRL table controls SQL caching

Column	Data Type	Description
REQUEST	CHAR	Future use
STATUS	CHAR	Future use
CACHEMAXCNT	INTEGER	Maximum count of entries
CACHECURCNT	INTEGER	Current count of entries used
CURRENT	INTEGER	Current entry
OLDEST	INTEGER	Oldest entry
STORAGEUSEKB	INTEGER	Total storage used by the cache

E.2.2.1 Notes

- There can be 0 or 1 row in this table. If no rows are present, no SQL statements have been cached.
- You can only issue SELECT and DELETE statements against this table.
- Deleting the row in DSCCACHECTRL clears the SQL cache structures. It does not disable caching, which is controlled through the DSSCACHEOPT table.

E.2.3 DSCCACHE

The DSCCACHE table represents the SQL cache. Each row is a cache entry.

Column	Data Type	Description
KEY	INTEGER	Non-unique key
LOCK	BINARY(4)	Lock word for access to entry
DBNAME	CHAR(8)	DBNAME of SQL session
DEFAULTSCHEMA	CHAR(18)	Default schema of session if statement contains at least one unqualified table reference
USECNT	INTEGER	Usage count
AGE	INTEGER	Age: A valued used in determining which entry to purge from a full cache when a new entry is inserted. The longer an entry has remained in the cache without being used, the higher is age.
COMPILECOST	INTEGER	Compilation cost
ACCPLANSCANCOST	FLOAT	Cost of scan in access plan
ACCPLANCPUCOST	FLOAT	Cost of CPU in access plan
ACCPLANROWCNT	FLOAT	Count of rows in access plan
EXECCOST	INTEGER	Cost of last execution of statement
COMPILECNT	INTEGER	Count of (re)compilations
COMPILESTAMP	TIMESTAMP	Timestamp of compilation
STMTSIZE	INTEGER	Size of statement
STATEMENT	VARCHAR(8192)	Statement
SQLDIBSIZE	INTEGER	Size of SQLDIB
SQLCMD	INTEGER	Type of SQL command
SQLITCL	INTEGER	Combined Itree/TELL table length
SQLARG	INTEGER	Bit flags for argument usage
SQLOPT	INTEGER	Session options flags

Column	Data Type	Description
SQLTBL	INTEGER	Length of tuple buffer row
SQLPBL	INTEGER	Length of parameter buffer
SQLCID	INTEGER	Cursor identifier
SQLSID	INTEGER	Section identifier
SQLNM1	CHAR(32)	Literal value 1
SQLNM2	CHAR(32)	Literal value 2
SQLITL	INTEGER	Size of Itree
SQLITBADDR	BINARY(4)	Address of Itree
RTREESIZE	INTEGER	Size of Rtree
RTREEOFFSET	INTEGER	Offset of Rtree for relocation purposes
RTREEDOFAOFF	INTEGER	Offset of DOFA in Rtree
RTREEADDR	BINARY(4)	Address of Rtree
FIBSIZE	INTEGER	Size of FIB
FIBADDR	BINARY(4)	Address of FIB
FOPSIZE	INTEGER	Size of FOP
GSTSIZE	INTEGER	Size of GST
FOPADDR	BINARY(4)	Address of FOP
LASTUSER	CHAR(8)	Reserved
GLOBALCURSORNAME	CHAR(18)	Reserved
FCRC	BINARY(4)	FCRC flags
SQLDAADDR	BINARY(4)	Address of cached input SQLDA

E.2.3.1 Notes

- One row of this table represents one cached statement.
- Rows cannot be inserted or updated.
- Because of the size of the STATEMENT column in DSCCACHE and because many of these columns are for internal use only, it is advisable to use a view on this table procedure. The supplied DSCCACHEV view below is an example of such a view.

The following acronyms are used in the table above.

- Itree — A data structure that contains the internal input representation of an SQL statement

- Rtree — A data structure that contains the internal runtime instruction of an SQL statement. The SQL runtime engine IDMSHLDB uses the Rtree.
- FIB — A data structure that contains runtime metadata.
- FOB/FOP — FIB objects list data structure
- GST — Global Security Table
- FCRC — Fixed part of Compiled Relational Command data structure
- SQLDA — The SQL Descriptor Area (SQLDA) is a data structure used to describe variable data passed as part of a dynamic SQL statement.

E.2.4 DSCCACHEV

SYSCA.DSCCACHEV is created during installation. It defines a view on the SYSCA.DSCCACHE table procedure as follows:

```
create view SYSCA.DSCCACHEV as
 select KEY, DBNAME, DEFAULTSCHEMA, USECNT, AGE
      , COMPILECNT as "#C", compilestamp
      , ACCPLANSCANCOST, ACCPLANCPUCOST
      , ACCPLANROWCNT, FIBSIZE, FIBADDR
      , SUBSTR(STATEMENT, 1, 72) as STMT1
 from SYSCA.DSCCACHE;
```

You have the option to define your own views.

E.3 Allowable Operations on DSCCACHE Tables

	DSCCACHOPT Table Procedure	DSCCACHECTRL Table Procedure	DSCCACHE Table Procedure	DSCCACHEV View
SELECT	X	X	X	X
INSERT	X			
UPDATE	X			
DELETE	X	X	X	X

E.4 Examples of Displaying and Controlling the Cache

E.4.1 CACHE Options

To display the cache options:

```
Select * from SYSCA.DSCCACHEOPT;
*+
*+  CACHEMAXCNT  DEFAULT    EXCEPTCNT  EXCEPTCON
*+  -----------  -------    ---------  ---------
*+         1000  OFF                2  <null>
*+       <null>  <null>         <null>  SYSTEM
*+       <null>  <null>         <null>  APPLDICT
```

To change the default for caching:

```
Update SYSCA.DSCCACHEOPT set DEFAULT = 'ON';
```

To add the dictionary 'TSTDICT' to the exception list:

```
Insert into SYSCA.DSCCACHEOPT (EXCEPTCON) values ('TSTDICT');
```

To remove the connect name 'SYSTEM' from the exception list:

```
Delete from SYSCA.DSCCACHEOPT where EXCEPTCON = 'SYSTEM';
```

To remove all the connect names from the exception list:

```
Delete from SYSCA.DSCCACHEOPT where EXCEPTCON is not null;
```

To decrease the number of entries in the cache from 1000 to 5:

```
Update SYSCA.DSCCACHEOPT set CACHEMAXCNT = 5;
```

Only the last 5 used entries are kept in the cache.

To increase the number of entries in the cache from 5 to 9999:

```
Update SYSCA.DSCCACHEOPTset CACHEMAXCNT = 9999;
```

The cache is extended with 9994 new slots.

To clear the SQL cache and remove all the SQL cache structures from the system, effectively disallowing any SQL caching:

```
Delete from SYSCA.DSCCACHEOPT;
```

To rebuild the SQL cache environment or to build the SQL cache environment in a system that has no SQL CACHE statement in its sysgen:

```
Insert into SYSCA.DSCCACHEOPT (CACHEMAXCNT, DEFAULT) values (1000, 'ON');
Insert into SYSCA.DSCCACHEOPT (EXCEPTCON) values ('APPLDICT');
Insert into SYSCA.DSCCACHEOPT (EXCEPTCON) values ('SYSTEM');
```

E.4.2 CACHE Control Parameters

To display cache control parameters:

```
Select * from SYSCA.DSCCACHECTRL;
*+
*+REQUEST  STATUS  CACHEMAXCNT  CACHECURCNT  CURRENT OLDEST
*+-------  ------  -----------  -----------  ------- ------
*+  A              1000                   7        6      0
*+
*+ STORAGEUSEKB
*+ ------------
*+           138
```

To clear the cache, but allow caching to continue as defined by the option in DSCCACHEOPT:

```
Delete from SYSCA.DSCCACHECTRL;
```

E.4.3 CACHE Entries

To display key columns of all cache entries:

```
Select * from SYSCA.DSCCACHEV;
*+
*+  KEY  DBNAME    DEFAULTSCHEMA     USECNT          AGE
*+  ---  ------    -------------     ------          ---
*+   29  SYSDICT   <null>                 4            0
*+   32  SYSDICT   <null>                 1            1
*+   28  SYSDICT   <null>                 2            1
*+   32  SYSDICT   <null>                 7            7
*+   29  SYSDICT   <null>                 6            6
*+
*+     #C  COMPILESTAMP                  FIBSIZE  FIBADDR
*+     --  ------------                  -------  --------
*+     1  2002-09-04-10.05.20.740186         736  12AC6208
*+     1  2002-09-04-10.07.20.009275        2528  12ACD088
*+     1  2002-09-04-10.06.19.785231        2580  12ACB888
*+     1  2002-09-04-10.02.39.729463         552  12AC0A08
*+     1  2002-09-04-10.03.00.735305         736  12ABFD88
*+
*+ STMT1
*+ -----
*+ Select * from SYSCA.DSCCACHEV
*+ select * from empnsql.department
*+ select * from empnsql.office
*+ select * from SYSCA.DSCCACHECTRL
*+ select * from sysca.dsccachev
```

To display cache entries with AGE > 1:

```
Select * from SYSCA.DSCCACHEV where AGE > 1;
```

To display cache entries for DBNAME SYSDICT:

```
Select * from SYSCA.DSCCACHEV where DBNAME = 'SYSDICT';
```

To display cache entries for statements that use schema EMPNSQL:

```
Select * from SYSCA.DSCCACHEV where STMT1 like '%EMPNSQL.%';
```

To remove cache entries that use schema EMPNSQL:

```
Delete from SYSCA.DSCCACHE where STATEMENT like '%empnsql.%';
```

E.5 Secure the Display and Changes

To secure the display of and any changes to SQL caching, the DSCCACHE tables (table procedures and views) must be secured using the standard Advantage CA-IDMS security mechanism.

Note: The SQL cache contains SQL source statements, which might include confidential information.

Appendix F. CICS Interface Enhancements for Two-Phase Commit Support

F.1 Overview

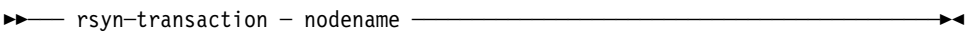
This appendix describes:

- Executing a resynchronization task in order to resynchronize with a back-end central version.
- Creating the resynchronization program.
- Defining the resynchronization transaction to CICS
- New CICSOPT and IDMSCINT parameters
- CICS OPTIXIT Example

F.2 Resynchronization Task Execution

The following syntax is used to execute a resynchronization task:

F.2.1 Syntax



F.2.2 Parameters

rsyn-transaction
The name of a CICS resynchronization transaction defined to the CICS system.

Nodename
The name of the Advantage CA-IDMS central version for which resynchronization is to be performed. The identified system must be accessible through the CICS interface for which the resynchronization transaction was defined.

F.2.3 Examples

F.2.3.1 Successful Manual Resynchronization Example

The following example shows how manual resynchronization is initiated with central version SYSTEM74 using a resynchronization task called RSYN whose interface module is named IDMSINTC. The resulting messages identify the target node name, the name of the interface module being used, the number of incomplete units of work that need to be recovered and the final outcome of the resynchronization process.

```
RSYN SYSTEM74

CA-IDMS Manual 2-PC Resync for IDMSINTC for CV node SYSTEM74 date 10/14/2003
      1 CA-IDMS in doubt units of work need recovery for CV node SYSTEM74
      1 CA-IDMS in doubt units of work recovery started for CV node SYSTEM74
CA-IDMS Two Phase Commit Resync startup completed for CV node SYSTEM74
```

F.2.3.2 Unsuccessful Manual Resynchronization Example 1

This example shows an error condition that occurred during a manual resynchronization because the central version nodename was not specified.

```
RSYN

IDMSCSYN error - CV node not specified
```

F.2.3.3 Unsuccessful Manual Resynchronization Example 2

This example depicts an error condition that occurred during a manual resynchronization because the central version nodename that was specified was not available through the CICS interface for which the resynchronization was defined.

```
RSYN SYSTEM81

CA-IDMS Manual 2-PC Resync for IDMSINTC for CV node SYSTEM81 date 10/14/2003
IDMSCSYN error - Requested CV node SYSTEM81 - Connected CV node SYSTEM74
CA-IDMS Two Phase Commit Resync aborted
```

F.2.3.4 Successful Automatic Resynchronization Example

The following example shows the output from an automatic resynchronization initiated when the first request is made to a back-end central version through a CICS interface module or when the interface is started in a CICS Transaction Server for OS/390 V1R1 (or later) for z/OS or OS/390.

```
CA-IDMS   Auto 2-PC Resync for IDMSINTC for CV node SYSTEM74 date 10/14/2003
      1 CA-IDMS in doubt units of work need recovery for CV node SYSTEM74
      1 CA-IDMS in doubt units of work recovery started for CV node SYSTEM74
CA-IDMS Two Phase Commit Resync startup completed for CV node SYSTEM74
```

F.2.4 Creating the Resynchronization Program

Linking the IDMSCSYN module with an IDMSCINT module creates the resynchronization program. A separate resynchronization program must be created for each version of the Advantage CA-IDMS interface module (IDMSINTC) that is used within a given CICS system.

