




# OS/390 Reference Guide



eXadas™ Data Integrator  
Version 2.2  
October, 2002

October, 2002

Copyright © 1993–2002 CrossAccess Corporation. All rights reserved.

This publication may not be reproduced, stored in a retrieval system or transmitted in whole or part, in any form or by any means, electronic, photocopy, recording or otherwise, without the prior written consent of CrossAccess Corporation.

The CrossAccess Corporation logo, eXadas, eXadas Data Integrator, and XDi are trademarks of CrossAccess Corporation. Other products and services referred to herein are or may be the trademarks of their respective owners.

CrossAccess Corporation  
One Tower Lane, Suite1700  
Oak Brook Terrace IL 60181  
(630) 928-3708 or (800) 427-6774  
FAX: (630) 954-0554

CrossAccess Corporation  
2900 Gordon Ave., Suite 100  
Santa Clara CA 95051  
(408) 735-7545 or (800) 982-9911  
FAX: (408) 735-0328

Technical Support: (800) 982-9911

# Table of Contents

---

<b>Chapter 1</b>	<b>Overview .....</b>	<b>1</b>
	<b>Introduction to eXadas .....</b>	<b>1</b>
	Product Overview.....	2
	<b>Operational Components.....</b>	<b>3</b>
	eXadas Server .....	3
	Region Controller .....	4
	Connection Handlers.....	4
	Query Processor.....	5
	Logger.....	6
	Initialization Services .....	6
	Enterprise Server.....	7
	Client Interface Module .....	7
	Client Connectors.....	8
	<b>Application Components .....</b>	<b>9</b>
	<b>Administrative Components.....</b>	<b>9</b>
	DataMapper.....	9
<b>Chapter 2</b>	<b>Deploying Applications.....</b>	<b>13</b>
	<b>Introduction to Deploying Applications .....</b>	<b>13</b>
	<b>Departments and Responsibilities.....</b>	<b>14</b>
	<b>Application Models .....</b>	<b>16</b>
	Types of Applications .....	16
	Selecting an Interface.....	16
	Connectors .....	16
	Precompiler.....	17
	<b>Initial Development .....</b>	<b>17</b>
	Creating Your Own Server .....	18
	Mapping Your Data .....	19
	Mapping Verification.....	20
	Creating Your Own Queries .....	22
	General Considerations.....	22
	Limiting Query Output and Performance Monitoring.....	23
	IMS Recommendations.....	24

---

Developing Your Application.....	25
<b>Deployment .....</b>	<b>26</b>
Determining How Many Servers Your Site Requires.....	27
Server Deployment Options.....	27
Creating Production Servers .....	28
Server Set-up Worksheet Instructions .....	30
eXadas Enterprise Server Set-up Worksheet Instructions .....	31
<b>Operations.....</b>	<b>33</b>
Starting the Servers .....	33
Monitoring and Control .....	33
Dynamic Configuration .....	34
<b>Chapter 3 Server Setup for IMS Access .....</b>	<b>35</b>
<b>Introduction to Server Setup for IMS Access.....</b>	<b>35</b>
<b>DRA Support.....</b>	<b>36</b>
DBCTL .....	36
DRA .....	36
<b>Setting Up the DRA for Use by eXadas.....</b>	<b>37</b>
<b>Configuration.....</b>	<b>38</b>
<b>BMP/DBB Support.....</b>	<b>40</b>
Configuration .....	40
<b>Chapter 4 Server Setup for IDMS Access .....</b>	<b>43</b>
<b>Introduction to Server Setup for IDMS Data Access.....</b>	<b>43</b>
<b>APF Authorization of the IDMS.LOADLIB .....</b>	<b>44</b>
<b>Setting up Security for IDMS Access .....</b>	<b>45</b>
User ID/Password Validation .....	45
Passing the Correct User Context To IDMS In Run-Units.....	45
<b>Setting up a Server to Access an IDMS Central Version.....</b>	<b>46</b>
<b>Mapping IDMS Data for SQL Access .....</b>	<b>46</b>
Running the Meta Data Utility (METAU) with IDMS Meta Data Grammar.....	47
<b>How IDMS Paths Are Converted Into SQL Rows .....</b>	<b>48</b>
<b>Accessing Multiple Databases in an IDMS Central Version .....</b>	<b>48</b>
<b>Accessing Multiple IDMS Central Versions from a Single Server .....</b>	<b>49</b>
<b>Chapter 5 Server Setup for CA-DATACOM/DB .....</b>	<b>51</b>
<b>Introduction to Server Setup for CA-DATACOM/DB .....</b>	<b>51</b>
<b>Define the Datacom Initialization Service to the Server.....</b>	<b>52</b>
<b>Ensure the Multi-User Facility Is Running Authorized .....</b>	<b>52</b>

	<b>CA-DATACOM/DB Data Access .....</b>	<b>53</b>
	<b>Setting Up CA-DATACOM/DB Security .....</b>	<b>54</b>
<b>Chapter 6</b>	<b>Communication Configuration.....</b>	<b>55</b>
	<b>Introduction to Communication Configuration .....</b>	<b>55</b>
	<b>Communications Options .....</b>	<b>56</b>
	Cross Memory .....	57
	IBM MQ Series .....	57
	Conceptual Overview .....	58
	Prerequisites to Using MQ Series.....	62
	OS/390 Queue Manager Definitions.....	62
	TCP/IP.....	63
	<b>Selecting a Communications Option .....</b>	<b>64</b>
	Bandwidth.....	64
	TCP/IP Use Of Hostnames vs. IP Addresses .....	64
	<b>Server Configuration .....</b>	<b>65</b>
	Cross Memory.....	65
	IBM MQ Series .....	66
	TCP/IP.....	67
<b>Chapter 7</b>	<b>SQL Security .....</b>	<b>69</b>
	<b>Introduction to SQL Security.....</b>	<b>69</b>
	<b>eXadas Security Concepts.....</b>	<b>70</b>
	<b>User Types .....</b>	<b>70</b>
	<b>Database Objects .....</b>	<b>71</b>
	<b>Defining User Privileges.....</b>	<b>72</b>
	System Privileges .....	72
	Database Privileges .....	74
	Stored Procedures Privileges .....	76
	Table and View Privileges .....	78
	<b>Authorization Requirements .....</b>	<b>81</b>
	<b>SQL Security and the eXadas SAF Exit.....</b>	<b>82</b>
	<b>Summary .....</b>	<b>83</b>
<b>Chapter 8</b>	<b>Mapping Data.....</b>	<b>85</b>
	<b>Introduction to Mapping Data .....</b>	<b>85</b>
	The Data Mapping Process.....	86
	<b>DataMapper .....</b>	<b>87</b>
	General Data Mapping .....	87
	<b>Meta Data Utility .....</b>	<b>88</b>

<b>Advanced Mapping Considerations .....</b>	<b>89</b>
Defining Indexes .....	89
Multi-Part Keys .....	89
VSAM Indexes .....	90
IMS Indexes .....	90
IDMS Indexes .....	91
Occurs Processing .....	92
Record Arrays .....	92
Multiple Record Arrays in a Single Database Record .....	94
Record Typing .....	95
IMS Segment Mapping Considerations .....	96
<b>Chapter 9 Optimization.....</b>	<b>97</b>
<b>Introduction to Optimization .....</b>	<b>97</b>
<b>Query Optimization .....</b>	<b>98</b>
Using Keys .....	98
<b>JOIN Optimization.....</b>	<b>99</b>
<b>Query Processor Optimization.....</b>	<b>101</b>
Immediate Return of Data .....	102
Static Catalogs .....	102
Result Set Staging .....	103
<b>IMS Data Access Optimization .....</b>	<b>103</b>
General Guidelines.....	103
IMS Native Access .....	103
Using Primary Indexes .....	104
Using Secondary Indexes .....	104
Defining IMS Indexes With the USE INDEX Statement.....	104
Using Search Fields .....	105
Partial Key Support .....	105
Path Calls .....	106
HDAM/HIDAM Access Considerations .....	106
DEDB Considerations .....	107
PCB Selection Options .....	108
PCB Selection by Verification .....	108
PCB Selection by Name .....	109
PSB Scheduling .....	109
Using the ASMTDL/I Interface .....	110
Using the DRA Interface .....	111
<b>VSAM Data Access Optimization.....</b>	<b>112</b>
General Guidelines.....	112
Primary and Alternate Indexes .....	113