F.2.5 Resynchronization Program Link Edit (z/OS and OS/390)

```
 //*-------------------------------------------------------------------
//*           LINK IEWL
//*-------------------------------------------------------------------
//LINK    EXEC PGM=IEWL,
//        PARM='LET,LIST,XREF,RENT',
//        REGION=128K,
//        COND=(8,LT,ASMSTEP)
//SYSLMOD      DD  DSN=idms.loadlib,DISP=SHR
//SYSPRINT     DD  SYSOUT=*
//SYSUT1       DD  DSN=&&SYSUT1,
//             UNIT=SYSDA,
//             SPACE=(6400,(80)),
//             DISP=(NEW,PASS)
//IN1          DD  DSN=idms.distload,DISP=SHR
//IN2          DD  DSN=user.objlib,DISP=SHR
//IN3          DD  DSN=cics.loadlib,DISP=SHR
```

```
//SYSLIN       DD  DDNAME=SYSIN
//SYSIN        DD  *
 ORDER DFHEAI
 INCLUDE IN3(DFHEAI)
 INCLUDE IN1(IDMSCSYN)
 INCLUDE IN2(idmscint)
 INCLUDE IN3(DFHEAIO)

 ENTRY CSYNEP1
 MODE AMODE(31),RMODE(ANY)
 NAME usercsyn(R)

//*----------------- ------------------ ------------------ --------
```

Field	Description
cics.loadlib	Data set name of the CICS load library
idms.distload	Data set name of the Advantage CA-IDMS SMP/E distribution load library
idms.loadlib	Data set name of the Advantage CA-IDMS loadlib
user.objlib	Data set name of the user object library containing the idmscint module.
idmscint	Name of the idmscint object module
usercsyn	User specified name of the RSYN load module

F.2.6 Resynchronization Program Link Edit (VSE/ESA)

```
// DLBL   idmslib,
// EXTENT ,nnnnnn
// LIBDEF *,SEARCH=(idmslib.sublib,user.sublib,cicslib.sublib)
// LIBDEF PHASE,CATALOG=idmslib.sublib
// OPTION CATAL
  PHASE   usercsyn,*
  INCLUDE DFHEAI
  INCLUDE  IDMSCSYN
  INCLUDE  idmscint
  INCLUDE  DFHEAIO
    ENTRY    CSYNEP1
// EXEC LNKEDT,SIZE=128K
/*
```

Field	Description
cicslib.sublib	Name of the sublibrary within the library containing CICS modules
idmslib	Filename of the file containing the Advantage CA-IDMS modules
idmslib.sublib	Name of the sublibrary within the library containing Advantage CA-IDMS modules

Field	Description
nnnnnn	Volume serial identifier of the appropriate disk volume
user.sublib	Name of the user object library where the idmscint module resides
idmscint	Name of the idmscint object module
usercsyn	User specified name of the RSYN load module

F.2.7 Defining a Resynchronization Transaction

A resynchronization transaction must be defined for each IDMSINTC interface to be used within a CICS system. Define the resynchronization transaction to CICS as follows:

```
DEFINE   TRANSACTION(rsyn-transaction-name) PROGRAM(usercsyn)
       GROUP(IDMSGRP) PROFILE(IDMSPRF)
       TASKDATAKEY(CICS)
```

Where:

rsyn-transaction-name is the name chosen for the resynchronization transaction.

usercsyn is the name chosen for the resynchronization program.

The installation default transaction name is **RSYN**, but another name can be chosen. The name specified in the transaction definition must be identical to the value for the RSYNTXN parameter of the associated interface's CICSOPT macro. Refer to "New IDMSCINT and CICSOPT Parameters" for a description of the RSYNTXN parameter.

F.2.8 Defining the Resynchronization Program

A resynchronization program must be defined for each IDMSINTC interface to be used within a CICS system. Define the resynchronization program to CICS as follows:

```
DEFINE  PROGRAM(usercsyn) GROUP(IDMSGRP) LANGUAGE(ASSEMBLER) CEDF(NO) EXECKEY(CICS)
```

Where:

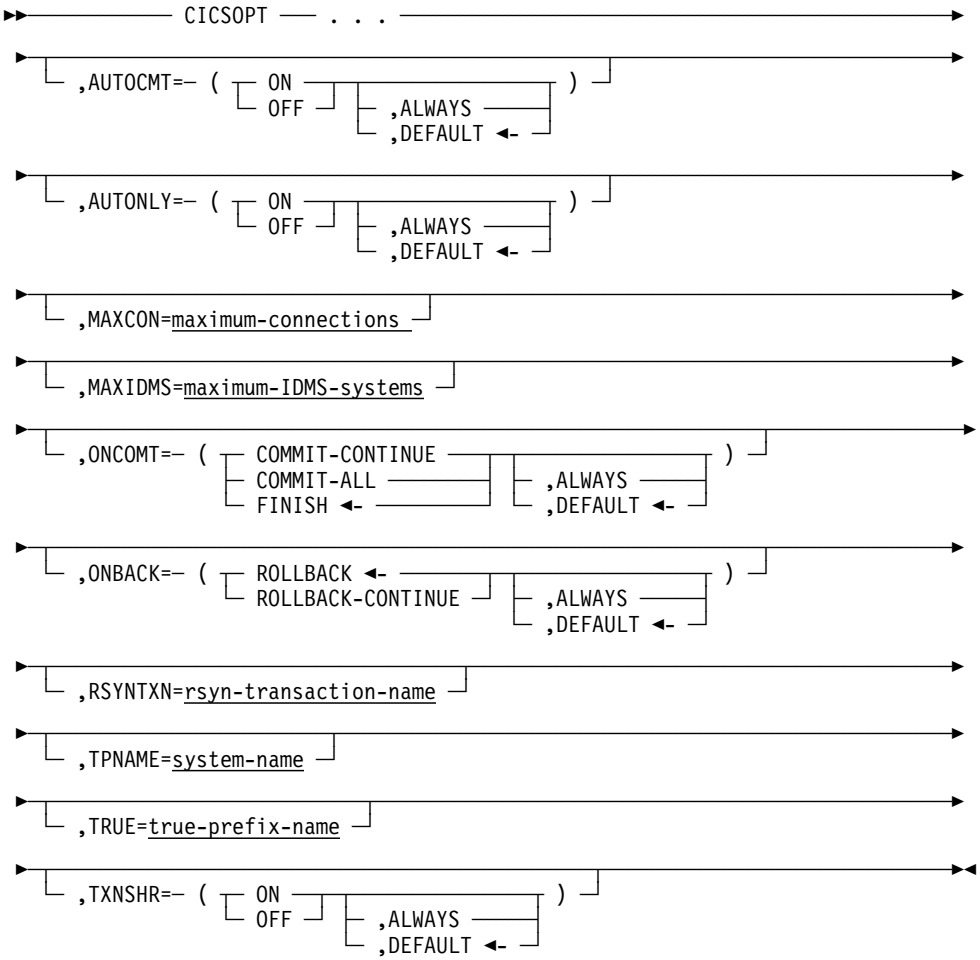
usercsyn is the name chosen for the resynchronization program.

F.3 New CICSOPT and IDMSCINT Parameters

F.3.1 New CICSOPT Parameters

The new or enhanced CICSOPT parameters available in Release 16.0 are described below.

F.3.1.1 Syntax



F.3.1.2 Parameters

AUTOCMT

Specifies whether or not database sessions opened by a program using this interface module are eligible for participation in a CICS UOW (Unit of Work).

ON

Specifies that database sessions are eligible to participate in a CICS Unit Of Work (UOW). If the database session is active at the time a CICS syncpoint

operation is performed, the session's updates are committed as part of the CICS UOW.

OFF
Specifies that database sessions are not eligible to participate in a CICS Unit Of Work (UOW).

If TXNSHR=ON is specified, the default for AUTOCMT is ON; otherwise it is OFF. An assembly error results if TXNSHR=ON and AUTOCMT=OFF are specified.

ALWAYS
Specifies that the AUTOCMT behavior specified in the CICSOPT parameter overrides whatever was specified in the IDMSCINT module with which the application is linked.

DEFAULT
Specifies that the AUTOCMT behavior specified in the CICSOPT parameter applies only if the corresponding IDMSCINT parameter specifies DEFAULT. This is the default.

AUTONLY
Specifies if database sessions opened by a program using this interface module are forced to participate in a CICS UOW.

ON
Specifies that database sessions are forced to participate in a CICS Unit Of Work (UOW). Even if the database session is terminated prior to the CICS syncpoint operation, the session's updates are committed as part of the CICS UOW. DML commands that would normally cause the session's updates to be committed (such as FINISH or COMMIT WORK) have no impact on the session's transaction, although they do impact the session. Conversely, if the session's transaction is forced to back out (either because of a DML ROLLBACK request or because of events such as a deadlock), a CICS SYNCPOINT BACKOUT is issued forcing the entire CICS UOW to be backed out.

OFF
Specifies that database sessions are not forced to participate in a CICS Unit Of Work (UOW).

If TXNSHR=ON is specified, the default for AUTONLY is ON; otherwise it is OFF. An assembly error results if TXNSHR=ON and AUTONLY=OFF are specified.

ALWAYS
Specifies that the AUTONLY behavior specified in the CICSOPT parameter overrides the specifications in the IDMSCINT module with which the application is linked.

DEFAULT
Specifies that the AUTONLY behavior specified in the CICSOPT parameter applies only if the corresponding IDMSCINT parameter specifies DEFAULT. This is the default.

MAXCON

Specifies the maximum number of different back-end central versions that a CICS task can access simultaneously through this CICS interface module. This limit applies only to database sessions for which AUTOCMT is enabled. If an application uses different interface modules, each one has its own limit.

maximum-connections

Must be a numeric value between 1 and 1000. If *maximum-connections* is not specified, the default maximum number of connections is 2.

MAXIDMS

Specifies the maximum number of different back-end central versions that a CICS interface module can access throughout the life of a CICS system. This limit applies only to database sessions for which AUTOCMT is enabled. If an application uses different interface modules, each one has its own limit.

maximum-IDMS-systems

Must be a numeric value between 1 and 1000. The default maximum number of back-end systems is the larger of 2 and 2 * the value of the MAXCVNO parameter.

ONBACK

Specifies the action that should be taken for database sessions opened by a program using this interface module when they participate in a CICS backout operation.

ROLLBACK

Specifies that database sessions should be terminated. This is the default.

ROLLBACK-CONTINUE

Specifies that database sessions should continue but currencies freed.

ALWAYS

Specifies that the ONBACK behavior specified in the CICSOPT parameter overrides whatever was specified in the IDMSCINT module with which the application is linked.

DEFAULT

Specifies that the ONBACK behavior specified in the CICSOPT parameter applies only if the corresponding IDMSCINT parameter specifies DEFAULT. This is the default.

ONCOMT

Specifies the action that should be taken for database sessions opened by a program using this interface module when they participate in a CICS syncpoint operation.

COMMIT-ALL

Specifies that database sessions should continue but currencies freed.

COMMIT-CONTINUE

Specifies that database sessions should continue and currencies retained.

FINISH

Specifies that database sessions should be terminated. This is the default.

ALWAYS
Specifies that the ONCOMT behavior specified in the CICSOPT parameter overrides the specification in the IDMSCINT module with which the application is linked.

DEFAULT
Specifies that the ONCOMT behavior specified in the CICSOPT parameter applies only if the corresponding IDMSCINT parameter specifies DEFAULT. This is the default.

TPNAME
Specifies the name by which DC/UCF will identify all tasks running under this CICS system.

system-name
Specify a four character name.

All interface modules executing within a single CICS system must use the same value for TPNAME. Any attempt to start another occurrence of the interface with a different tpname value than the one specified in the first interface that is currently executing in the CICS system will fail unless a CICS_NAME parameter is specified in the SYSIDMS file included in the CICS startup JCL.

The *system-name* forms the first part of the local transaction ID for database requests and the first four characters of the front-end system ID for external request units. "BULK" is appended to *system-name* to create the front-end system ID. The front-end system ID is used for several purposes:

- Determines the packet size for communications
- Determines the maximum number of simultaneous requests from this CICS system to Advantage CA-IDMS.
- Can also be used as an alternate task code for controlling external request unit processing

If the TPNAME parameter is omitted, the CICS sysid as defined during the CICS system startup becomes the *system-name*.

TRUE
Specifies a prefix to be used in forming Task Related User Exit (TRUE) entry names.

true-prefix
Must be a one to five character value that is unique across all interface modules in use within a CICS system. If *true-prefix* is less than five characters, it is padded on the right with $'s. If not specified, the default prefix is constructed as the last five characters of the IDMSINTC module name, padded on the right with $'s if necessary.

RSYNTXN
Specifies the name of the CICS resynchronization transaction defined for this interface.

rsyn-transaction-name

Must be the name of a transaction defined to CICS and associated with a resynchronization program. If not specified, the default transaction name is RSYN.

TXNSHR

Specifies whether or not database sessions opened by a program using this interface module should share the same transaction as other sessions started by the same CICS task.

ON

Specifies that database sessions should share transactions.

OFF

Specifies that database sessions should not share transactions. This is the default.

ALWAYS

Specifies that the TXNSHR behavior specified in the CICSOPT parameter overrides whatever was specified in the IDMSCINT module with which the application is linked.

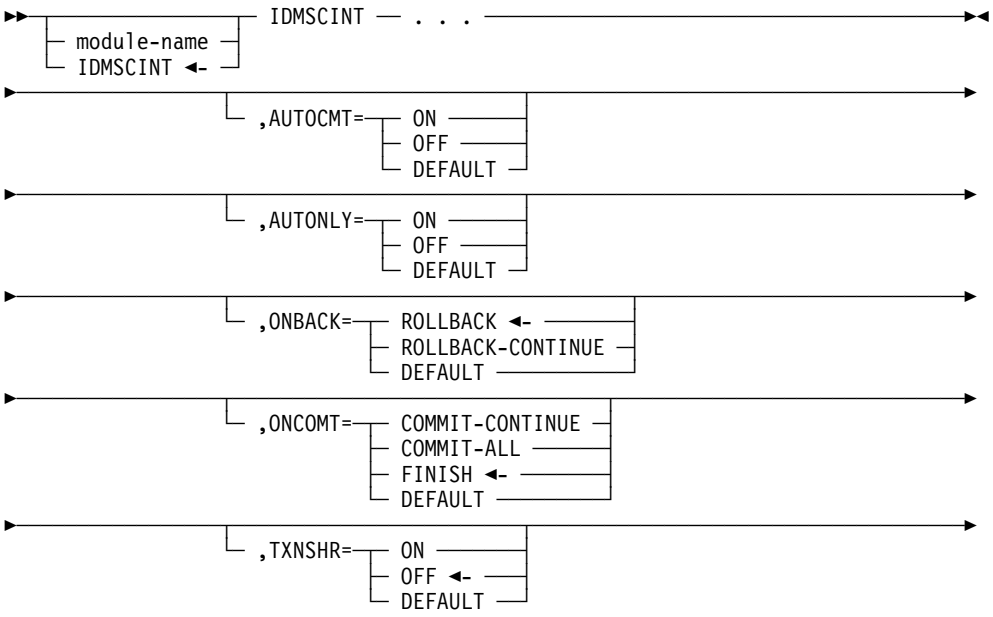
DEFAULT

Specifies that the TXNSHR behavior specified in the CICSOPT parameter applies only if the corresponding IDMSCINT parameter specifies DEFAULT. This is the default.

F.3.2 New IDMSCINT Parameters

The new or enhanced IDMSCINT parameters available in Release 16.0 are described below.

F.3.2.1 Syntax



F.3.2.2 Parameters

AUTOCMT

Specifies whether or not database sessions opened by a program linked with this IDMSCINT module are eligible for participation in a CICS UOW (Unit of Work).

ON

Specifies that database sessions are eligible to participate in a CICS Unit Of Work (UOW). If the database session is active at the time a CICS syncpoint operation is performed, the session's updates are committed as part of the CICS UOW.

OFF

Specifies that database sessions are not eligible to participate in a CICS Unit Of Work (UOW).

DEFAULT

Specifies that whether or not database sessions are eligible for participation in a CICS Unit Of Work (UOW) is determined by the AUTOCMT parameter of the interface's CICSOPT macro.

If TXNSHR=ON is specified, the default for AUTOCMT is ON; otherwise it is OFF. An assembly error results if TXNSHR=ON and AUTOCMT=OFF are specified.

AUTONLY

Specifies whether or not database sessions opened by a program linked with this IDMSCINT module are forced to participate in a CICS UOW.

ON

Specifies that database sessions are forced to participate in a CICS Unit Of Work (UOW). Even if a database session is terminated prior to the CICS syncpoint operation, the session's updates are committed as part of the CICS

UOW. DML commands that would normally cause the session's updates to be committed (such as FINISH or COMMIT WORK) have no impact on the session's transaction, although they do impact the session. Conversely, if the session's transaction is forced to back out (either because of a DML ROLLBACK request or because of events such as a deadlock), a CICS SYNCPOINT BACKOUT is issued forcing the entire CICS UOW to be backed out.

OFF
Specifies that database sessions are not forced to participate in a CICS Unit Of Work (UOW).

DEFAULT
Specifies that whether or not database sessions are forced to participate in a CICS Unit Of Work (UOW) is determined by the AUTONLY parameter of the interface's CICSOPT macro.

If TXNSHR=ON is specified, the default for AUTONLY is ON; otherwise it is OFF. An assembly error results if TXNSHR=ON and AUTONLY=OFF are specified.

ONBACK
Specifies the action that should be taken for database sessions opened by a program linked with this IDMSCINT module when they participate in a CICS backout operation.

ROLLBACK
Specifies that database sessions should be terminated. This is the default.

ROLLBACK-CONTINUE
Specifies that database sessions should continue but currencies freed.

DEFAULT
Specifies that the backout action for sessions is determined by the ONBACK parameter of the interface's CICSOPT macro.

ONCOMT
Specifies the action that should be taken for database sessions opened by a program linked with this IDMSCINT module when they participate in a CICS syncpoint operation.

COMMIT-ALL
Specifies that database sessions should continue but currencies freed.

COMMIT-CONTINUE
Specifies that database sessions should continue and currencies retained.

FINISH
Specifies that database sessions should be terminated. This is the default.

DEFAULT
Specifies that the commit action for sessions be determined by the ONCOMT parameter of the interface's CICSOPT macro.

TXNSHR

Specifies whether or not database sessions opened by a program linked with this IDMSCINT module should share the same transaction as other sessions started by the same CICS task.

ON

Specifies that database sessions should share transactions.

OFF

Specifies that database sessions should not share transactions. This is the default.

DEFAULT

Specifies that whether or not database sessions share transactions is determined by the TXNSHR parameter of the interface's CICSOPT macro.

F.4 CICS OPTIXIT

OPTIXIT programs enable users to dynamically route database sessions to different back-end central versions. In order to support two-phase commit processing with CICS, OPTIXIT users must enhance their exit code to be able to route resynchronization requests to the correct back-end central version. In order to facilitate this, resynchronization requests are identified by an SSC program name of INTCRSYN and the name of the node to which the request must be routed is contained in the OPTI control block passed as a second parameter to the exit.

The following is an example of the type of coding necessary to recognize and route resynchronization requests successfully.

F.4.1 OPTIXIT Example

```
TITLE 'OPTIXIT  -  example of CICS OPTI exit needed for CICS RESYNC'
OPTIXIT  CSECT
         USING OPTIXIT,R15                  ---> Base
          B     START                       Go processs OPTI exit call
         DROP  R15
          #MOPT CSECT=OPTIXIT,ENV=USER
START     DS    0H
          STM   R14,R12,12(R13)             Save callers registers
          LR    R12,R15                     Swap base to R12
         USING OPTIXIT,R12                  ---> Base
         USING OPTXPLST,R1                  ---> Parameter list
          L     R2,OPTXSSCA                 Get address of Subschema Control
         USING SSC,R2                       ---> SSC
          L     R3,OPTXOPTA                 Get address of OPTI structure
         USING OPI,R3                       ---> OPTI structure
          CLC   SSCPNAME,=C'INTCRSYN'       Pseudo SSC for CICS RESYNC?
          BE    CICSRSYN                    Yes, special process for CICS RESYNC
*************************************************************************
*
* perform normal OPTIXIT logic for real SSC
*
*************************************************************************
          B     RETURN                      Exit
CICSRSYN DS    0H
          LA    R5,OPINODE                  Point at name of backend CV
          LA    R4,SYSLIST                  Get table of known backend CVs
         USING SYSTABLE,R4                  ---> SYSTABLE
LOOP      DS    0H
          CLI   SYSNAME,C'*'                Is this end of CV table ?
          BE    RETURN                      Yes, just exit
          CLC   SYSNAME,0(R5)               Is this CV one of my CVs ?
          BE    MATCH                       Yes, we have a match
          LA    R4,SYSTSIZE(R4)             Bump to next CV in the table
          B     LOOP                        Keep looking for my CVs
MATCH     DS    0H
          MVC   OPICVNUM,SYSCV#             Update OPTI with CV number
          MVC   OPISVCNO,SYSSVC#            Update OPTI with SVC number
          B     RETURN                      Exit
RETURN    DS    0H
          LM    R14,R12,12(R13)             Restore callers registers
```

```
         BR     R14                        Return to caller
        DROP  R2,R3,R4,R12                Drop SSC, OPI, SYSTABLE, base
         EJECT
SYSLIST  EQU   *                           Backend CV table
         DC    C'SYSTEM71',AL1(71),AL1(173)  CV 71 uses SVC number 173
         DC    C'SYSTEM72',AL1(72),AL1(176)  CV 72 uses SVC number 176
         DC    C'SYSTEM73',AL1(73),AL1(176)  CV 73 uses SVC number 176
         DC    C'SYSTEM74',AL1(74),AL1(173)  CV 74 uses SVC number 173
         DC    C'*'                        End of backend CV table
         SPACE 2
         LTORG ,                           Literal pool
         SPACE 2
OPTXPLST DSECT                             OPTI exit PLIST
OPTXSSCA DS    A(0)                        A(SSC)
OPTXOPTA DS    A(0)                        A(OPTI)
         SPACE 2
SYSTABLE DSECT                             Backend CV table dsect
SYSNAME  DS    CL8                         Backend CV node name
SYSCV#   DS    XL1                         Backend CV number
SYSSVC#  DS    XL1                         Backend CV SVC number
SYSTSIZE EQU   *-SYSTABLE                  Size of one SYSTABLE entry
         EJECT
         COPY  #OPTIDS                     OPTI dsect
         COPY  #SSCDS                      Subschema control dsect
         END
```

Appendix G. TCP/IP API Commands, Error Codes, Socket Structures, and String Conversion

G.1 Overview

This appendix provides information about TCP/IP, including:

- Detailed description of each supported socket function
- ERRNO codes associated with sockets
- Socket structure description
- String conversion functions

G.2 Function Descriptions

This section describes the socket functions that are supported by Advantage CA-IDMS. The following information is provided for each function:

- An assembler #SOCKET macro invocation showing all of the parameters that can be specified.
- A list of parameters that can be passed when invoking the function in COBOL, PL/I, and Advantage CA-ADS. The first of these parameters is the function name as defined in the SOCKET-CALL-INTERFACE record.
- A description of the function-dependent parameters.
- Additional notes if applicable to a specific function.

G.2.1 ACCEPT

ACCEPT accepts the first connection request on the queue of pending connection requests. If the queue is empty, the call waits until the first connection request arrives or fails with an EWOULDBLOCK condition if the socket had been marked as non-blocking. If successful, a new socket descriptor is returned.

Assembler

```
label    #SOCKET ACCEPT,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               SOCKADDR=sockaddr,
               SOCKADDL=sockaddr-length,
               NEWSOCK=new-socket-descriptor,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-ACCEPT,
return-code,
errno,
reason-code,
socket-descriptor,
sockaddr,
sockaddr-length,
new-socket-descriptor
```

G.2.1.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor that was used on the BIND and LISTEN functions
sockaddr	The name of an area in which to return the sockaddr structure of the connecting client. The format of that structure depends on the domain of the corresponding socket. This parameter can be assigned to NULL if the caller is not interested in the connector's address.
sockaddr-length	The name of a fullword field containing the length of *sockaddr*. If SOCKADDR is assigned to NULL, *sockaddr-length* must be 0. On return, *sockaddr-length* contains the size required to represent the connecting socket. If the value is 0, the contents of *sockaddr* are unchanged. If the *sockaddr* is too small to contain the full sockaddr structure, it is truncated. The maximum value for this parameter is 4096.
new-socket-descriptor	The name of the fullword field where the socket descriptor of the new connection is returned.

G.2.1.2 Notes

When the timeout value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket timeouts, refer to 9.5.3, “Associating Timeouts to Sockets” on page 9-25.

G.2.2 BIND

BIND assigns a local name to an unnamed socket.

Assembler

```
label    #SOCKET BIND,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               SOCKADDR=sockaddr,
               SOCKADDL=sockaddr-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-BIND,
return-code,
errno,
reason-code,
socket-descriptor,
sockaddr,
sockaddr-length
```

G.2.2.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor to bind.
sockaddr	The name of an area that contains the sockaddr structure to be bound to the socket. The format of the sockaddr structure depends on the domain of the corresponding socket. **VSE/ESA systems:** Only the domain AF_INET is supported.
sockaddr-length	The name of a fullword field containing the length of sockaddr. *sockaddr-length* can be specified as an absolute expression. The maximum value for this parameter is domain dependent. If the domain is: ■ AF_INET — it is the length of the SOCKET-SOCKADDR-IN record (SIN#LEN for assembler) ■ AF_INET6 — it is the length of the SOCKET-SOCKADDR-IN6 record (SIN6#LEN for assembler)

G.2.3 CLOSE

CLOSE deletes the socket descriptor from the internal descriptor table maintained for the application program and terminates the existence of the communications endpoint. If the socket was connected, the connection is terminated in an orderly fashion.

Assembler

```
label    #SOCKET CLOSE,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-CLOSE,
return-code,
errno,
reason-code,
socket-descriptor
```

G.2.3.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor to close.

G.2.4 CONNECT

CONNECT initiates a connection on a socket.

Assembler

```
label    #SOCKET CONNECT,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               SOCKADDR=sockaddr,
               SOCKADDL=sockaddr-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-CONNECT,
return-code,
errno,
reason-code,
socket-descriptor,
sockaddr,
sockaddr-length
```

G.2.4.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword containing the socket descriptor to which to connect.

Parameter	Description
sockaddr	The name of an area that contains the sockaddr structure to which to connect. The format of the sockaddr structure depends on the domain of the corresponding socket. **VSE/ESA systems:** Only the domain AF_INET is supported. Specify family *AF@INET* when building the sockaddr structure.
sockaddr-length	The name of a fullword field containing the length of *sockaddr*. *sockaddr-length* can be specified as an absolute expression. The maximum value for this parameter is domain dependent. If the domain is: ■ AF_INET — it is the length of the SOCKET-SOCKADDR-IN record (SIN#LEN for assembler) ■ AF_INET6 — it is the length of the SOCKET-SOCKADDR-IN6 record (SIN6#LEN for assembler)

G.2.4.2 Notes

- When the timeout value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket timeouts, refer to 9.5.3, “Associating Timeouts to Sockets” on page 9-25.
- After a CONNECT error, including a timeout condition, the corresponding socket cannot be used. For the application to continue processing, it must close the current socket and create a new socket.

G.2.5 FCNTL

FCNTL provides control over a socket descriptor. Depending on the command, it retrieves or sets control information.