Partial Key Support.....	114
VSAM Query Processor Optimizations.....	115
VSAM Service .....	115
<b>Server Execution.....</b>	<b>115</b>
Dispatching Priority .....	115
WLM Support .....	116
<b>Chapter 10 Server Operations .....</b>	<b>117</b>
<b>Introduction to Server Operations .....</b>	<b>117</b>
<b>Starting Servers.....</b>	<b>118</b>
<b>Monitoring and Controlling Servers .....</b>	<b>118</b>
Displaying Active Services in a Server.....	119
Displaying Users Connected to a Server.....	119
Displaying Configurations .....	120
Modifying Configuration Parameters .....	122
Displaying Memory Utilization .....	123
<b>Starting and Stopping Individual Services .....</b>	<b>124</b>
<b>Stopping the Server .....</b>	<b>125</b>
<b>Chapter 11 Views .....</b>	<b>127</b>
<b>Introduction to Views.....</b>	<b>127</b>
<b>What is a View? .....</b>	<b>128</b>
How the Query Processor Handles Views .....	129
Advantages and Disadvantages of Views .....	130
<b>How to Create a View.....</b>	<b>130</b>
Using Views for Record and Column Filtering .....	132
Horizontal Views .....	132
Vertical Views.....	133
Row/Column Subset Views .....	133
Grouped Views .....	133
Using Views for Security.....	134
<b>Joined Views.....</b>	<b>134</b>
<b>Dropping Views .....</b>	<b>135</b>
<b>Chapter 12 Server Logging .....</b>	<b>137</b>
<b>Introduction to Server Logging.....</b>	<b>137</b>
Controlling Logged Information .....	138
<b>The Log Print Utility.....</b>	<b>139</b>
<b>Log Print Filtering .....</b>	<b>140</b>

---

<b>Chapter 13 Utilities</b> .....	<b>141</b>
<b>Introduction to Utilities</b> .....	<b>141</b>
<b>Meta Data Utility</b> .....	<b>142</b>
Meta Data Grammar .....	144
Running the Meta Data Utility .....	145
<b>USE TABLE Statement Syntax</b> .....	<b>150</b>
Table Source Definitions .....	151
Column Definitions .....	161
Record Arrays .....	176
<b>USE [UNIQUE] INDEX Statement Syntax</b> .....	<b>180</b>
Defining VSAM Indexes .....	182
Defining IMS Indexes .....	182
<b>DROP TABLE Statement Syntax</b> .....	<b>182</b>
<b>DROP INDEX Statement Syntax</b> .....	<b>183</b>
<b>DB2 Grammar</b> .....	<b>184</b>
CONNECT TO DB2 Statement Syntax .....	185
IMPORT DB2 TABLE Statement Syntax .....	186
IMPORT DB2 INDEX Statement Syntax .....	188
<b>CICS VSAM Grammar</b> .....	<b>190</b>
<b>ADABAS USE Statement Generator</b> .....	<b>192</b>
<b>Chapter 14 Open Catalog</b> .....	<b>197</b>
<b>Introduction to Open Catalog</b> .....	<b>197</b>
Open Catalog Overview .....	198
<b>Objects Used to Define and Access a Table</b> .....	<b>202</b>
<b>What are Fragments?</b> .....	<b>204</b>
IMS Example .....	205
IDMS Example .....	206
<b>Record Arrays</b> .....	<b>208</b>
<b>Differences With Meta Data Tables</b> .....	<b>210</b>
VARCHAR Columns .....	211
Use of NULL IS Definitions .....	211
Use of the REMARKS Column .....	212
Predefined Table Names .....	213
Use of Indexes .....	213
Ability to Delete Meta Data Tables .....	214
<b>Installing Meta Table in the eXadas System Catalog</b> .....	<b>214</b>
<b>Chapter 15 System Exits</b> .....	<b>219</b>
<b>Introduction to System Exits</b> .....	<b>219</b>



<b>Security: SAF Exit Specifics .....</b>	<b>220</b>
Activating the SAF Exit.....	221
SAF Exit API Overview .....	224
SAF Exit Initialization .....	225
SAF Exit Validation .....	226
SAF Exit Termination.....	228
<b>Accounting: SMF Exit Specifics.....</b>	<b>228</b>
Activating the SMF Exit .....	228
SMF Exit API Overview.....	231
Initialization .....	233
Validation/Accounting .....	233
Authorization Violations.....	235
Termination.....	235
<b>CPU Resource Governor .....</b>	<b>235</b>
Activating the CPU Resource Governor Exit .....	236
CPU Resource Governor Exit API Overview .....	238
Initialization .....	240
Validation/Accounting .....	240
Termination.....	242
<b>Workload Manager Exit .....</b>	<b>242</b>
Activating the WLM Exit .....	243
WLM Exit API Overview .....	247
Initialization .....	248
Management/Reporting.....	249
TCB Initialization/Termination.....	249
User Connect/Disconnect .....	250
SQL Statement Processing.....	250
Termination.....	251
<b>DB2 Thread Management Exit .....</b>	<b>251</b>
Activating the DB2 Thread Management Exit .....	252
Developing Your Own DB2 Thread Management Exit.....	254
<b>Record Processing Exit .....</b>	<b>256</b>
Initialization .....	257
Process .....	258
Termination.....	258
Update .....	258
Verification .....	258
Performance Considerations .....	259

---

<b>Chapter 16 Stored Procedures.....</b>	<b>261</b>
<b>Introduction to Stored Procedures .....</b>	<b>261</b>
Stored Procedure Overview .....	262
General Concepts .....	262
Residency and Language Environment .....	264
Interfacing with CICS .....	267
Interfacing With IMS.....	269
Interfacing with CA-DATACOM/DB .....	271
Support Routines.....	272
Samples .....	273
<b>Defining Stored Procedures.....</b>	<b>274</b>
CREATE PROCEDURE Syntax and Description.....	275
DROP PROCEDURE Syntax and Description.....	281
Deactivating the LE Environment .....	282
Specifying CICS Transaction Scheduling Information .....	282
Specifying CA-DATACOM/DB Resource Information .....	283
<b>Writing Stored Procedures.....</b>	<b>284</b>
<b>Invoking Stored Procedures.....</b>	<b>292</b>
CALL Statement Syntax .....	293
ODBC Stored Procedure Support.....	295
<b>CICS Interface Description .....</b>	<b>297</b>
CACSPBR Interface Description.....	297
Parameters Passed to the CICS Application Program .....	303
Compiling and Linking Applications that Use CACSPBR .....	304
CACSPBR Return Codes.....	305
CACSP62 Abend Codes .....	306
<b>CA-DATACOM/DB Interface Description .....</b>	<b>310</b>
CACTDCOM Interface Description .....	311
Compiling and Linking Applications That Use CACTDCOM .....	315
CACTDCOM Return Codes .....	316
<b>IMS DRA Interface Description .....</b>	<b>320</b>
CACTDRA Interface Description.....	321
Compiling and Linking Applications That Use CACTDRA.....	323
CACTDRA Return Codes.....	324
<b>Invoking Existing IMS Transactions.....</b>	<b>325</b>
APPC/IMS Overview.....	326
APPC/MVS Overview .....	327
Configuring APPC/IMS and APPC/MVS .....	327

Application Design Requirements .....	327
Stored Procedure Limitations.....	328
Testing APPC/MVS Stored Procedures.....	329
Sample Stored Procedures .....	330
Adding Transaction Security .....	330
Sync-Point Conversations.....	331
<b>Support Routine Descriptions .....</b>	<b>331</b>
Get RUN OPTIONS (CACSPGRO) Calling Conventions.....	332
Get User ID (CACSPGUI) Calling Conventions .....	332
Get User Password (CACSPGPW) Calling Conventions.....	333
<b>Chapter 17 Enterprise Server .....</b>	<b>335</b>
<b>Introduction to Enterprise Server Installation and Configuration .....</b>	<b>335</b>
<b>Deployment of the Enterprise Server .....</b>	<b>336</b>
Deployment Steps .....	337
<b>Operations.....</b>	<b>339</b>
Starting OS/390 Enterprise Servers .....	339
Monitoring and Controlling Enterprise Servers.....	340
Displaying Active Services in an Enterprise Server .....	341
Displaying Servers Connected to an Enterprise Server .....	341
Displaying Configurations .....	342
Modifying Configuration Parameters .....	343
Displaying Memory Utilization .....	343
Starting and Stopping Individual Services.....	343
Stopping the Enterprise Server .....	344
<b>Integration and Configuration.....</b>	<b>345</b>
Base System Configuration Review .....	345
Parameter Correlation .....	345
Supported Protocols.....	346
Communication Value Formats .....	346
<b>Enterprise Server Integration .....</b>	<b>347</b>
The Mechanics of a Transparent Integration .....	347
<b>Data Source Handler Service Configuration .....</b>	<b>347</b>
<b>Dynamic eXadas Server Scheduling.....</b>	<b>348</b>
<b>Expanded Scheduling and Connection Balancing With Cross</b>	
<b>Memory .....</b>	<b>349</b>
Using IMS DBB Access in a DBRC FORCE=YES Environment .....	350
<b>Chapter 18 SQL Update .....</b>	<b>353</b>
<b>Introduction to SQL Update .....</b>	<b>353</b>
<b>Transactions.....</b>	<b>354</b>