Assembler

```
label     #SOCKET FCNTL,
                RETCODE=return-code,
                ERRNO=errno,
                RSNCODE=reason-code,
                SOCK=socket-descriptor,
                COMMAND=command,
                ARGUMENT=argument,
                RETVAL=returned-value,
                PLIST=parameter-list-area,
                RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-FCNTL,
return-code,
errno,
reason-code,
socket-descriptor,
command,
argument,
returned-value
```

G.2.5.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor to process.
command	The name of a fullword field containing the command to perform on the socket. *command* can be specified as an absolute expression.
argument	The name of a fullword field containing the argument that applies to some commands. *argument* can be specified as an absolute expression. While *argument* is optional, it must be specified for setting functions.
returned-value	The name of a fullword field that contains the returned information from any retrieval commands. While *returned-value* is optional , it must be specified for retrieval function.

G.2.5.2 Notes

- **VSE/ESA systems:** The F@GETFL and F@SETFL commands are not supported.
- The following table lists the commands and arguments that can be specified. The EQUate symbol is generated by #SOCKET macro and the field names are associated with the SOCKET-MISC-DEFINITIONS record.

EQUate Symbol	Field Name	Description
F@GETFL	SOCKET-FCNTL-GET	Get file status command
F@SETFL	SOCKET-FCNTL-SET	Set file status command
F@GETIMO	SOCKET-FCNTL-GETIMO	Get timeout value associated with a socket
F@SETIMO[1]	SOCKET-FCNTL-SETIMO	Associate a timeout value with a socket

EQUate Symbol	Field Name	Description
NONBLOCK	SOCKET-FCNTL-NONBLOCK	Set socket in non-blocking mode

Note: [1] — Acceptable argument values for the F@SETIMO command:

- 1 through 32767 — the timeout value in seconds
- 0 — no timeout processing is wanted, but a task abend
- -1 — indefinite wait (equivalent for FOREVER)

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

G.2.6 FD_CLR

FD_CLR clears a socket descriptor's bit in a bit list.

Assembler

```
label     #SOCKET FD_CLR,
                RETCODE=return-code,
                ERRNO=errno,
                RSNCODE=reason-code,
                SOCK=socket-descriptor,
                BITLIST=bit-list,
                BITLISTL=bit-list-length,
                BITORDER=bit-order,
                PLIST=parameter-list-area
```

G.2.6.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor whose bit must be cleared (set to zero) in the bit list.
bit-list	The name of the area containing the bit list.
bit-list-length	The name of a fullword field containing the length of the bit-list in bytes. *bit-list-length* can be specified as an absolute expression. *bit-list-length* must be a multiple of 4.

Parameter	Description
bit-order	The name of the fullword containing the order in which the bits are addressed in the bit list. This order should be the same as the value specified on the *option* parameter of the SELECT or SELECTX function. *bit-order* can be specified as an absolute expression. The accepted values are: ■ SEL@BBKW (default) ■ SEL@BFWD

G.2.6.2 Notes

- This function is only available to the Assembler interface.
- For performance reasons, FD_CLR does not call the RHDCSOCK processor to execute the function. Instead, the corresponding code is expanded in your program. This code is substantial, so it is best to code the function call in a subroutine.

G.2.7 FD_ISSET

FD_ISSET tests a socket descriptor's bit in a bit list to see if it is ON or OFF

Assembler

```
label    #SOCKET FD_ISSET,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               BITLIST=bit-list,
               BITLISTL=bit-list-length,
               BITORDER=bit-order,
               RETVAL=returned-bit-status,
               PLIST=parameter-list-area
```

G.2.7.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor whose bit needs testing in the bit list.
bit-list	The name of the area containing the bit list.

Parameter	Description
bit-list-length	The name of a fullword field containing the length of the bit-list in bytes. *bit-list-length* can be specified as an absolute expression. *bit-list-length* must be a multiple of 4.
bit-order	The name of a fullword containing the order in which the bits are addressed in the bit list. This order should be the same as the value specified on the *option* parameter of the SELECT or SELECTX function. *bit-order* can be specified as an absolute expression. The accepted values are: ■ SEL@BBKW (default) ■ SEL@BFWD
returned-bit-status	The name of a fullword field that will contain the status of the tested bit: ■ 0 — OFF ■ 1 — ON

G.2.7.2 Notes

- This function is only available to the Assembler interface.
- For performance reasons, FD_ISSET does not call the RHDCSOCK processor to execute the function. Instead, the corresponding code is expanded in your program. This code is substantial, so it is best to code the function call in a subroutine.

G.2.8 FD_SET

FD_SET sets a socket descriptor's bit in a bit list ON.

Assembler

```
label     #SOCKET FD_SET,
                RETCODE=return-code,
                ERRNO=errno,
                RSNCODE=reason-code,
                SOCK=socket-descriptor,
                BITLIST=bit-list,
                BITLISTL=bit-list-length,
                BITORDER=bit-order,
                PLIST=parameter-list-area
```

G.2.8.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor whose bit must be set ON in the bit list.
bit-list	The name of the area containing the bit list.
bit-list-length	The name of a fullword field containing the length of the bit-list in bytes. *bit-list-length* can be specified as an absolute expression. *bit-list-length* must be a multiple of 4.
bit-order	The name of the fullword containing the order in which the bits are addressed in the bit list. This order should be the same as the value specified on the *option* parameter of the SELECT or SELECTX function. *bit-order* can be specified as an absolute expression. The accepted values are: ■ SEL@BBKW (default) ■ SEL@BFWD

G.2.8.2 Notes

- This function is only available to the Assembler interface.
- For performance reasons, FD_SET does not call the RHDCSOCK processor to execute the function. Instead, the corresponding code is expanded in your program. This code is substantial, so it is best to code the function call in a subroutine.

G.2.9 FD_ZERO

FD_ZERO clears all bits in a bit list.

Assembler

```
label    #SOCKET FD_ZERO,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               BITLIST=bit-list,
               BITLISTL=bit-list-length,
               PLIST=parameter-list-area
```

G.2.9.1 Parameters

Parameter	Description
bit-list	The name of the area containing the bit list.
bit-list-length	The name of a fullword field containing the length of the bit-list in bytes. *bit-list-length* can be specified as an absolute expression. *bit-list-length* must be a multiple of 4.

G.2.9.2 Notes

- This function is only available to the Assembler interface.
- For performance reasons, FD_ZERO does not call the RHDCSOCK processor to execute the function. Instead, the corresponding code is expanded in your program. This code is substantial, so it is best to code the function call in a subroutine.

G.2.10 FREEADDRINFO

FREEADDRINFO frees the ADDRINFO structure that has been allocated by the system during the processing of a previous call to the GETADDRINFO #SOCKET function.

Assembler

```
label    #SOCKET FREEADDRINFO,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               AINFOIN=pointer-to-addrinfo-structure,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-FREEADDRINFO,
return-code,
errno,
reason-code,
pointer-to-addrinfo-structure
```

G.2.10.1 Parameters

Parameter	Description
pointer-to-addrinfo-structure	The name of a fullword field containing the address of the ADDRINFO structure to release.

G.2.10.2 Notes

- The FREEADDRINFO function is supported as of z/OS V1R4.
- The FREEADDRINFO function *is not* supported in these operating environments:
  - VSE/ESA
  - z/VM

G.2.11 GETADDRINFO

GETADDRINFO converts a host name and/or a service name into a set of socket addresses and other associated information. This information can be used to open a socket and connect to the specified service.

Assembler

```
label    #SOCKET GETADDRINFO,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               HOSTNAME=hostname,
               HOSTNAML=hostname-length,
               SERVNAME=service-name,
               SERVNAML=service-name-length,
               AINFOIN=pointer-to-input-addrinfo-structure,
               AINFOOUT=pointer-to-output-addrinfo-structure,
               CANONAML=canonical-name-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETADDRINFO,
return-code,
errno,
reason-code,
hostname,
hostname-length,
service-name,
service-name-length,
pointer-to-input-addrinfo-structure,
pointer-to-output-addrinfo-structure,
canonical-name-length
```

G.2.11.1 Parameters

Parameter	Description
hostname	The name of an area containing the name of the host to resolve.
hostname-length	The name of a fullword field containing the length of *hostname*. *hostname-length* can be specified as an absolute expression. *hostname* and *hostname-length* are optional. If they are not specified, *service-name* and *service-name-length* must be specified. The maximum value for this parameter is 256.
service-name	The name of an area containing the name of the service.
service-name-length	The name of a fullword field containing the length of *service-name*. *service-name-length* can be specified as an absolute expression. *service-name* and *service-name-length* are optional. If they are not specified, *hostname* and *hostname-length* must be specified. The maximum value for this parameter is 32.
pointer-to-input-addrinfo-structure	The name of a fullword field containing the address of an input ADDRINFO structure. The following fields in the ADDRINFO structure can be set: flags, family, socket type, and protocol. If this pointer is assigned to NULL, it is equivalent to an ADDRINFO structure where all fields are set to 0.
pointer-to-output-addrinfo-structure	The name of a fullword field that contains the address of the output ADDRINFO structure returned by the system. This structure has to be explicitly released by the user using the FREEADDRINFO #SOCKET call.
canonical-name-length	The name of a fullword field in which the system returns the length of the canonical name. The system returns the canonical name in the first output ADDRINFO structure if *hostname* is specified and the AI_CANONNAMEOK flag is set in the input ADDRINFO structure. If the canonical name length is not needed, *canonical-name-length* can be omitted.

G.2.11.2 Notes

- For more information on the ADDRINFO structure, refer to A.4, “DCMT DISPLAY DBTRACE” on page A-8.
- The GETADDRINFO function is supported as of z/OS V1R4.
- The GETADDRINFO function *is not* supported in these operating environments:
  - VSE/ESA

– z/VM

- The following table lists the flags that can be set or returned in the ADDRINFO structure. The EQUate symbol is generated by the #SOCKET TCPIPDEF macro call and the field names are associated with the SOCKET-MISC-DEFINITIONS record.

EQUate Symbol	Field Name	TCP Protocol Value
AI@PASSV	SOCKET-AIFLAGS-PASSIVE	AI_PASSIVE
AI@CANOK	SOCKET-AIFLAGS-CANONNAMEOK	AI_CANONNAMEOK
AI@NHOST	SOCKET-AIFLAGS-NUMERICHOST	AI_NUMERICHOST
AI@NSERV	SOCKET-AIFLAGS-NUMERICSERV	AI_NUMERICSERV
AI@V4MAP	SOCKET-AIFLAGS-V4MAPPED	AI_V4MAPPED
AI@ALL	SOCKET-AIFLAGS-ALL	AI_ALL
AI@ADDRC	SOCKET-AIFLAGS-ADDRCONFIG	AI_ADDRCONFIG

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

G.2.12 GETHOSTBYADDR

GETHOSTBYADDR takes an IP address and domain and tries to resolve it through a name server. If successful, it returns the information in a HOSTENT structure.

Assembler

```
label    #SOCKET GETHOSTBYADDR,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               IPADDR=ip-address,
               IPADDRL=ip-address-length,
               DOMAIN=domain,
               HOSTENTP=hostentp,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETHOSTBYADDR,
return-code,
errno,
reason-code,
ip-address,
ip-address-length,
domain,
hostentp
```

G.2.12.1 Parameters

Parameter	Description
ip-address	The name of a fullword field containing the binary format IP address to resolve.
ip-address-length	The name of a fullword field containing the length of *ip-address*. *ip-address-length* can be specified as an absolute expression. The maximum value for this parameter is defined by IPADDR4L in assembler and SOCKET-IPADDR4L in other languages.
domain	The name of a fullword field containing the domain. *domain* can be specified as an absolute expression. Currently, only AF_INET is supported.
hostentp	The name of a fullword field in which the system returns the address of a HOSTENT structure containing the information about the host.

G.2.12.2 Notes

- The HOSTENT structure area is allocated by the system at the Advantage CA-IDMS task level, and freed at task termination. It is reused by subsequent calls to a DNS function: GETHOSTBYADDR or GETHOSTBYNAME.
- For more information on the HOSTENT structure, refer to A.4, “DCMT DISPLAY DBTRACE” on page A-8.

z/VM systems: The DNS socket functions are supported by Advantage CA-IDMS's internal DNS resolver. Refer to the TCP/IP Considerations section of the *Advantage CA-IDMS System Operations* manual for information about configuring the DNS resolver.

VSE/ESA systems: The DNS socket functions can be supported by Advantage CA-IDMS's internal DNS resolver. If the socket functions are supported by:

- Advantage CA-IDMS — Refer to the TCP/IP Considerations section of *Advantage CA-IDMS System Operations* for information about configuring the DNS resolver.
- Barnard Software Inc. or Connectivity Systems Inc. — Refer to the appropriate TCP/IP stack documentation for configuring DNS support.

G.2.13 GETHOSTBYNAME

GETHOSTBYNAME takes a host name and tries to resolve it through a name server. If successful, it returns the information in a HOSTENT structure.

Assembler

```
label    #SOCKET GETHOSTBYNAME,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               HOSTNAME=hostname,
               HOSTNAML=hostname-length,
               HOSTENTP=hostentp,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETHOSTBYNAME,
return-code,
errno,
reason-code,
hostname,
hostname-length,
hostentp
```

G.2.13.1 Parameters

Parameter	Description
hostname	The name of an area containing the name of the host to resolve.
hostname-length	The name of a fullword field containing the length of *hostname*. *hostname-length* can be specified as an absolute expression. The maximum value for this parameter is 256.
hostentp	The name of a fullword field where the system returns the address of a HOSTENT structure containing the information about the host.

G.2.13.2 Notes

- The HOSTENT structure area is allocated by the system at the Advantage CA-IDMS task level, and freed at task termination. It is reused by subsequent calls to a DNS function: GETHOSTBYADDR or GETHOSTBYNAME.
- For more information on the HOSTENT structure, refer to the A.4, “DCMT DISPLAY DBTRACE” on page A-8.

z/VM systems: The DNS socket functions are supported by Advantage CA-IDMS's internal DNS resolver. Refer to the TCP/IP Considerations section of the

Advantage CA-IDMS System Operations manual for information about configuring the DNS resolver.