---

<b>SQL Update Statements .....</b>	<b>355</b>
INSERT.....	355
UPDATE.....	356
DELETE .....	356
COMMIT .....	356
ROLLBACK.....	357
<b>SQL Update and Mapped Tables.....</b>	<b>357</b>
Mappings Containing Multiple Records.....	357
Insert Positioning .....	358
Data Defaulting in Database Records on INSERT .....	359
Update and Delete Behavior .....	359
Update and NULL Records .....	359
Mappings Containing Record Arrays .....	359
Group Items and Overlapping Fields.....	360
General Recommendations .....	360
<b>Adabas Update Considerations.....</b>	<b>360</b>
<b>CA-DATACOM/DB Update Considerations .....</b>	<b>360</b>
<b>DB2 Update Considerations .....</b>	<b>362</b>
<b>IDMS Update Considerations .....</b>	<b>362</b>
General IDMS Update Considerations .....	363
SQL INSERT Considerations .....	363
SQL DELETE Considerations.....	364
<b>IMS Update Considerations .....</b>	<b>364</b>
IMS PSB Considerations .....	365
Update and Non-Update SQL Requests in a Single Transaction .....	365
PCB Processing Options .....	365
<b>CICS VSAM Update Considerations .....</b>	<b>366</b>
Transport Protocol .....	366
Flow of Interactions .....	367
<b>Chapter 19 Using Field Procedures .....</b>	<b>369</b>
<b>Introduction to Using Field Procedures .....</b>	<b>369</b>
<b>Specifying a Field Procedure.....</b>	<b>370</b>
<b>When Exits are Taken.....</b>	<b>370</b>
<b>Execution Environment .....</b>	<b>371</b>
The Field Procedure Parameter List (FPPL).....	371
The Work Area .....	372
The Field Procedure Information Block (FPIB).....	372
Value Descriptors.....	373

<b>Field-Encoding (Function Code 0)</b> .....	<b>374</b>
<b>Field-Decoding (Function Code 4)</b> .....	<b>376</b>
<b>Sample Field Procedures</b> .....	<b>378</b>
Sample Field Procedure CACFP001.....	379
Sample Field Procedure CACFP999.....	379
<b>Appendix A Configuration Parameters</b> .....	<b>381</b>
<b>Introduction to Configuration Parameters</b> .....	<b>381</b>
<b>Configuration Parameter Format</b> .....	<b>382</b>
<b>Configuration Parameter Relationships</b> .....	<b>383</b>
<b>Configuration Parameter Descriptions</b> .....	<b>385</b>
BTREE BUFFERS.....	385
CPU Governor.....	385
DATASOURCE.....	386
Sample Address Field for TCP/IP Protocol with data source name, CACSAMP.....	386
Sample Address field for Cross Memory Protocol with data source name, CACSAMP .....	386
Sample Address Field for MQ Series Protocol with data source name, CACSAMP .....	387
DECODE BUFFER SIZE.....	387
DEFLOC .....	388
FETCH BUFFER SIZE .....	388
INTERLEAVE INTERVAL.....	389
JOIN MAX TABLES ANALYZED.....	390
LD TEMP SPACE .....	391
LOCALE.....	392
MAX ROWS EXAMINED .....	393
MAX ROWS EXCEEDED ACTION.....	393
MAX ROWS RETURNED .....	394
MESSAGE POOL SIZE .....	395
NL .....	395
NL CAT .....	395
PDQ.....	396
RESPONSE TIME OUT.....	397
SAF EXIT .....	397
SERVICE INFO ENTRY .....	398
SMF EXIT.....	402
STATEMENT RETENTION .....	402
STATIC CATALOGS .....	403

TASK PARAMETERS .....	404
TRACE LEVEL .....	405
USER CONFIG .....	405
USERID .....	406
USERPASSWORD .....	407
VSAM AMPARMS .....	407
WLM UOW .....	409
<b>Appendix B Sample Stored Procedure VTAM and CICS Definitions .....</b>	<b>411</b>
<b>Introduction to Sample Stored Procedure VTAM and CICS</b>	
<b>Definitions .....</b>	<b>411</b>
<b>VTAM Resource Definitions .....</b>	<b>412</b>
<b>CICS Resource Definitions .....</b>	<b>414</b>
<b>Appendix C MTO Command Reference .....</b>	<b>423</b>
<b>Introduction to MTO Commands .....</b>	<b>423</b>
MTO Facility .....	424
<b>Commands .....</b>	<b>424</b>
SET,NAME=name, ORD=number, VALUE=value .....	424
CANCEL,USER=userid .....	425
CANCEL,SESSIONID=sessionid .....	425
DISPLAY,QUERIES .....	426
CANCEL,QUERY=name,SESSIONID=sessionid .....	426
MODIFY,servicename,TRACELEVEL=number .....	426
MODIFY,servicename,TRIGGER START=number .....	427
MODIFY,servicename,TRIGGER STOP=number .....	427
MODIFY,servicename,OUTPUT=DISPLAY .....	427
MODIFY,servicename,OUTPUT=DEFAULT .....	428
MODIFY,servicename,FLUSH .....	428
FLUSH,NAME=name .....	428
DISPLAY, {SERVICES   USERS   CONFIG=name   CONFIGS	
MEMORY   ALL } .....	428
START,SERVICE=name .....	430
STOP, {TASKID=tasknumber   SERVICE=name   ALL} .....	430
<b>Appendix D Sample SERVICE INFO ENTRY Definitions .....</b>	<b>431</b>
<b>Introduction .....</b>	<b>431</b>
Region Controller and Logger .....	432
<b>Query Processor .....</b>	<b>432</b>
<b>Connection Handler .....</b>	<b>432</b>
Cross Memory Transport Layer .....	433

---

TCP/IP Transport Layer .....	433
MQ Series .....	433
<b>IMS Interface Initialization Services.....</b>	<b>433</b>
CACIMSIF .....	434
CACDRA .....	434
<b>DB/2 Access.....</b>	<b>434</b>
<b>Datacom Initialization Service .....</b>	<b>434</b>
<b>VSAM Service.....</b>	<b>435</b>
<b>Work Load Manager Initialization Service .....</b>	<b>435</b>
<b>Language Environment Initialization Service .....</b>	<b>436</b>
<b>Multiple Catalog Support .....</b>	<b>436</b>
<b>Appendix E Meta Table Definitions .....</b>	<b>439</b>
<b>Introduction .....</b>	<b>439</b>
<b>Index.....</b>	<b>559</b>





# Overview

## Introduction to eXadas

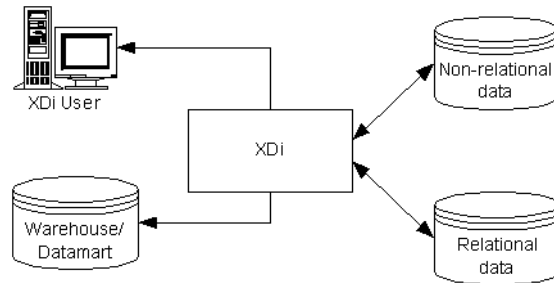
This chapter provides an overview of the eXadas™ eData Engine and includes the following topics:

- [“Product Overview,”](#) on page 2,
- [“Operational Components,”](#) on page 3,
- [“Application Components,”](#) on page 9, and
- [“Administrative Components,”](#) on page 9.

# Product Overview

eXadas is a powerful, efficient, and easy-to-implement eData solution. End-users and developers can transparently access distributed mission critical information using the desktop and Internet tools and applications of their choice. The following figure demonstrates how eXadas can be used to access relational and non-relational data.

**Figure 1: Accessing Data With eXadas**



eXadas is a complete, high-powered, eData solution that delivers:

- SQL access to relational and legacy data;
- a scalable, high-performance, easy-to-use product;
- a standards-based solution introducing no new application interfaces (APIs); and
- a modular solution that integrates easily with existing environments.

The eXadas eData Engine contains the following major components:

- Server,
- Enterprise Server,
- Client Interface Module,
- C Precompiler,
- Connectors (ODBC and JDBC), and
- DataMapper.

These components are grouped into three functional areas:

- Operational, which process requests for data and deliver the results to the client tool or application.
- Administrative, which configure components and manage system data.
- Application-enabling, which provide a 3GL hook into the eData Engine.

This chapter provides an overview and summary-level description of these components.

---

# Operational Components

Operational components provide the processing required to connect tools and applications with data. They are responsible for:

- accepting and validating SQL from the application or tool,
- communicating between the end-user and data source platforms,
- accessing the appropriate data source(s), and
- converting results into a consistent relational format.

The Operational components include:

- Server,
- Enterprise Server,
- Client Interface Module, and
- Connectors (ODBC and JDBC).

These modules are discussed in the sections that follow.

## eXadas Server

All data access is performed by platform-specific eXadas Servers. A Server is responsible for the following functions:

- Accepting SQL queries from clients.
- Determining the type of data to be accessed.
- Rewriting the SQL query into the native file or database access language needed.
- Query optimization based on generic SQL query rewrite and file or database specific optimization.
- Querying multiple data sources for JOINS.
- JOIN optimization based on index statistics held in the eXadas Meta Data Catalog.
- Translating result sets into a consistent relational format. For non-relational data sources this involves restructuring data into columns and rows. For relational data sources, result sets are translated into a single relational format.
- Post Query processing of result sets as needed. For example, ORDER BY sorting.

A Server accepts connection requests from client applications. Client applications can access a Server using either a CrossAccess-supplied Connector or applications developed using the C precompiler. Precompiler-developed applications can either reside on the same platform that the Server is executed on or on a remote platform.

There are five types of tasks that can run in the Server:

- Region Controller, which includes an MTO Operator Interface,
- Connection Handlers,
- Query Processors,
- Logger, and
- Initialization Services.

These tasks are described in the sections that follow.

## **Region Controller**

The Server has multiple tasks running within it. The main task is the Region Controller. The Region Controller is responsible for starting, stopping, and monitoring the other tasks running within the Server. The Region Controller determines which tasks to start based on configuration parameter settings. See [Appendix A, “Configuration Parameters,”](#) for more information on configuration parameters.

The Region Controller also supplies an OS/390 MTO (Master Terminal Operator) interface that can be used to monitor and control a Server address space.

## **Connection Handlers**

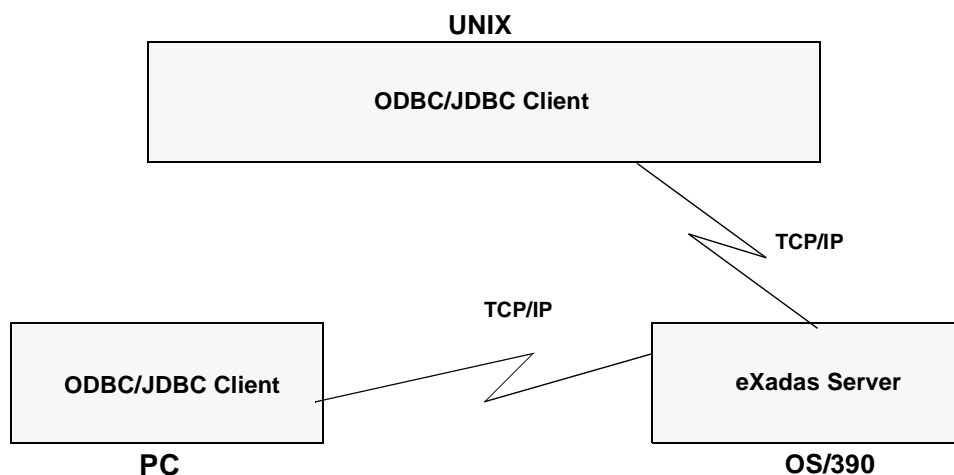
A Connection Handler (CH) task is responsible for listening for connection requests from client applications and routing them to the appropriate Query Processor task.

eXadas contains four typical transport layer modules that can be loaded by the Connection Handler task:

- TCP/IP,
- OS/390 Cross Memory Services, and
- MQ Series.

A local OS/390 client application can connect to a Server using any of these methods (the recommended approach is to use OS/390 Cross Memory Services). Remote client applications (running under Windows, a UNIX platform, or a different OS/390 image) use TCP/IP or MQ Series to communicate with a remote server.

Figure 2: Sample Communication Implementation

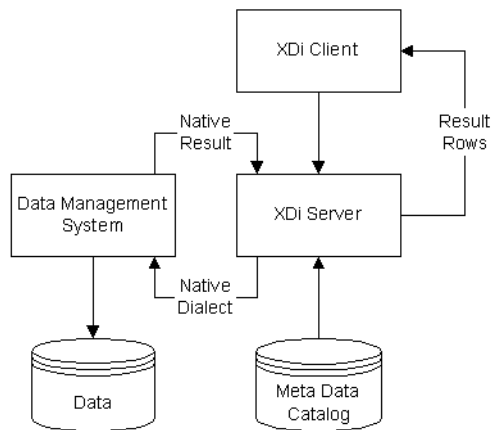


## Query Processor

The Query Processor is the subcomponent of the Server that is responsible for translating client SQL into database and file-specific data access requests. The Query Processor uses native database and file facilities that maintain both the structural integrity and the performance characteristics of the data source. [Figure 3: “Operational Processing Flow,” on page 6](#), shows the operational processing flow.

The Query Processor treats the different database/file systems as a single data source and is capable of processing SQL statements that access either a single type of database/file system or reference multiple types of databases/file systems. The Query Processor also supports SQL update operations (DELETE, INSERT, and UPDATE).

To process SQL data access requests, data definitions must be mapped to logical tables. The eXadas DataMapper tool is used in conjunction with the eXadas Meta Data Utility to perform this mapping. This information is stored in eXadas-generated Meta Data Catalogs, which emulate DB2 system catalogs. See [“DataMapper,” on page 9](#), for more information.

**Figure 3: Operational Processing Flow**

## Logger

A single Logger task can be running within a Server. The Logger reports on Server activities and is also used in error diagnosis situations.

## Initialization Services

Initialization Services are special tasks used to prepare the Server execution environment to:

- access IMS data,
- access DB2 data,
- access CA-DATACOM/DB data,
- initialize high-level language environments for use by exits,
- or allow the Server to use the OS/390 Work Load Manager (WLM) services to process queries in WLM goal mode.

Currently, the following Initialization Services are supplied:

- IMS BMP/DBB Initialization Service is used to initialize the IMS Region Controller to access IMS data using an ASMTDLI interface.
- IMS DRA Initialization Service is used to initialize the DRA interface and to connect to an IMS DBCTL region to access IMS data using the DRA interface.
- VSAM Initialization Service is used to initialize the Region Controller to access VSAM data.
- CAF Initialization Service is used to connect to a DB2 subsystem to access/update DB2 data using the DB2 Call Attach Facility.
- Datacom Initialization Service is used to initialize the Server for connections to the CA-DATACOM/DB Multi-User Facility (MUF) using the CA-DATACOM/DB DBNTRY interface for native command processing.

- WLM Initialization Service is used to initialize and register with the OS/390 Work Load Manager subsystem (using the WLM System Exit). This allows individual queries to be processed in WLM goal mode.
- The Language environment Initialization Service is used to initialize IBM's Language Environment or COBOL II, which allows exits to be written in a high-level language.

## Enterprise Server

The Enterprise Server is an optional component. The Enterprise Server can be used to manage a large number of concurrent users across multiple data sources. An Enterprise Server contains the same tasks that a Server uses, with the exception of the Query Processor and the Initialization Services.

Like a Server, the Enterprise Server's Connection Handler is responsible for listening for client connection requests. However, when a connection request is received, the Enterprise Server does not forward the request to a Query Processor task for processing, instead the connection request is forwarded to a Data Source Handler (DSH) and then to a Server for processing. The Enterprise Server maintains the end-to-end connection between the client application and the target Server. It is responsible for sending/receiving messages between the client application and the Server.

The Enterprise Server is also used to perform load balancing. Using configuration parameters, the Enterprise Server determines the locations of the Servers that it will be communicating with and whether those Servers are running on the same platform as the Enterprise Server.

The Enterprise Server can automatically start a local Server if there are no instances active. It can also start additional instances of a local Server when the currently active instances have reached the maximum number of concurrent users they can service, or the currently active instances are all busy.

## Client Interface Module

The Client Interface Module is eXadas-supplied code that is linked with a user written client program to establish and maintain connections with Servers and Enterprise Servers.

The Client Interface Module performs the following functions:

- Determining and loading the appropriate transport layer module, based on configuration.
- Establishing communications with a Server or Enterprise Server.
- De-referencing host variables in SQL statements.

- Storing and retrieving data in the application storage area(s).
- Presenting error and feedback information to the application.

The Client Interface Module can establish multiple connections to a Server or Enterprise Server(s) on behalf of a single application program.

## Client Connectors

Desktop tools and applications can issue SQL data access requests to a Server through an eXadas ODBC or JDBC client.

The ODBC and JDBC clients provide a single interface between end-user tools and applications and other eXadas operational components. Fast performance and application integrity are provided by the 32-bit thread safe ODBC and JDBC Connectors. A single client can access all data sources on all platforms.