VSE/ESA systems: The DNS socket functions can be supported by Advantage CA-IDMS's internal DNS resolver. If the socket functions are supported by:

- Advantage CA-IDMS — Refer to the TCP/IP Considerations section of *Advantage CA-IDMS System Operations* for information about configuring the DNS resolver.
- Barnard Software Inc. or Connectivity Systems Inc. — Refer to the appropriate TCP/IP stack documentation for configuring DNS support.

G.2.14 GETHOSTID

GETHOSTID retrieves the IP address of the local host corresponding to the current TCP/IP stack.

Assembler

```
label    #SOCKET GETHOSTID,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               IPADDR=ip-address,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETHOSTID,
return-code,
errno,
reason-code,
ip-address
```

G.2.14.1 Parameters

Parameter	Description
ip-address	The name of a fullword field in which the service returns the IP address in binary format.

G.2.14.2 Notes

This service only supports IPv4.

G.2.15 GETHOSTNAME

GETHOSTNAME retrieves the name of the local host corresponding to the current TCP/IP stack.

Assembler

```
label    #SOCKET GETHOSTNAME,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               HOSTNAME=hostname,
               HOSTNAML=hostname-length,
               RETLEN=returned-hostname-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETHOSTNAME,
return-code,
errno,
reason-code,
hostname,
hostname-length,
returned-hostname-length
```

G.2.15.1 Parameters

Parameter	Description
hostname	The name of an area in which the service returns the host name.
hostname-length	The name of a fullword field containing the length of *hostname*. *hostname-length* can be specified as an absolute expression. The maximum value for this parameter is 256.
returned-hostname-length	The name of a fullword field in which the actual length of the host name is returned.

G.2.16 GETNAMEINFO

GETNAMEINFO resolves a socket address into a hostname and a service name.

Assembler

```
label    #SOCKET GETNAMEINFO,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCKADDR=sockaddr,
               SOCKADDL=sockaddr-length,
               SERVNAME=service-name,
               SERVNAML=service-name-length,
               RETSNAML=returned-service-name-length,
               HOSTNAME=hostname,
               HOSTNAML=hostname-length,
               RETHNAML=returned-hostname-length,
               FLAGS=flags,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETNAMEINFO,
return-code,
errno,
reason-code,
sockaddr,
sockaddr-length,
service-name,
service-name-length,
returned-service-name-length,
hostname,
hostname-length,
returned-hostname-length,
flags
```

G.2.16.1 Parameters

Parameter	Description
sockaddr	The name of the sockaddr structure containing the information that must be resolved: the domain (or socket family), the port number and the IP address.
sockaddr-length	The name of a fullword field containing the length of *sockaddr*. *sockaddr-length* can be specified as an absolute expression. The maximum value for this parameter is domain dependent. If the domain is: ■ AF_INET — it is the length of the SOCKET-SOCKADDR-IN record (SIN#LEN for assembler) ■ AF_INET6 — it is the length of the SOCKET-SOCKADDR-IN6 record (SIN6#LEN for assembler)
service-name	The name of an area where the system returns the service name corresponding to the port number specified in the sockaddr structure.

Parameter	Description
service-name-length	The name of a fullword field containing the length of *service-name*. *service-name-length* can be specified as an absolute expression. The maximum value for this parameter is 4096.
returned-service-name-length	The name of a fullword field into which the actual length of the service name is returned. *service-name*, *service-name-length* and *returned-service-name-length* are optional; specify all 3 parameters, or none of them. If none of these parameters are specified, *hostname, hostname-length,* and *returned-hostname-length* must be specified.
hostname	The name of an area where the system returns the *hostname* corresponding to the IP address specified in the sockaddr structure.
hostname-length	The name of a fullword field containing the length of *hostname*. *hostname-length* can be specified as an absolute expression. The maximum value for this parameter is 4096.
returned-hostname-length	The name of a fullword field into which the length of the host name is returned. *hostname*, *hostname-length* and *returned-hostname-length* are optional; specify all 3 parameters, or none of them. If none of these parameters are specified, *service-name, service-name-length,* and *returned-service-name-length* must be specified.
flags	The name of a fullword field containing flags to control the resolution of the socket address.

G.2.16.2 Notes

- The GETNAMEINFO function is supported as of z/OS V1R4.
- The GETNAMEINFO function *is not* supported in these operating environments:
  - VSE/ESA
  - z/VM
- The following table lists the flags that can be passed. The EQUate symbol is generated by the #SOCKET TCPIPDEF macro call and the field names are associated with the SOCKET-MISC-DEFINITIONS.

EQUate Symbol	Field Name	Description
NI@NFDQN	SOCKET-NIFLAGS-NOFQDN	Returns the node name portion only

EQUate Symbol	Field Name	Description
NI@NREQD	SOCKET-NIFLAGS-NAMEREQD	Returns an error if the host is not located
NI@NHOST	SOCKET-NIFLAGS-NUMERICHOST	Returns the numeric form of the host
NI@NSERV	SOCKET-NIFLAGS-NUMERICSERV	Returns the numeric form of the server
NI@DGRAM	SOCKET-NIFLAGS-DGRAM	The service is a datagram service

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

G.2.17 GETPEERNAME

GETPEERNAME retrieves the name of the peer connected to a socket.

Assembler

```
label    #SOCKET GETPEERNAME,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               SOCKADDR=sockaddr,
               SOCKADDL=sockaddr-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETPEERNAME,
return-code,
errno,
reason-code,
socket-descriptor,
sockaddr,
sockaddr-length
```

G.2.17.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor from which to retrieve the peer name.

Parameter	Description
sockaddr	The name of an area in which to return the sockaddr structure of the peer. The format of this structure depends on the domain of the corresponding socket. This parameter can be assigned to NULL if the caller is not interested in the peer's address.
sockaddr-length	The name of a fullword field containing the length of *sockaddr*. If SOCKADDR is assigned to NULL, *sockaddr-length* must be 0. On return, *sockaddr-length* contains the size required to represent the peer. If the size of *sockaddr* is too small to contain the full sockaddr structure, it is truncated. The maximum value for this parameter is 4096

G.2.18 GETSOCKNAME

GETSOCKNAME retrieves the current name of a socket into a sockaddr structure.

Assembler

```
label    #SOCKET GETSOCKNAME,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               SOCKADDR=sockaddr,
               SOCKADDL=sockaddr-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETSOCKNAME,
return-code,
errno,
reason-code,
socket-descriptor,
sockaddr,
sockaddr-length
```

G.2.18.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor from which to retrieve the name.

Parameter	Description
sockaddr	The name of an area in which to return the sockaddr structure of the socket. The format of this structure depends on the domain of the corresponding socket. This parameter can be assigned to NULL if the caller is not interested in the socket's address.
sockaddr-length	The name of a fullword field containing the length of *sockaddr*. If SOCKADDR is assigned to NULL, *sockaddr-length* must be 0. On return, *sockaddr-length* contains the size required to represent the socket. If the size of *sockaddr* is too small to contain the full sockaddr structure, it is truncated. . The maximum value for this parameter is 4096

G.2.19 GETSOCKOPT

GETSOCKOPT retrieves the options currently associated with a socket.

Assembler

```
label     #SOCKET GETSOCKOPT,
                RETCODE=return-code,
                ERRNO=errno,
                RSNCODE=reason-code,
                SOCK=socket-descriptor,
                LEVEL=level,
                OPTNAME=option-name,
                OPTVAL=option-value,
                OPTLEN=option-value-length,
                PLIST=parameter-list-area,
                RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETSOCKOPT,
return-code,
errno,
reason-code,
socket-descriptor,
level,
option-name,
option-value,
option-value-length
```

G.2.19.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor for which the service is to be performed.

Parameter	Description
level	The name of a fullword field containing the level for the *option*. *level* can be specified as an absolute expression.
option-name	The name of a fullword field indicating the *option* to retrieve. *option-name* can be specified as an absolute expression.
option-value	The name of an area that will contain the requested data.
option-value-length	The name of a fullword field that contains the length of *option-value*. On return, *option-value-length* contains the size of the data returned in *option-value*. The maximum value for this parameter is 4096.

G.2.19.2 Notes

- **VSE/ESA systems:** The GETSOCKOPT function is not supported.
- The following table lists the options that can be specified. The EQUate symbol is generated by the #SOCKET TCPIPDEF macro call and the field names are associated with the SOCKET-MISC-DEFINITIONS.

EQUate Symbol	Field Names	Description
S@SOCKET	SOCKET-SOCKOPT-SOLSOCKET	Level number for socket options
SO@REUSE	SOCKET-SOCKOPT-REUSEADDR	Allows local address reuse
SO@KEEPA	SOCKET-SOCKOPT-KEEPALIVE	Activate the keep-alive mechanism.
SO@OOBIN	SOCKET-SOCKOPT-OOBINLINE	Accept out-of-band data.
SO@SNBUF	SOCKET-SOCKOPT-SNDBUF	Reports send buffer size information
SO@RCBUF	SOCKET-SOCKOPT-RCVBUF	Reports receive buffer size information

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

G.2.20 GETSTACKS

GETSTACKS retrieves the list of all the TCP/IP stacks currently defined in the system.

Assembler

```
label     #SOCKET GETSTACKS,
                  RETCODE=return-code,
                  ERRNO=errno,
                  RSNCODE=reason-code,
                  BUFFER=buffer,
                  BUFFERL=buffer-length,
                  FORMAT=output-format
                  RETLEN=output-length,
                  RETNSTKS=stacks-count,
                  PLIST=parameter-list-area,
                  RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETSTACKS,
return-code,
errno,
reason-code,
buffer,
buffer-length,
output-format,
output-length,
stacks-count
```

G.2.20.1 Parameters

Parameter	Description
buffer	The name of a buffer that receives the list of all the stacks. This parameter is optional.
buffer-length	The name of a fullword field containing the length of *buffer*. *buffer-length* can be specified as an absolute expression. This parameter is optional. If the size of *buffer* is too small to contain the full output, it is truncated. The maximum value for this parameter is 4096.

Parameter	Description
output-format	The name of a fullword field indicating the format desired for the output. *output-format* can be specified as an absolute expression. If the *output-format* value is: ■ 1 — All the names of the different stacks are listed in a sequence of 8-byte character string. ■ 2 — All the names of the different stacks are listed in a sequence of the following structure: a 1-byte field containing the length of the name followed by the name itself. This is an optional parameter. If it is not specified, output-format 1 is assumed.
output-length	The name of a fullword field containing the actual length required to hold all the output in the requested format..
stacks-count	The name of a fullword field containing the number of TCP/IP stacks currently defined (but not necessarily active) in the system.

G.2.20.2 Notes

- The *buffer* and *buffer-length* parameters are optional. If these parameters are not specified, only the *output-length* and *stacks-count* values are returned.
- Refer to 9.5.2, “Using Multiple TCP/IP Stacks” on page 9-24 for more information.

G.2.21 HTONL

HTONL converts a fullword integer from host byte order to network byte order. Within Advantage CA-IDMS, host and network byte order are the same. Therefore, the HTONL function does not apply to the mainframe environment; it is implemented for the application programmer's convenience.

Assembler

```
label    #SOCKET HTONL,
               FIELDIN=input-field,
               FIELDOUT=output-field,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-HTONL,
input-field,
output-field
```

G.2.21.1 Parameters

Parameter	Description
input-field	The name of a fullword field containing the integer to convert.
output-field	The name of a fullword field that receives the converted integer.

G.2.22 HTONS

HTONS converts a halfword integer from host byte order to network byte order. Within Advantage CA-IDMS, host and network byte order are the same. Therefore, the HTONS function does not apply to the mainframe environment; it is implemented for the application programmer's convenience.

Assembler

```
label     #SOCKET HTONS,
                FIELDIN=input-field,
                FIELDOUT=output-field,
                PLIST=parameter-list-area,
                RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-HTONS,
input-field,
output-field
```

G.2.22.1 Parameters

Parameter	Description
input-field	The name of a halfword field containing the integer to convert.
output-field	The name of a halfword field that receives the converted integer.

G.2.23 INET_ADDR

INET_ADDR translates an IP address in standard dotted string format into its binary format.

Assembler

```
label    #SOCKET INET_ADDR,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               IPADDRS=ip-address_string,
               IPADDRSL=ip-address-string-length,
               IPADDR=ip-address,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-INETADDR,
return-code,
errno,
reason-code,
ip-address-string,
ip-address-string-length,
ip-address
```

G.2.23.1 Parameters

Parameter	Description
ip-address-string	The name of an area containing the IP address in standard dotted string format.
ip-address-string-length	The name of a fullword field containing the length of *ip-address-string*, which can be specified as an absolute expression. The maximum value for this parameter is defined by IPADDS4L in assembler and SOCKET-IPADDS4L in other languages.
ip-address	The name of a fullword field that will contain the IP address in binary format.

G.2.24 INET_NTOA

INET_NTOA translates an IP address in binary format into standard dotted string format. The IP address is in IPv4 format.

Note: INET_NTOA does not support IPv6 format. For new applications use INET_NTOP, which supports IPv6 and IPv4 formats.