The eXadas client serves as both an ODBC or JDBC Connector and a Connection Handler to other platforms, leveraging the underlying TCP/IP or MQ Series communications backbone.

Five software components interact to enable data access using eXadas:

- A platform-specific ODBC Driver Manager that loads Connectors on behalf of an application. This component is delivered with the operating system for all Windows platforms (for ODBC only).
- The ODBC and JDBC Connector that processes function calls, submits SQL requests to a specific data source, and returns results to the application.
- Data source definitions that consist of the name and location of the data the user wants to access. The required data source definitions consist of a data source name and communications parameters (TCP/IP or MQ Series). The data source name is used to identify a specific Server or Enterprise Server that will be used to service data access requests.
- The Client Interface Module that is used to bridge from the ODBC or JDBC client to the Query Processor task running in a Server.
- The Connection Handler that is used to communicate with a Server or Enterprise Server. eXadas supplies a Connection Handler that supports TCP/IP or MQ Series implementations.

For more information on the eXadas ODBC or JDBC Connectors, see the *eXadas Connectors Guide*.



---

# Application Components

Application-enabling components provide developers with a means of using eXadas' data delivery capabilities within 3GL applications. The eXadas C precompiler enables applications using embedded SQL to access data sources that span platforms using a single, standard API.

An application is written as if all data being accessed is in a single relational database. The eXadas precompilers convert SQL into calls to eXadas components. Multiple data sources that reside on local or remote platforms can be accessed by the application without regard to location or the type of file or database to be accessed.

The precompiler enables mainframe applications to access LAN-based data as easily as they enable UNIX-based applications to access mainframe data.

## Administrative Components

Administrative Components are tools and utilities used to perform the housekeeping and data administration required to define an installation's environment as well as the data to be accessed by eXadas. The eXadas DataMapper is one of these administrative components.

### DataMapper

The DataMapper is a Microsoft Windows-based application that automates many of the tasks required to create logical table definitions for non-relational data structures. The objective is to view a single file or portion of a file as one or more relational tables. The mapping must be accomplished while maintaining the structural integrity of the underlying database or file.

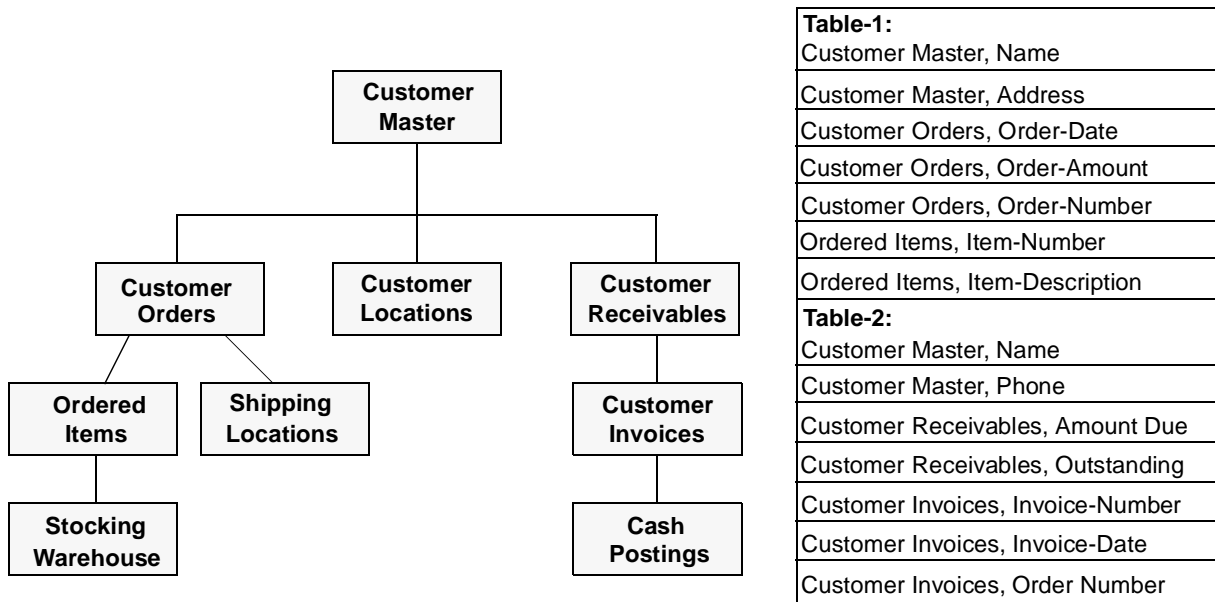
The DataMapper interprets existing physical data definitions that define both the content and the structure of the data. The tool is designed to minimize administrative work, using a definition-by-default approach.

The DataMapper accomplishes the creation of logical table definitions for the supported data structures by creating Meta Data Grammar from existing data definitions (COBOL copybooks). The Meta Data Grammar is used as input to the Meta Data Utility to create a Meta Data Catalog that defines how the data structure is mapped to an equivalent logical table. The Meta Data Catalogs are used by Query Processor tasks to facilitate both the access and translation of the data structure into relational result sets.

The DataMapper import utilities create initial logical tables from COBOL copybooks. A point-and-click environment is used to refine these initial logical tables to match site- and user-specific requirements. You can utilize the initial table definitions automatically created by DataMapper, or customize those definitions as needed.

A sample mapping of the Fields within the Segments of a hierarchical IMS database to two logical tables are shown in [Figure 4: “Sample Mapping from Hierarchical IMS DB to Logical Table.”](#)

**Figure 4: Sample Mapping from Hierarchical IMS DB to Logical Table**



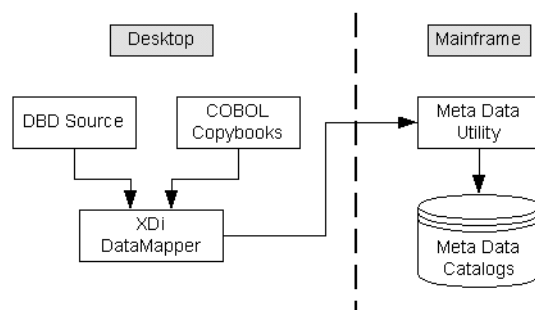
Multiple logical tables can be created that map to a single physical file or database.

For example, a site may choose to create multiple table definitions that all map to an employee VSAM file:

- One table is used by department managers who need access to information about the employees in their departments,
- another by HR managers who have access to all employee information,
- another by HR clerks who have access to information that is not considered confidential, and
- another by the employees themselves who can query information about their own benefits structure.

Customizing these table definitions to the needs of the user is not only beneficial to the end-user, but is recommended.

[Figure 5: “DataMapper Workflow,”](#) shows the Data Administration workflow with DataMapper.

**Figure 5: DataMapper Workflow**

**NOTE:** The DataMapper contains embedded FTP support to facilitate file transfer to and from the mainframe.

The steps in [Figure 5: “DataMapper Workflow,”](#) are:

1. Import existing descriptions of your non-relational data into DataMapper. COBOL copybooks, IMS-DL/I Database Definitions (DBDs), and IDMS schema/subschema into the DataMapper.

The DataMapper creates default logical table definitions from the COBOL copybook information. If these default table definitions are suitable for end-users, skip to Step 3.

2. Refine or customize the default table definitions as needed by the users. For example, importing the record layout for the VSAM customer master file creates the default `Customer_Table`. Two additional tables can also be created from the original:
  - `Marketing_Customer_Table` containing only those data items required by the marketing department.
  - `Service_Customer_Table` containing only those data items required by support representatives.
3. Export the logical table definitions to the mainframe where the database/file resides. These definitions are then used as input to the Meta Data Utility, which creates the Meta Data Catalogs.

After completing these steps, you are ready to use eXadas operational components with your tools and applications to access your data.

For step-by-step information on how to use the DataMapper to map data to logical tables, see the *eXadas DataMapper Guide*.



# 2

## Deploying Applications

### Introduction to Deploying Applications

This chapter begins by identifying the different departments within your organization that may need to be involved in order to quickly develop and deploy an eXadas-based application.

Developing your own applications is discussed from a Rapid Application Development (RAD) perspective. The goal is to assist you in determining which of the available eXadas interfaces to use and then to allow you to quickly develop your application using eXadas. The chapter also describes how to refine your application for optimum performance and how to get your eXadas-based application into production. The chapter concludes with a brief overview of the options for monitoring and controlling your application(s) for day-to-day operations.

This chapter contains the following topics:

- “[Departments and Responsibilities](#),” on page 14, discusses the departments and organizations that may be involved in developing and implementing an eXadas-based application.
- “[Application Models](#),” on page 16, describes the different types of applications that you are likely to develop using eXadas and provides recommendations about the different eXadas-supported application programming interfaces (APIs) that you can use for these applications.
- “[Initial Development](#),” on page 17, provides a set of guidelines that allows you to rapidly develop a proof-of-concept prototype of the eXadas-based application that meets your business needs.
- “[Deployment](#),” on page 26, provides guidelines for the steps required to put your eXadas-based applications into production for medium to large scale deployments on a single OS/390 system.
- “[Operations](#),” on page 33, includes an overview of the steps required to run your applications on a day-to-day basis and the features available to monitor, control, and troubleshoot your applications using eXadas.