Assembler

```
label    #SOCKET INET_NTOA,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               IPADDR=ip-address,
               IPADDRS=ip-address-string,
               IPADDRSL=ip-address-string-length,
               RETIPASL=returned-ip-address-string-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-INETNTOA,
return-code,
errno,
reason-code,
ip-address,
ip-address-string,
ip-address-string-length,
returned-ip-address-string-length
```

G.2.24.1 Parameters

Parameter	Description
ip-address	The name of a fullword field containing the IP address in binary format.
ip-address-string	The name of an area in which to return the IP address in standard dotted string format.
ip-address-string-length	The name of a fullword field containing the length of *ip-address-string*. *ip-address-string-length* can be specified as an absolute expression. The maximum value for this parameter is 4096.
returned-ip-address-string-length	The name of a fullword field in which the actual length of the IP address string is returned.

G.2.25 INET_NTOP

INET_NTOP translates an IP address in binary format into standard string format.

Assembler

```
label    #SOCKET INET_NTOP,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               DOMAIN=domain,
               IPADDR=ip-address,
               IPADDRS=ip-address-string,
               IPADDRSL=ip-address-string-length,
               RETIPASL=returned-ip-address-string-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-INETNTOP,
return-code,
errno,
reason-code,
domain,
ip-address,
ip-address-string,
ip-address-string-length,
returned-ip-address-string-length
```

G.2.25.1 Parameters

Parameter	Description
domain	The name of a fullword field containing the domain. *domain* can be specified as an absolute expression. Possible values are AF@INET and AF@INET6.
ip-address	The name of an area containing the IP address in binary format: a fullword for an Ipv4 address, or a 16-byte area for an Ipv6 address.
ip-address-string	The name of an area in which to return the IP address in standard string format.
ip-address-string-length	The name of a fullword field containing the length of *ip-address-string*. *ip-address-string-length* can be specified as an absolute expression. The maximum value for this parameter is 4096.
returned-ip-address-string-length	The name of a fullword field in which the actual length of the IP address string is returned.

G.2.26 INET_PTON

INET_PTON translates an IP address in standard string format into its binary format.

Assembler

```
label     #SOCKET INET_PTON,
                RETCODE=return-code,
                ERRNO=errno,
                RSNCODE=reason-code,
                DOMAIN=domain,
                IPADDRS=ip-address_area,
                IPADDRSL=ip-address-string-length,
                IPADDR=ip-address,
                PLIST=parameter-list-area,
                RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-INETPTON,
return-code,
errno,
reason-code,
domain,
ip-address-string,
returned-ip-address-string-length,
ip-address
```

G.2.26.1 Parameters

Parameter	Description
domain	The name of a fullword field containing the domain. *domain* can be specified as an absolute expression. Possible values are AF@INET and AF@INET6.
ip-address-string	The name of an area containing the IP address in standard string format.
ip-address-string-length	The name of a fullword field containing the length of *ip-address-string*. *ip-address-string-length* can be specified as an absolute expression. The maximum value for this parameter is determined by the type of address: ■ IPv4 address — IPADDS4L in assembler and SOCKET-IPADDS4L in other languages ■ IPv6 address — IPADDS6L in assembler and SOCKET-IPADDS6L in other languages
ip-address	The name of an area in which to return the IP address in binary format: a fullword for an IPv4 address, or a 16-bytes area for an IPv6 address.

G.2.27 LISTEN

LISTEN indicates that an application is ready to accept client connection requests and defines the maximum length of the connection request queue.

Assembler

```
label    #SOCKET LISTEN,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               BACKLOG=backlog,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-LISTEN,
return-code,
errno,
reason-code,
socket-descriptor,
backlog
```

G.2.27.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor on which to listen.
backlog	The name of a fullword field containing the backlog value. *backlog* can be specified as an absolute expression. It defines the maximum number of pending connections that may be queued. The value cannot exceed the maximum number of connections allowed by the installed TCP/IP. **VSE/ESA systems:** The BACKLOG parameter is ignored. The installed TCP/IP determines the backlog value for a given socket.

G.2.28 NTOHL

NTOHL converts a fullword integer from network byte order to host byte order. Within Advantage CA-IDMS, host and network byte order are the same. Therefore, the NTOHL function does not apply to the mainframe environment; it is implemented for the application programmer's convenience.

Assembler

```
label    #SOCKET NTOHL,
               FIELDIN=input-field,
               FIELDOUT=output-field,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-NTOHL,
input-field,
output-field
```

G.2.28.1 Parameters

Parameter	Description
input-field	The name of a fullword field containing the integer to convert.
output-field	The name of a fullword field that receives the converted integer.

G.2.29 NTOHS

NTOHS converts a halfword integer from network byte order to host byte order. Within Advantage CA-IDMS, host and network byte order are the same. Therefore, the NTOHS function does not apply to the mainframe environment; it is implemented for the application programmer's convenience.

Assembler

```
label    #SOCKET NTOHS,
               FIELDIN=input-field,
               FIELDOUT=output-field,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-NTOHS,
input-field,
output-field
```

G.2.29.1 Parameters

Parameter	Description
input-field	The name of a halfword field containing the integer to convert.
output-field	The name of a halfword field that receives the converted integer.

G.2.30 READ

READ reads a number of bytes from a socket into an area.

Assembler

```
label    #SOCKET READ,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               BUFFER=buffer,
               BUFFERL=buffer-length,
               RETLEN=read-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-READ,
return-code,
errno,
reason-code,
socket-descriptor,
buffer,
buffer-length,
read-length
```

G.2.30.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor to read from.
buffer	The name of the area where the data is to be placed.
buffer-length	The name of a fullword field containing the length of the *buffer*. *buffer-length* can be specified as an absolute expression.
read-length	The name of a fullword field in which the actual length of the data read is returned.

G.2.30.2 Notes

When the timeout value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket timeouts, refer to 9.5.3, “Associating Timeouts to Sockets” on page 9-25.

G.2.31 RECV

RECV reads a number of bytes from a connected socket into an area.

Assembler

```
label     #SOCKET RECV,
                RETCODE=return-code,
                ERRNO=errno,
                RSNCODE=reason-code,
                SOCK=socket-descriptor,
                BUFFER=buffer,
                BUFFERL=buffer-length,
                FLAGS=flags,
                RETLEN=read-length,
                PLIST=parameter-list-area,
                RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-RECV,
return-code,
errno,
reason-code,
socket-descriptor,
buffer,
buffer-length,
flags,
read-length
```

G.2.31.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor from which to read.
buffer	The name of the area where the data is to be placed.
buffer-length	The name of a fullword field containing the length of the *buffer*. *buffer-length* can be specified as an absolute expression.
flags	The name of a fullword field containing information on how the data is to be received. **VSE/ESA systems:** MSG@PEEK is the only flag value that is supported. The remaining flags are not supported and returns an error if specified. When the MSG@PEEK flag is specified only the first byte of the RECV buffer is returned, even if a larger buffer size is specified. **z/VM systems:** MSG@WALL is not supported.
read-length	The name of a fullword field in which the actual length of the data read is returned.

G.2.31.2 Notes

- When the timeout value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket timeouts, refer to 9.5.3, “Associating Timeouts to Sockets” on page 9-25.
- The following table lists the flags that can be specified. The EQUate symbol is generated by the MSGFLAGS DSECT by the #SOCKET TCPIPDEF macro call and the field names are associated with the SOCKET-MISC-DEFINITIONS.

EQUate Symbol	Field Name	Description
MSG@DROU	SOCKET-MSGFLAGS-DONTROUTE	Send without network routing
MSG@OOB	SOCKET-MSGFLAGS-OOB	Send and receive out-of-band data
MSG@PEEK	SOCKET-MSGFLAGS-PEEK	Peek at incoming data
MSG@WALL	SOCKET-MSGFLAGS-WAITALL	Wait until all data returned

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

G.2.32 RECVFROM

RECVFROM reads a number of bytes from a datagram socket into an area.

Assembler

```
label    #SOCKET RECVFROM,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               BUFFER=buffer,
               BUFFERL=buffer-length,
               FLAGS=flags,
               SOCKADDR=sockaddr,
               SOCKADDL=sockaddr-length,
               RETLEN=read-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-RECVFROM,
return-code,
errno,
reason-code,
socket-descriptor,
buffer,
buffer-length,
flags,
sockaddr,
sockaddr-length,
read-length
```

G.2.32.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor from which to read.
buffer	The name of the area where the data is to be placed.
buffer-length	The name of a fullword field containing the length of the *buffer*. *buffer-length* can be specified as an absolute expression.
flags	The name of a fullword field containing information on how the data is to be received. The list of the different flags supported can be found in the MSGFLAGS DSECT generated by the #SOCKET TCPIPDEF macro call and in the SOCKET-MISC-DEFINITIONS record for other languages. See the RECV function description for an explanation of flags that can be specified.
sockaddr	The name of an area in which to return the sockaddr structure of the sender of the data. The format of this structure depends on the domain of the corresponding socket. This parameter can be assigned to NULL if the caller is not interested in the sender's address.
sockaddr-length	The name of a fullword field containing the length of *sockaddr*. If SOCKADDR is assigned to NULL, *sockaddr-length* must be 0. On return, *sockaddr-length* contains the size required to represent the socket. If the size of *sockaddr* is too small to contain the full sockaddr structure, it is truncated. The maximum value for this parameter is 4096.
read-length	The name of a fullword field in which the actual length of the data read is returned.

G.2.32.2 Notes

- When the timeout value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket timeouts, refer to 9.5.3, “Associating Timeouts to Sockets” on page 9-25.
- **VSE/ESA systems:** The RECVFROM function is not supported.

G.2.33 SELECT and SELECTX

SELECT synchronizes processing of several sockets operating in non-blocking mode. Sockets that are ready for reading, ready for writing, or have a pending exceptional condition can be selected. If no sockets are ready for processing, SELECT can block indefinitely or wait for a specified period of time (which may be zero) and then return.

SELECT examines the socket descriptors specified by *read-list*, *write-list*, and *exception-list* to see if some are ready for reading, ready for writing, or have an exceptional condition pending, respectively. On return, SELECT updates each of the lists to indicate which socket descriptors are ready for the requested operation. The total number of ready descriptors in all the lists is returned.

SELECTX has the same functionality as SELECT with the additional capability of waiting on one or more ECBs in addition to a time interval. This allows interruption of a wait if an external event occurs.

Assembler

```
label     #SOCKET SELECT,
                RETCODE=return-code,
                ERRNO=errno,
                RSNCODE=reason-code,
                NFDS=number-of-socket-descriptors,
                READLST=read-list,
                READLSTL=read-list-length,
                WRITLST=write-list,
                WRITLSTL=write-list-length,
                EXCELST=exception-list,
                EXCELSTL=exception-list-length,
                OPTION=option,
                TIMEOUT=timeval-structure,
                RETNFDS=returned-number-of-descriptors,
                PLIST=parameter-list-area,
                RGSV=(rgsv)

label     #SOCKET SELECTX,
                RETCODE=return-code,
                ERRNO=errno,
                RSNCODE=reason-code,
                NFDS=number-of-socket-descriptors,
                READLST=read-list,
                READLSTL=read-list-length,
                WRITLST=write-list,
                WRITLSTL=write-list-length,
                EXCELST=exception-list,
                EXCELSTL=exception-list-length,
                OPTION=option,
                TIMEOUT=timeval-structure,
                ECB=ecb,
                ECBLIST=ecb-list,
                RETNFDS=returned-number-of-descriptors,
                PLIST=parameter-list-area,
                RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-SELECT,
return-code,
errno,
reason-code,
number-of-socket-descriptors,
read-list,
read-list-length,
write-list,
write-list-length,
exception-list,
exception-list-length,
option,
timeval-structure,
returned-number-of-descriptors

SOCKET-FUNCTION-SELECTX,
return-code,
errno,
reason-code,
number-of-socket-descriptors,
read-list,
read-list-length,
write-list,
write-list-length,
exception-list,
exception-list-length,
option,
timeval-structure,
ecb,
ecb-list,
returned-number-of-descriptors
```

G.2.33.1 Parameters

Parameter	Description
number-of-socket-descriptors	The name of a field containing the highest socket descriptor specified in any of the lists + 1. Only socket descriptors whose value is less than *number-of-socket-descriptors* are considered in servicing the request.
read-list	The name of an area containing a bit list identifying the socket descriptors to be examined for a "ready to read" condition. Only socket descriptors whose corresponding bit in the bit list is on are considered. On return, the bits that are set indicate the descriptors that are ready to read. Specify NULL if the *read-list* is to be ignored.
read-list-length	The name of a fullword field containing the length in bytes of *read-list*. *read-list-length* can be specified as an absolute expression. *read-list-length* must be a multiple of 4; specify 0 if the *read-list* is to be ignored.

Parameter	Description
write-list	The name of an area containing a bit list identifying the socket descriptors to be examined for a "ready to write" condition. Only socket descriptors whose corresponding bit in the bit list is on are considered. On return, the bits that are set indicate the descriptors that are ready to write. Specify NULL if the *write-list* is to be ignored.
write-list-length	The name of a fullword field containing the length in bytes of *write-list*. *write-list-length* can be specified as an absolute expression. *write-list-length* must be a multiple of 4; specify 0 if the *write-list* is to be ignored.
exception-list	The name of an area containing a bit list identifying the socket descriptors to be examined for an exception condition. Only socket descriptors whose corresponding bit in the bit list is on are considered. On return, the bits that are set indicate the descriptors that have had exceptions. Specify NULL if the *exception-list* is to be ignored.
exception-list-length	The name of a fullword field containing the length in bytes of *exception-list*. *exception-list-length* can be specified as an absolute expression. *exception-list-length* must be a multiple of 4; specify 0 if the *exception-list* is to be ignored.
option	Name of a fullword field containing the way the different bits are interpreted in the different bit-lists. *option* can be specified as an absolute expression. See G.2.33.2, “Notes” on page G-47 for a list of *options* that can be specified.
timeval-structure	The name of the area containing the TIMEVAL structure. If the parameter is assigned to NULL, SELECT waits until at least one of the descriptors is ready. If the timeout value (number of seconds + number of microseconds) is 0, SELECT checks the descriptors and returns immediately without waiting. The TIMEVAL structure is generated by the #SOCKET TCPIPDEF macro call and described in the SOCKET-TIMEVAL record.
returned-number-of-descriptors	The name of a fullword field in which the total number of ready descriptors is returned.
ecb	The name of an area containing an Advantage CA-IDMS ECB.
ecb-list	The name of an area containing an Advantage CA-IDMS ECB list. Each entry in the ECB list is represented by two fullwords: ■ The first fullword is a pointer to the ECB. ■ The second fullword is zero, except for the last entry in the list. In this case the high-order bit is turned ON to identify the end of the ECB list.