## Departments and Responsibilities

The majority of this guide focuses on how to use eXadas to develop your own applications. In general, to successfully install, use, and implement eXadas-based applications, several resources may need to be involved including:

- developers,
- system programmers,
- security administrators,
- database administrators,
- network administrators, and
- operations staff.

Different terms may be used at your site to describe the various individuals (or group of individuals) in the preceding list. The term **developers** is used to refer to the individuals that are responsible for the coordination of the implementation of the eXadas system and site applications.

The following table describes the various departments and individuals involved in the process.

**Table 1: Departments and Responsibilities**

Departments/Individuals	Responsibilities
Developers	<ul style="list-style-type: none"> <li>• Verify the eXadas installation.</li> <li>• Learn the eXadas components.</li> <li>• Create Server(s) instance(s).</li> <li>• Define data sources.</li> <li>• Map data.</li> <li>• Identify remote connectivity requirements.</li> <li>• Configure the Server.</li> <li>• Develop applications that use eXadas.</li> <li>• Implement eXadas-based applications.</li> <li>• Tune eXadas-based applications.</li> </ul>
Systems Programmers	<ul style="list-style-type: none"> <li>• Install the OS/390 Server infrastructure.</li> <li>• Customize system exits to meet site requirements.</li> <li>• Provide tuning assistance for eXadas-based applications.</li> </ul>
Security Administrators	<ul style="list-style-type: none"> <li>• Provide authorization for Server components.</li> <li>• Provide authorization for file access by Server(s).</li> <li>• Provide authorization for IMS PSBs accessed by Server(s).</li> </ul>
IMS Database Administrators	<ul style="list-style-type: none"> <li>• Set up the IMS environment for use by eXadas.</li> <li>• Verify eXadas data access does not violate site standards.</li> <li>• Provide tuning assistance for eXadas-based applications accessing data.</li> </ul>
Network Administrators	<ul style="list-style-type: none"> <li>• Setting up the infrastructure for communication protocols.</li> <li>• Providing assistance in tuning eXadas-based applications.</li> </ul>
Operations	<ul style="list-style-type: none"> <li>• Monitor and control Servers.</li> <li>• Set Server dynamic configuration.</li> </ul>

**NOTE:** Depending on the eXadas components that you have purchased, not all of these individuals will need to be involved with the implementation of eXadas. Check with your CrossAccess sales representative or CrossAccess Technical Support for more information on the individuals required for the modules purchased at your site.

# Application Models

There are several different types of applications that you are likely to develop using eXadas. The type of application and the client platforms that you are planning to use determine the type of eXadas programming interface you can or should use. Each of these topics is discussed in more detail in the sections that follow.

## Types of Applications

eXadas is particularly well-suited for use in developing the following types of applications:

- Decision Support Systems (DSS),
- Client and Server Applications,
- Executive Information Systems (EIS),
- Web-based Applications,
- Data Warehouses,
- DataMarts, and
- Application Migration.

The platform on which your eXadas-based client application is deployed largely determines which eXadas programming interface you use. The available options are discussed in the section that follows.

## Selecting an Interface

There are several different application programming interfaces (APIs) that you can use to develop eXadas-based client applications. You have the following options for developing your own client applications using eXadas programming interfaces.

- ODBC,
- JDBC, and
- Precompiler.

Each of these options is discussed in the sections that follow.

### Connectors

If you are planning to use eXadas from a Graphical User Interface (GUI) running on Windows 95 or Windows NT, you must use the eXadas ODBC or JDBC



Connector to access a server. eXadas also supports ODBC or JDBC access on several UNIX platforms. If you are planning to deploy an eXadas client on UNIX, consult the *eXadas Connectors Guide* for the UNIX platform(s) that you plan to use to determine whether a Connector is available on that platform.

## Precompiler

A C precompiler is available for all platforms. If you are planning to create UNIX-, NT-, or OS/390-based client applications, then you need to use a precompiler-based application.

# Initial Development

CrossAccess recommends that you develop a proof-of-concept prototype of the application you are planning to create using eXadas. Very little of the development work required to create a proof-of-concept prototype will be lost when you start developing the real application. Most of the tasks discussed in the section involve merely setting up the infrastructure for the final application. The following topics are discussed:

- [“Creating Your Own Server,” on page 18](#), provides tips and guidelines for setting up your own Server environment(s) for use in developing your eXadas-based application.
- [“Mapping Your Data,” on page 19](#), provides tips and guidelines on how to map your data so that it can be accessed by eXadas.
- [“Mapping Verification,” on page 20](#), recommends steps that you should perform to ensure that the data has been mapped correctly. By verifying your mappings up front you can save yourself a lot of time and effort later.
- [“Creating Your Own Queries,” on page 22](#), provides tips and guidelines for prototyping or modeling your queries before you start developing your application. By prototyping/modeling your queries up-front you can ensure that you have mapped all of the data that you need to access. By prototyping/modeling your queries up-front you can also start optimizing your eXadas environment early in the development life cycle so that eXadas is ready to go into production when your application is complete.
- [“IMS Recommendations,” on page 24](#), provides tips and guidelines on how to access IMS data while you are developing your application.

## Creating Your Own Server

When developing your own applications, CrossAccess recommends that you create your own development Server for each application. If you are creating a group of related applications, then you should only create one Server to handle all of these applications.

You should clone one of the supplied Server jobs/procedures to create your new Server. For initial development, CrossAccess recommends using a batch job-oriented Server. Using a batch job allows you to have more control over starting/stopping the Server (than using a started task Server) and makes any troubleshooting easier. See [“Server Set-up Worksheet Instructions,” on page 30](#), for additional information.

**NOTE:** If you are planning to access IMS data with your new application, see the section [“IMS Recommendations,” on page 24](#), for additional tips on how to set up your development Server for IMS data access.

As part of Server set-up, make sure you configure an initial Query Processor data source and the appropriate communication services so that your client applications can communicate with the Server/Query Processor. For more information on configuring a data source in the Server, see the *eXadas OS/390 Getting Started Guide*. For more information on configuring Server communications interfaces see [Chapter 6, “Communication Configuration.”](#)

If you are going to develop client applications on the same OS/390 system, CrossAccess recommends that you use a Cross Memory services Communications Handler to communicate with the Server. When configuring your development Server, you will need to define a unique Cross Memory services data space name and queue name.

If you are going to develop remote client applications, CrossAccess recommends that you use a TCP/IP Connection Handler Service for initial development.

Before configuring a TCP/IP Connection Handler Service, contact your network administrator to obtain the hostname or IP address where you will be running the Server. Also ask your network administrator for a unique port number that is not being used by any other applications on that OS/390 system. The hostname/IP address and port number are required to configure your client application. For more information on TCP/IP Connection Handler set-up requirements, see [Chapter 6, “Communication Configuration.”](#)

If you are planning to use an MQ Series Connection Handler, contact your MQ Series administrator to have them create a Local Queue for use by your Server. You do not need to create another Model Queue since Servers can share Model Queues. Obtain the names of the Local and Model Queues that your Server should use. For more information on MQ Series communications see [Chapter 6, “Communication Configuration.”](#)

You may want to inform your network administrator about the communications required to deploy your application into production. CrossAccess recommends that you create separate development and production Server(s). You will continue

---

to use the development Server during the refinement process and can use the development Server when your application(s) are in production in troubleshooting situations.

When you discuss production deployment of your application, inform the network administrator of the communications protocol(s) you are planning to use for your production system, and the estimated number of users that your application is planning to support. As a rule-of-thumb, assume that a single Server can support about 50 concurrent users, unless you are planning on accessing IMS data using a BMP or DBB interface. In that situation assume that a Server can only support one user. Depending on the number of concurrent users your application will support, you can easily compute the number of TCP/IP ports or LUs that your production application requires.

After customizing your Server JCL and completing the configuration process, submit/start your new Server to verify that you have customized the JCL correctly and that you have not created any invalid configuration parameters. Once you have started your Server you can use the MTO Operator Interface to verify that the query processors and Connection Handlers that you have configured are operational. For more information on using the MTO Operator Interface, see [Chapter 10, “Server Operations.”](#)

Additionally, if you have configured a TCP/IP Connection Handler you can use the **TSO NETSTAT SOCKET** command to verify that the Connection Handler(s) are listening for connections on the proper hostname/IP address and port number.

Once you have created your development Server and verified that it is operational, you can move on to the next step of mapping your data.

## Mapping Your Data

The next step in creating your own eXadas-based application is to map your data into a format that eXadas can understand. For an overview of how to map your data, see [Chapter 8, “Mapping Data.”](#) For details on how to use the DataMapper, see the *eXadas DataMapper Guide*.

You have several options for mapping your data. If your application’s access requirements are well-defined, you can perform the mapping process in a single operation. If the access requirements are fluid, then mapping will be an iterative process as you identify new information to be accessed by your application.

You also need to decide whether to create a new set of Meta Data Catalogs. If your application uses unique logical table names and the data that you will be mapping is static, then you should use the same set of Meta Data Catalogs that were created as part of the eXadas installation, customization, and verification process. Using the same set of catalogs makes troubleshooting easier when CrossAccess Technical Support personnel are involved.

If your logical table names will not be unique across multiple production data servers and/or your project will be delivered in multiple phases, create a new set of Meta Data Catalogs. For more information about creating new Meta Data Catalogs see [Chapter 13, “Utilities.”](#)

**NOTE:** If you create a new set of Meta Data Catalogs, ensure that your application’s Server JCL is updated to reference the new Meta Data Catalogs after they have been created. Also ensure that the Server is stopped and then restarted before you attempt to access any of the new logical tables that you have mapped for your application. Failure to perform both of these steps results in query failure when you issue a query against the new logical tables, or, even worse, an incorrect/invalid result set may be returned. These situations can be very confusing when they occur. If either situation occurs, verify that the Server is set up properly.

The need to create multiple sets of Meta Data Catalogs can occur when your company has segregated the same types of files/databases based on size, organization, and/or department that you want to access using the same logical table name. In these situations, you have no choice but to create multiple sets of Meta Data Catalogs, each of which can either be referenced by a different Server, or reference multiple sets in a single Server. For additional information, see [Chapter 13, “Utilities.”](#) For your own development needs, you may need to only create a single Server and set of Meta Data Catalogs, unless the contents of the data varies by organization or department. In these situations, for development and testing purposes, you will also need to create multiple Servers.

Once you have completed mapping a logical table, or set of logical tables the next step is to verify that the data has been mapped properly. This process is discussed next.

## Mapping Verification

Once you have completed mapping a logical table or set of logical tables, verify that the data has been mapped properly by issuing a simple query against each logical table and reviewing the result set output to verify that the data looks correct.

You can either use the eXadas CACSAMP sample application to issue the verification query or, if you have purchased an eXadas Connector, you can use a query tool (like Microsoft Query, Microsoft Access, or Brio Query) to issue the verification query. If your tables contain a large number of columns, then using a query tool is highly recommended, as these tools do a much better job of formatting and presenting your data than the CACSAMP application.

The content of the data that you have mapped determines the type of verification query you need to create. If the data contains multiple formats based on one or more views, then this data is referred to as **partitioned data**. You must map a different logical table for each different record type that you plan to access.

If you have non-partitioned data then, for the verification query, you do not have to supply a WHERE clause. If your data is partitioned, you need to define a view on the table or the query specifying a WHERE clause that filters out the other types of records contained in the database/file. If you don't use a view, you also have to supply the same type of WHERE clause filtering when you create your own queries for the application you are developing.

For the verification query, retrieve a representative sample of your data (a hundred records or so). You can do this by using the configuration parameter-based governor features that eXadas supports. This requires setting the MAX ROWS EXAMINED configuration parameter to the number of records you want returned and specifying RETURN on the MAX ROWS EXCEEDED ACTION. Setting these parameters in such a fashion tells eXadas to only read the requested number of records/segments from the database/file and then return the result set. For more information on these configuration parameters, see [Appendix A, "Configuration Parameters."](#)

**NOTE:** If you set the MAX ROWS EXCEEDED ACTION to RETURN, remove this configuration parameter and/or set the action to ABORT when you are developing your own application and when you deploy the application into production. Failure to do so results in getting an invalid result set returned to your application. You will receive truncated data without any indication that not all of your data was returned.

Look at each column of the data returned from the verification query and verify that the data looks correct. If the data looks garbled, then the mapping was not performed properly. In this case, you may want to dump your physical data and compare it against what was returned from the verification query and compare the offsets and lengths reported in the DataMapper (for each column) against the physical data.

If there are discrepancies between the information in the DataMapper and the physical data, then your logical table needs to be re-mapped, the table dropped from the Meta Data Catalogs (using the DROP TABLE statement), and the table re-added using the Meta Data Utility. You then need to run the verification query again to see if this has corrected the problem(s). For more information on using the Meta Data Utility, see [Chapter 13, "Utilities."](#)

If the mapping offsets and lengths are correct, you need to verify that the SQL data type that you have assigned (to the column) is compatible with the physical data. If an incorrect SQL data type has been assigned, the SQL data type needs to be corrected, the logical table dropped, and the table re-added to the Meta Data Catalogs. You then need to run the verification query again to see if this has corrected the problem(s). For more information about eXadas SQL data type conversions see [Chapter 13, "Utilities."](#)

Once you have verified that the data is mapped correctly, you can start developing your own queries against the table(s) you have defined.

## Creating Your Own Queries

This section contains guidelines for developing queries for use in your own application. It includes some general considerations about how to create sample queries that your application will issue. It also discusses limiting the amount of data retrieved and performance monitoring.

### General Considerations

The programming interface you are using to develop your eXadas-based applications determines how you will develop your own queries. CrossAccess recommends that you prototype your queries.

If you are using a tool like MS Query, you can directly develop your queries using these types of front-end tools. If you are using a GUI-based tool like Visual Basic or Power Builder, or plan to use an ADK-based programming interface, model your queries using CACCSAMP. CACCSAMP allows you to supply a SQL SELECT statement with a maximum length of 16,384 bytes. This should be sufficient for most queries that you will develop for one of your own applications.

The reason to model/prototype your queries is to identify the different types of queries that your application uses and for each query to verify that the correct data is being retrieved, in the correct sequence. This modeling/prototyping should be done early in the development process before you develop your application logic that will be wrapped around the query. Once you have identified your queries, you can concurrently go on to develop the remainder of your application and also start refining your eXadas definitions to get the best performance for each query.

If you use CACCSAMP to model your queries, there are certain limitations. These primarily have to do with queries that use parameter markers to supply input to the query and in situations where your application uses inner/outer cursor loops. Recommendations on how to model each type of query follows.

### *Parameter Markers*

Parameter markers are often used in queries to supply run-time values in a WHERE clause. For example, suppose that your application displays a list of employee names and social security numbers. The user selects an employee to get further detail information about the employee. These are referred to as **drill down queries**. In a drill down situation the application would typically issue a query like:

```
SELECT EMPNO, LAST_NAME, FIRST_NAME, MI FROM EMPLOYEE_TABLE;
```

The result set returned from the above query would be displayed to the end user. When the user selected one of the names listed, the following type of query would be issued:

```
SELECT EMPNO, START_DATE, ... FROM EMPLOYEE_TABLE WHERE EMPNO = ?;
```

The parameter would be supplied on the SQL OPEN statement using a host variable reference that contains the employee number whose detail information is to be retrieved. CACCSAMP does not support the use of parameter markers. In this situation, model the second query by supplying a valid key value in place of the parameter marker. This simulates the way the second select statement will work in your application and allows you to verify that the correct detail data has been mapped and is being returned.

### ***Inner/Outer Cursor Loops***

For most applications, you can issue a join or union between two or more tables in order to obtain the information you need in a single SQL statement. However, some types of applications need to issue multiple queries simultaneously in order to obtain the data that they need. These types of queries are referred to as **inner/outer cursor loops**. In these situations, one query is issued that causes multiple rows of data to be retrieved. This query is referred to as the **outer query**. For each row retrieved, zero, one, or more queries are issued based on the data returned from the original query. These queries are referred to as the **inner query(s)**. Once an inner query completes, another row of data is retrieved from the outer query. This process continues until all result set rows have been retrieved in the outer query loop.

The CACCSAMP program also does not support the use of inner/outer cursor requests. You can model these queries like the drill down queries covered earlier. You would issue the outer query and then issue one or more inner queries where the WHERE clause has been modified to supply representative input values that have been retrieved by the outer query.

### **Limiting Query Output and Performance Monitoring**

When initially modeling/prototyping your queries, you should limit the amount of output returned for each query. You can use the same types of governors that you used to verify that the data was mapped correctly.

**NOTE:** Setting governor limits can be extremely important when your queries contain joins and/or unions. If done incorrectly these queries can generate Cartesian products from the tables being joined. If you are using governors, turn these governors off before performing pre-production testing of your application.