G.2.33.2 Notes

- Refer to FD_ZERO, FD_CLR, FD_SET and FD_ISSET #SOCKET function for more information about manipulating bits in bit lists.
- For programming languages like COBOL and Advantage CA-ADS where it is difficult to manipulate bits in bit lists, byte lists can be used by specifying a SOCKET-SELECT-BYTELIST for *option*. In this case, the *read-list*, *write-list* and *exception-list* are byte lists instead of bit lists. In byte lists, each byte represents one socket descriptor. A socket descriptor will be processed if its corresponding byte is set to the character '1'. A socket descriptor's corresponding byte is the *n*th byte relative to 1 in the list, where *n* is equal to the value of socket descriptor + 1.
- ECB and ECBLIST are mutually exclusive parameters.
- **z/VM systems:** If multiple TCP/IP stacks are used, all the sockets represented by a bit in the 3 bit lists must be created in the same TCP/IP stack.
- **VSE/ESA systems:** The SELECT and SELECTX functions are not supported.
- The following table lists the options that can be specified. The EQUate symbol is generated by the #SOCKET TCPIPDEF macro call and the field names are associated with the SOCKET-MISC-DEFINITIONS.

EQUate Symbol	Field Name	Description
SEL@BBKW	SOCKET-SELECT-BITBACKWARD	Specifies the bits in the fullwords are in the backward order. This is the default value if the parameter is assigned to NULL.
SEL@BFRW	SOCKET-SELECT-BITFORWARD	Specifies the bits in each fullword are in the forward order
SEL@BYTV	SOCKET-SELECT-BYTELIST	The *read-list*, *write-list*, and *exception-list* are byte lists instead of bit lists.

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

G.2.34 SEND

SEND sends data on a connected socket.

Assembler

```
label    #SOCKET SEND,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               BUFFER=buffer,
               BUFFERL=buffer-length,
               FLAGS=flags,
               RETLEN=sent-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-SEND,
return-code,
errno,
reason-code,
socket-descriptor,
buffer,
buffer-length,
flags,
sent-length
```

G.2.34.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor on which to do the send.
buffer	The name of the area containing the data to be sent.
buffer-length	The name of a fullword field containing the length of the *buffer*. *buffer-length* can be specified as an absolute expression.
flags	The name of a fullword field containing information on how the data is to be sent. The list of the different flags supported can be found in the MSGFLAGS DSECT generated by the #SOCKET TCPIPDEF macro call and in the SOCKET-MISC-DEFINITIONS record for other languages. See the RECV function description for an explanation of flags that can be specified. **VSE/ESA systems:** No flag values are supported and an error is returned if a value is specified.
sent-length	The name of a fullword field in which the actual length of the data sent is returned.

G.2.34.2 Notes

When the timeout value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket timeouts, refer to 9.5.3, “Associating Timeouts to Sockets” on page 9-25.

G.2.35 SENDTO

SENDTO sends data on a datagram socket.

Assembler

```
label    #SOCKET SENDTO,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               BUFFER=buffer,
               BUFFERL=buffer-length,
               FLAGS=flags,
               SOCKADDR=sockaddr,
               SOCKADDL=sockaddr-length,
               RETLEN=sent-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-SENDTO,
return-code,
errno,
reason-code,
socket-descriptor,
buffer,
buffer-length,
flags,
sockaddr,
sockaddr-length,
sent-length
```

G.2.35.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor on which to do the send.
buffer	The name of the area containing the data to be sent.
buffer-length	The name of a fullword field containing the length of the *buffer*. *buffer-length* can be specified as an absolute expression.

Parameter	Description
flags	The name of a fullword field containing information on how the data is to be sent. The list of the different flags supported can be found in the MSGFLAGS DSECT generated by the #SOCKET TCPIPDEF macro call and in the SOCKET-MISC-DEFINITIONS record for other languages. See the RECV function description for an explanation of flags that can be specified.
sockaddr	The name of an area containing the sockaddr structure describing where data is to be sent. The format of this structure depends on the domain of the corresponding socket.
sockaddr-length	The name of a fullword field containing the length of *sockaddr*. *Sockaddr-length* can be specified as an absolute expression. The maximum value for this parameter is domain dependent. If the domain is: ■ AF_INET — it is the length of the SOCKET-SOCKADDR-IN record (SIN#LEN for assembler) ■ AF_INET6 — it is the length of the SOCKET-SOCKADDR-IN6 record (SIN6#LEN for assembler)
sent-length	The name of a fullword field in which the actual length of the data sent is returned.

G.2.35.2 Notes

- When the timeout value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket timeouts, refer to 9.5.3, “Associating Timeouts to Sockets” on page 9-25.
- **VSE/ESA systems:** The SENDTO function is not supported.

G.2.36 SETSOCKOPT

SETSOCKOPT sets options associated with a socket.

Assembler

```
label     #SOCKET SETSOCKOPT,
                  RETCODE=return-code,
                  ERRNO=errno,
                  RSNCODE=reason-code,
                  SOCK=socket-descriptor,
                  LEVEL=level,
                  OPTNAME=option-name,
                  OPTVAL=option-value,
                  OPTLEN=option-value-length,
                  PLIST=parameter-list-area,
                  RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-SETSOCKOPT,
return-code,
errno,
reason-code,
socket-descriptor,
level,
option-name,
option-value,
option-value-length
```

G.2.36.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor for which the service is to be performed.
level	The name of a fullword field containing the level for the *option*. *level* can be specified as an absolute expression.
option-name	The name of a fullword field indicating the *option* to set. *option-name* can be specified as an absolute expression.
option-value	The name of an area containing the data to associate with the socket.
option-value-length	The name of a fullword field containing the length of *option-value*. *option-value-length* can be specified as an absolute expression. The maximum value for this parameter is 16.

G.2.36.2 Notes

- The list of level and options currently supported are listed by the #SOCKET TCPIPDEF macro call for assembler and in the SOCKET-MISC-DEFINITIONS record for other languages. See GETSOCKOPT for a description of the options that can be specified.
- **VSE/ESA systems:** Only the SO@REUSE option is supported.

G.2.37 SETSTACK

SETSTACK sets the requested TCP/IP stack affinity for the current executing Advantage CA-IDMS task.

Assembler

```
label    #SOCKET SETSTACK,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               NAME=stack-name,
               NAMEL=stack-name-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-SETSTACK,
return-code,
errno,
reason-code,
stack-name,
stack-name-length
```

G.2.37.1 Parameters

Parameter	Description
stack-name	The area containing the name of the TCP/IP stack to set. This name can be the JOBNAME of the corresponding TCPIP stack, a *hostname* or an IP-address in binary or string format.
stack-name-length	The name of a fullword field containing the length of *stack-name*. *stack-name-length* can be specified as an absolute expression. The maximum value for this parameter is 256.

G.2.37.2 Notes

- To clear TCP/IP stack affinity for the current task, call the SETSTACK function using *stack-name* value equal to '*ALL'.
- To restore the default TCP/IP stack affinity for the current task, call the SETSTACK function using *stack-name* value equal to '*DEFAULT'.
- Refer to 9.5.2, “Using Multiple TCP/IP Stacks” on page 9-24 for more information.

G.2.38 SHUTDOWN

SHUTDOWN gracefully shuts down all or part of a duplex socket connection.

Assembler

```
label    #SOCKET SHUTDOWN,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               HOW=how-condition,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-SHUTDOWN,
return-code,
errno,
reason-code,
socket-descriptor,
how-condition
```

G.2.38.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor to shut down.
how-condition	The name of a fullword field indicating the effect of the shutdown on read and write operations. *how-condition* can be specified as an absolute expression.

G.2.38.2 Notes

The following table lists the conditions that can be specified. The EQUate symbol is generated by the #SOCKET TCPIPDEF macro call and the field names are located in the SOCKET-MISC-DEFINITIONS record. They are:

EQUate Symbol	Field Name	Description
SHUT_R	SOCKET-SHUTDOWN-READ	Terminate read communication (from the socket)
SHUT_W	SOCKET-SHUTDOWN-WRITE	Terminate write communication (to the socket)
SHUT_RW	SOCKET-SHUTDOWN-READ-WRITE	Terminate both read and write communication

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

G.2.39 SOCKET

SOCKET creates a socket in a communications domain.

Assembler

```
label     #SOCKET SOCKET,
                  RETCODE=return-code,
                  ERRNO=errno,
                  RSNCODE=reason-code,
                  DOMAIN=domain,
                  TYPE=type,
                  PROTNUM=protocol-number,
                  NEWSOCK=new-socket-descriptor,
                  PLIST=parameter-list-area,
                  RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-SOCKET,
return-code,
errno,
reason-code,
domain,
type,
protocol-number,
new-socket-descriptor
```

G.2.39.1 Parameters

Parameter	Description
domain	The name of a fullword field containing the domain or address family of the socket. See the G.2.39.2, “Notes” on page G-55 for a list of domains that can be specified.
type	The name of a fullword field containing the type of the socket. *type* can be specified as an absolute expression. See the G.2.39.2, “Notes” on page G-55 for a list of socket types that can be specified.
protocol-number	The name of a fullword field containing the protocol. *protocol-number* can be specified as an absolute expression. See the G.2.39.2, “Notes” on page G-55 for a list of supported protocols.
new-socket-descriptor	The name of a fullword field where the newly created socket descriptor is returned.

G.2.39.2 Notes

- These SYSIDMS parameters can be used to control sockets:
  - TCP/IP_MAXIMUM_SOCKETS — Specifies the maximum number of sockets created globally in the DC/UCF system.
  - TCP/IP_MAXIMUM_SOCKETS_PER_TASK — Specifies the maximum number of sockets created by a single task.

  Refer to *Advantage CA-IDMS Common Facilities* for more information.
- The following table lists the domains that can be specified. The EQUate symbol is generated by the #SOCKET TCPIPDEF macro call and the field names are located in the SOCKET-MISC-DEFINITIONS record. They are:

EQUate Symbol	Field Name	Description
AF@INET [1]	SOCKET-FAMILY-AFINET	AF_INET address family
AF@INET6	SOCKET-FAMILY-AFINET6	AF_INET6 address family
VSE/ESA systems: [1] — Only supports DOMAIN=AF@INET		

- The following table lists the socket types that can be specified. The EQUate symbol is generated by the #SOCKET TCPIPDEF macro call and the field names are located in the SOCKET-MISC-DEFINITIONS record. They are:

EQUate Symbol	Field Name	Description
STREAM [1]	SOCKET-TYPE-STREAM	Stream — connection oriented and reliable
DATAGRAM	SOCKET-TYPE-DATAGRAM	Datagram — connectionless and unreliable
VSE/ESA systems: [1] — Only supports TYPE=STREAM.		

- The following table lists the protocols that can be specified. The EQUate symbol is generated by the #SOCKET TCPIPDEF macro call and the field names are located in the SOCKET-MISC-DEFINITIONS record. They are:

EQUate Symbol	Field Name	Description
PROTIP	SOCKET-PROTOCOL-IP	Default protocol
PROTTCP [1]	SOCKET-PROTOCOL-TCP	TCP protocol
PROTUDP	SOCKET-PROTOCOL-UDP	UDP protocol

EQUate Symbol	Field Name	Description
PROTIPV6	SOCKET-PROTOCOL-IPV6	IPv6 protocol

VSE/ESA systems: [1] — Only supports PROTNUM=PROTTCP

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

G.2.40 WRITE

WRITE sends data on a socket.

Assembler

```
label    #SOCKET WRITE,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               BUFFER=buffer,
               BUFFERL=buffer-length,
               RETLEN=sent-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-WRITE,
return-code,
errno,
reason-code,
socket-descriptor,
buffer,
buffer-length,
sent-length
```

G.2.40.1 Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the socket descriptor on which to send.
buffer	The name of the area containing the data to be sent.
buffer-length	The name of a fullword field containing the length of the *buffer*. *buffer-length* can be specified as an absolute expression.
sent-length	The name of a fullword field in which the actual length of the data sent is returned.

G.2.40.2 Notes

When the timeout value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket timeouts, refer to 9.5.3, “Associating Timeouts to Sockets” on page 9-25.

G.3 Return, Errno, and Reason Codes

The return code value returned by a call to the socket program interface can be a binary 0 (call successfully executed) or non-zero (an error occurred). In the latter case, the errno field explains why the function call failed. Two different situations arise:

- Advantage CA-IDMS generates the error. Errno is set to a value in the range 12000-12999 as documented below. The reason code is not used and is 0.
- The error is generated by operating system services. Errno and (where applicable) reason-code are documented in the appropriate operating system services documentation.
  - z/OS and OS/390 — Refer to the *UNIX System Services - Messages and Codes*
  - z/VM — refer to the *z/VM TCP/IP Programmer's Reference*
  - VSE/ESA— refer to the following resources:
    - *Connectivity Systems TCP/IP for VSE: Programmer's Guide*
    - *Barnard Systems TCP/IP Tools*
    - *TCP/IP for VSE/ESA - IBM Program Setup and Supplementary Information*
- **z/VM systems:** For some ERRNO codes returned by Advantage CA-IDMS, the variable assigned to the RSNCODE parameter may contain the IPRCODE extracted from the corresponding IUCV parameter list. See the IPARML DSECT for the complete list of values.
- **VSE/ESA and z/VM systems:** The value of some ERRNO codes can differ from the equivalent standard POSIX value that is returned on z/OS. For example, the standard value for ETIMEDOUT ERRNO code (connection timed out) is 1127, but is 60 on z/VM. The standard ERRNO code is returned to the variable assigned to the ERRNO parameter. Applications must check the variable for common ERRNO codes that are handled programatically. The ERRNO code value returned by the operating system is saved in a variable assigned to the RSNCODE parameter.

G.3.1 ERRNO Numbers Set By The Socket Program Interface

The name shown in the following table is the EQUate symbol generated by the #SOCKET macro. The equivalent condition name in the SOCKET-CALL-INTERFACE record is prefixed with:

- SOCKET-ERRNO- — for COBOL and Advantage CA-ADS
- SOCKET_ERRNO_ — for PL/I

Name	Value	Description
	1 - 11999	The ERRNO is generated by the operating system. Refer to the appropriate operating system documentation.
RNOINPL	12000	Invalid #SOCKET parameter list
RNOINAEC	12001	Invalid ASYNCECB parameter
RNOINAII	12002	Invalid AINFOIN parameter
RNOINAIO	12003	Invalid AINFOOUT parameter
RNOINBF	12004	Invalid BUFFER parameter
RNOINBFL	12005	Invalid BUFFERL parameter
RNOINBKL	12006	Invalid BACKLOG parameter
RNOINCAL	12007	Invalid CANONAML parameter
RNOINCMD	12008	Invalid COMMAND parameter
RNOINDOM	12009	Invalid DOMAIN parameter
RNOINEL	12010	Invalid EXCELST parameter
RNOINELL	12011	Invalid EXCELSTL parameter
RNOINFLG	12012	Invalid FLAGS parameter
RNOINFMT	12013	Invalid FORMAT parameter
RNOINFLT	12014	Invalid FROMLTE parameter
RNOINHDL	12015	Invalid HANDLE parameter
RNOINHNA	12016	Invalid HOSTNAME parameter
RNOINHNL	12017	Invalid HOSTNAML parameter
RNOINHNT	12018	Invalid HOSTENTP parameter
RNOINHOW	12019	Invalid HOW parameter
RNOINIL	12020	Invalid IPADDRL parameter
RNOINIP	12021	Invalid IPADDR parameter
RNOINIPS	12022	Invalid IPADDRS parameter
RNOINISL	12023	Invalid IPADDRSL parameter
RNOINLEV	12024	Invalid LEVEL parameter
RNOINMXP	12025	Invalid MAXPTERM parameter
RNOINMXT	12026	Invalid MAXTASK parameter
RNOINNA	12027	Invalid NAME parameter

Name	Value	Description
RNOINNAL	12028	Invalid NAMEL parameter
RNOINNS	12029	Invalid NEWSOCK parameter
RNOINNSD	12030	Invalid NFDS parameter
RNOINONA	12031	Invalid OPTNAME parameter
RNOINOVA	12032	Invalid OPTVAL parameter
RNOINOVL	12033	Invalid OPTLEN parameter
RNOINPNA	12034	Invalid PROTNAME parameter
RNOINPNL	12035	Invalid PROTNAML parameter
RNOINPNT	12036	Invalid PROTENTP parameter
RNOINPNU	12037	Invalid PROTNUM parameter
RNOINPOR	12038	Invalid PORT parameter
RNOINRHL	12039	Invalid RETHNAML parameter
RNOINRIL	12040	Invalid RETIPASL parameter
RNOINRL	12041	Invalid READLST parameter
RNOINRLL	12042	Invalid READLSTL parameter
RNOINRLN	12043	Invalid RETLEN parameter
RNOINRND	12044	Invalid RETNFDS parameter
RNOINRNS	12045	Invalid RETNSTKS parameter
RNOINSA	12046	Invalid SOCKADDR parameter
RNOINSAL	12047	Invalid SOCKADDL parameter
RNOINSNA	12048	Invalid SERVNAME parameter
RNOINSNL	12049	Invalid SERVNAML parameter
RNOINSNT	12050	Invalid SERVENTP parameter
RNOINSOC	12051	Invalid SOCK parameter
RNOINTLT	12052	Invalid TOLTE parameter
RNOINTYP	12053	Invalid TYPE parameter
RNOINWL	12054	Invalid WRITLST parameter
RNOINWLL	12055	Invalid WRITLSTL parameter
RNOINOPT	12056	Invalid OPTION parameter
RNOINTIM	12057	Invalid TIMEOUT parameter
RNOINARG	12058	Invalid ARGUMENT PARAMETER

Name	Value	Description
RNOINRV	12059	Invalid RETVAL parameter
RNOINECB	12060	Invalid ECB parameter
RNOINECL	12061	Invalid ECBLIST parameter
RNOINRSL	12062	Invalid RETSNAML parameter
RNOINBL	12063	Invalid BITLIST parameter
RNOINBLL	12064	Invalid BITLISTL parameter
RNOINBOR	12065	Invalid BITORDER parameter
RNO2BUFF	12100	Specify BUFFER and BUFFERL, or none of them
RNO2HNAM	12101	Specify HOSTNAME and HOSTNAML, or none of them
RNO2NAME	12102	Specify NAME and NAMEL, or none of them
RNO2PNAM	12103	Specify PROTNAME and PROTNAML, or none of them
RNO2SNAM	12104	Specify SERVNAME and SERVNAML, or none of them
RNO3HNAM	12105	Specify HOSTNAME/HOSTNAML/ RETHNAML, or none
RNO3SNAM	12106	Specify SERVNAME/SERVNAML/ RETSNAML, or none
RNORQHS	12107	HOSTNAME or SERVNAME (or both) is required
RNORQECB	12108	ECB or ECBLIST is required
RNOXCECB	12109	ECB and ECBLIST are mutually exclusive
RNOIECBL	12110	Invalid ECB in ECBLIST
RNOINARQ	12111	Invalid asynchronous command request
RNOINAIS	12112	Invalid ADDRINFO structure
RNOSYSP1	12113	ASYNCECB and HANDLE are system parms
RNOINHDA	12114	Invalid area pointed to by HANDLE
RNOIIPA	12115	Invalid format for IP-address
RNOIIPA6	12116	Invalid format for IP-address (V6)

Name	Value	Description
RNOFNS	12200	Function not supported by interface
RNOFRSVD	12201	Function reserved for the system
RNOCAAIO	12202	Cannot allocate an AIO parameter list
RNOCANSU	12203	Cannot assign new socket to user
RNOCRSFU	12204	Cannot remove socket from user table
RNOCSHNT	12205	Cannot save HOSTENT structure info
RNOCSAIO	12206	Cannot save ADDRINFO structure info
RNONAINF	12207	Cannot find ADDRINFO to free
RNONOLTE	12208	No LTE available from current TCE
RNOSLIND	12209	SOCKET line not defined
RNOSLINO	12210	SOCKET line not opened
RNOSLRCY	12211	SOCKET line has been recycled
RNOPINL	12212	Plug-in module not loaded
RNODRTCE	12213	Driver's TCE doesn't point to the PLE
RNOINEPI	12214	Invalid environment when entering the plug-in
RNOSENA	12215	Socket environment not active
RNOUSTCA	12216	User's socket table cannot be allocated
RNOUSTNE	12217	User's socket table does not exist
RNOSSTCA	12218	System's socket table cannot be allocated
RNOSSTNE	12219	System's socket table does not exist
RNOSTKNF	12220	Requested stack not found
RNOSTKNA	12221	Requested stack not active
RNOSDTCE	12222	Socket Descriptor table cannot be extended
RNOCASWA	12223	Cannot allocate SELECT work area
RNOINSWA	12224	Inconsistent fields in SELECT work area
RNOSBLEM	12225	All SELECT bit lists are empty
RNOSNCSS	12226	All sockets not created under same stack
RNOCASBL	12227	Cannot allocate socket's bit list
RNOMAXSO	12228	Maximun number of sockets reached

Name	Value	Description
RNOMAXST	12229	Maximum number of sockets per task reached
RNOCADNS	12230	Cannot allocate DNS work area
RNOINDNS	12231	Invalid response from DNS server
RNOPITNL	12232	Plugin table module not loaded [1]
RNOHIUCV	12300	HNDIUCV error
RNOCIUCV	12301	CMSIUCV error
RNOIUCVS	12302	IUCV error for a socket function
RNOSEVER	12303	IUCV connection severed by TCP/IP
RNOTOIUC	12304	Time out during IUCV connection
	>12999	The ERRNO is generated by the operation system. Refer to the appropriate operating system documentation.

Note: [1] — VSE only

G.4 Socket Structure Descriptions

G.4.1 ADDRINFO Structure

The ADDRINFO structure is input and output to the GETADDRINFO function call.

Field	Description
Flags	A set of flags
Family	Address family (AF_INET or AF_INET6)
Socket type	Type of socket (STREAM or DATAGRAM)
Protocol	Protocol in use for the socket
SOCKADDR length	Length of SOCKADDR structure
Canonical name	Address of canonical name associated with input node name
SOCKADDR structure	Address of the SOCKADDR structure
New ADDRINFO	Address of next ADDRINFO structure

G.4.2 HOSTENT Structure

The HOSTENT structure is returned by the GETHOSTBYADDR and GETHOSTBYNAME function calls.

Field	Description
Hostname	Address of hostname (null-terminated string)
Aliases	Address of a zero-terminated array of pointers to aliases, which are null-terminated strings
Address type	Address family of returned IP addresses (AF_INET or AF_INET6)
Address length	Length of returned IP addresses
Addresses	Address of a zero-terminated array of pointers to IP addresses

G.4.3 SOCKADDR Structure

The SOCKADDR structure describes the address of a socket. There are two versions of this structure: IPv4 and IPv6.

G.4.3.1 SOCKADDR for IPv4

Field	Description
Family	A 2-byte field describing the socket address family type: AF_INET
Port number	The port number for this socket
Address	The 4-byte IP address of the TCP/IP stack
Zeros	Eight bytes of binary zeros

G.4.3.2 SOCKADDR for IPv6

Field	Description
Family	A 2-byte field describing the socket address family type: AF_INET6
Port number	The port number for this socket
Flow	Flow information
Address	The 16-byte IP address of the TCP/IP stack
Scope ID	Scope identifier

G.4.4 TIMEVAL Structure

The TIMEVAL structure may be passed as input to the SELECT and SELECTX function calls in order to specify a wait interval.

Field	Description
Seconds	Number of seconds to wait
Microseconds	Number of microseconds to wait.

G.5 String Conversion Functions

Different encoding schemes exist for representing strings. On mainframe computers, EBCDIC is often used, while on other platforms ASCII or UNICODE is used. Currently, Advantage CA-IDMS does not support UNICODE. However, conversion from EBCDIC to ASCII and vice versa can be implemented with the new IDMSIN01 function STRCONV, which is described below. For more information about IDMSIN01, refer to "*Advantage CA-IDMS Callable Services.*"

STRCONV converts a string in a buffer by replacing the old string with the new one. The conversion uses tables defined in RHDCCODE:

- To convert from ASCII to EBCDIC, EBCTAB is used.
- To convert from EBCDIC to ASCII, ASCTAB is used.

The tables delivered on the installation tape contain the EBCDIC IBM-037 and ASCII ISO8859-1 tables.

G.5.1 Assembler

Assembler programs use the IDMSIN01 macro to invoke the character conversion functionality as follows:

```
IDMSIN01 STRCONV,CONVFUN=convfun,                              X
      BUFFER=buffer,BUFFERL=bufferl
```

G.5.2 COBOL

COBOL programs use the CALL IDMSIN01 interface to invoke the character conversion functionality as follows:

Define these variables:

```
01  RPB                    PIC X(36).
01  IN01-REQ.
    02 REQUEST-CODE        PIC S9(8) COMP.
    02 REQUEST-RETURN      PIC S9(8) COMP.
01  IN01-STRFUNC           PIC X(4).
01  buffer                 PIC X(80).
01  bufferl                PIC S9(8) COMP.
```

Code the call as follows:

```
    MOVE 34 TO REQUEST-CODE.
    MOVE 'convfun' TO IN01-STRFUNC.
    CALL 'IDMSIN01' USING RPB,
                          IN01-REQ,
                          IN01-STRFUNC,
                          buffer,
                          bufferl.
```

G.5.3 PL/I

PL/I programs use the CALL IDMSIN01 interface to invoke the character conversion functionality as follows:

Define these variables:

```
DCL RPB                  CHAR (36);
DCL 01 IN01_REQ,
    02 REQUEST_CODE      FIXED BINARY(31),
    02 REQUEST_RETURN    FIXED BINARY(31);
DCL IN01_STRFUNC         FIXED BINARY(31);
DCL buffer               CHAR (80);
DCL bufferl              FIXED BINARY(31);
```

Code the call as follows:

```
REQUEST_CODE = 34;
IN01_STRFUNC = 'convfun';
CALL 'IDMSIN01' ( RPB,
                  IN01_REQ,
                  IN01_STRFUNC,
                  buffer,
                  bufferl);
```

G.5.4 Parameters

convfun
The function to execute. To convert a string from ASCII to EBCDIC, specify 'ATOE'. To convert a string from EBCDIC to ASCII, specify 'ETOA'.

buffer
The name of the area that contains the string to convert.

bufferl
The name of a fullword field containing the length in bytes of the string.