When you run your model/prototype queries you may want to monitor each query's performance. This can be as simple as running each query in a separate CACCSAMP run and recording the elapsed time each query takes, or if you are using an ODBC or JDBC interface recording the elapsed time from when you issued the query and the result set is returned (you will probably want to use a stop watch for this).

You can also monitor the Server using SDSF. You can record the CPU time used before and after the query is issued as well as the EXCP counts. This kind of monitoring is valuable only when you have a single user accessing the development Server.

Alternately, you can use the SMF or WLM System Exits to monitor your queries running in the Server. The SMF Exit collects CPU and elapsed (wall clock) times for individual users. To use the SMF Exits, you will need to connect and disconnect from the Server for each query in order to obtain accurate statistics for each of your queries. You will also need to execute the Query Processor tasks in single-user mode, where only one user is sharing an instance of a Query Processor TCB. Once you have collected SMF data, extract the eXadas-generated SMF records from the SMF data sets so that you can analyze the information collected. For more information on the SMF System Exits, see [Chapter 15, “System Exits.”](#)

The WLM Exit can also be used to obtain performance information. This information includes elapsed time, CPU time, service units, I/Os, and other system information. To obtain this information, the WLM system Exit must use a reporting service class. You will also need to execute the Query Processor tasks in single-user mode, where only one user is sharing an instance of a Query Processor TCB. You can use the RMF Monitor II/III reporting facilities to view the information captured by WLM for your queries. For more information about the WLM System Exit, see [Chapter 15, “System Exits.”](#)

If your queries are running too slowly, you may want to start the refinement process to improve query performance. See [“Deployment,” on page 26](#), for more information about the refinement process. This may be especially necessary if you are using an ODBC or JDBC front-end tool and you expect to demo the system to your end users. For additional information about queries, see [Chapter 9, “Optimization.”](#)

Another way to improve performance for queries that generate large result sets is to use the Hiperspace feature. This feature takes advantage of OS/390 expanded memory as an alternative to OS/390 temporary files. It allows fast access to data stored in temporary storage. On large machines, it is mapped over page data sets. Typical queries that benefit are those with GROUP BY and ORDER BY clauses and any large table query when PDQ is off.

**NOTE:** PDQ is automatically disabled when GROUP BY and/or ORDER BY clauses are present.

Hiperspace is enabled using the LD TEMP SPACE parameter. For more information on LD TEMP SPACE, see [Appendix A, “Configuration Parameters.”](#)

## IMS Recommendations

If you are accessing IMS data, during initial development, use a DBB interface to access your data. When using a DBB interface, an IMS Region Controller is loaded in the Server’s address space. The databases that you will be accessing are either explicitly allocated in the Server’s JCL, or if your IMS environment was set up properly, the database(s) to access can be dynamically allocated by IMS.

Using a DBB interface is easier to set up and modify (no IMS STAGE 1 generations are required). This allows you to easily modify the PSB that is used as



your access requirements change during the development process. Additionally, using a DBB interface does not have potential performance impacts on your other production IMS applications, which can occur when you use a BMP or DRA interface to access your IMS data. Using a DBB interface restricts the number of users that can concurrently use a Server based on the number of PCBs that have been defined in the PSB that the IMS Region Controller has loaded. The PSB name is specified in the Server's JCL.

If your application is accessing multiple databases, then you must manually create a composite PSB that contains PCBs for all of the databases and secondary index paths that your application will use. You may want to create a composite PSB that contains 10 PCBs for each database/secondary index you will be accessing, or you can modify the number of PCBs to fit your own needs.

You can compute the number of PCBs that are required using the following formula:

$$\text{Number of IMS tables referenced in an SQL query} \times \text{Number of concurrent users}$$

For example, if you have a query that joins 3 IMS tables together and you have 5 developers that will be using the Server for development purposes you will need 15 PCBs that have sensitivity to the database/secondary index paths for the 3 databases participating in the join. Typically, your application issues multiple different types of queries probably accessing several different databases so that actual number of PCBs will be larger than the 15 computed above. Another consideration is that the number of PCBs that are required is dependent upon the number of concurrent users issuing queries. It is unlikely that all 5 developers will be issuing the same types of queries simultaneously so that the actual number of PCBs that are needed may be less than the computed maximum. If you run out of PCBs, the person issuing a query that runs out of PCBs receives the message: `All PCBs in use` and the query is not processed.

As a general rule, if you create 10 PCBs per database/secondary index and give each PCB access to all paths in the database being referenced you should not run into any problems. If you follow this advice you will typically create a very large composite PSB. While using a large PSB in a DBB environment is acceptable for use in a development environment, this situation is not acceptable in a production environment. Use a DRA interface for the majority of your queries in a production environment. Converting your application from using a DBB interface to using a DRA interface is discussed in detail later in [“Deployment,” on page 26](#).

## Developing Your Application

If you have followed the development strategy outlined in the preceding sections then

- you have created the infrastructure for your eXadas-based application,
- you have a development Server in place and operational,

- you have defined (mapped) the data that your application will access, and
- you have prototyped/modeled the queries that your application will issue.

If you are developing an application using a front-end query tool like MS Access or Brio Query, your application is probably almost complete and you are ready to proceed to the refinement phase. If you are using an ODBC or JDBC programming tool like Visual Basic or Power Builder or developing an ADK-based application, you can start refining these applications since you can be reasonably sure that your access requirements have been defined and verified using the modeling/prototyping approach discussed earlier. In either case you are ready to move on to the refinement phase where you will tune your eXadas environment to get the best performance. The refinement phase should be performed in conjunction with the remainder of your application development work.

## Deployment

Once you have finished the refinement process and completed your other application development activities, you are ready to deploy your eXadas-based application into production. As part of completing your development activities you will probably want to run some full scale unit and system tests. When you run these tests make sure to remove any governor limits that you were using so that you get an accurate picture of how your application will perform in a production environment. Additionally, make sure you reset all trace levels to their default value of 4.

You should also consult with your operations staff and security administrators to determine which, if any, of the system exits require activation for your application. eXadas contains a set of sample exits that can be used for security authorization, SMF reporting, and WLM goal mode support. An exit is also supplied that allows you to limit the amount of CPU time that can be used to process queries. For more information about system exits, see [Chapter 15, “System Exits.”](#)

You have different options on how to configure and deploy your eXadas-based system into a production environment. The two variables that have the most impact are:

- the number of concurrent users you need to service and,
- (for IMS access) whether all users will be accessing data using the DRA interface or whether some will be using a BMP/DBB interface.

Rules for determining how many Servers you need to define and configure is discussed next, followed by the options that you have for deploying individual Servers.

## Determining How Many Servers Your Site Requires

The Server is designed to support 50 concurrent users accessing Sequential files, VSAM files, and/or IMS databases using the DRA interface in a single address space using 32 megabytes of memory. The Server may need more memory than this depending upon how many queries are being executed concurrently and whether these queries contain an ORDER BY clause or other predicates that requires sorting. However, for planning purposes you can assume that a single Server can handle 50 users.

The 50 user rule is not valid if you are using a DBB or BMP interface to access IMS data. As has been previously discussed, in these IMS environments a large composite PSB is required for all the databases that your application is accessing and a sufficient number of PCBs must be defined to service all queries that are active concurrently within the Server. Although CrossAccess recommends using a DBB interface with a fairly large PSB for development purposes, this practice is not advised in a production environment. Instead, CrossAccess recommends that if you are planning on using a DBB or BMP interface to access IMS data, that you set up those Servers to execute in single-user mode. This allows you to create a much smaller PSB and when using a BMP interface, have much less impact on your other IMS production applications.

Based on the above information you can calculate how many Servers you are going to need for your application. The formula is as follows:

$$(\text{Number of concurrent users} / 50 + 1) + \text{Number of concurrent IMS DBB/BMP users.}$$

**NOTE:** If you have created multiple sets of system catalogs, you will have to perform the above calculation for each set of system catalogs that exist in your production environment.

The next topic discusses the different options you have for deploying the number of Servers you need.

## Server Deployment Options

You can run your Servers as either started tasks or batch jobs. Your operations staff should be consulted as to which options should be used.

If you are deploying a large number of Servers, especially Servers that will be using a DBB/BMP interface to access IMS data, you probably want to also use the Enterprise Server to control your Servers. The Enterprise Server is capable of starting/stopping Servers (on the same OS/390 image) and performs automatic load balancing based on data source names. For more information about the Enterprise Server see the [Chapter 17, “Enterprise Server.”](#)

**NOTE:** If you are accessing IMS data using a DBB/BMP interface, it is highly recommended that you use an Enterprise Server. In a DBB/BMP situation without an Enterprise Server, you have to pre-start all of these types of Servers and assign them to individual users. If you attempt to share single user Servers between multiple client applications, then if the Server is already servicing one client when another attempts to connect to the Server, that user will receive an error stating that the Server is already in use.

## Creating Production Servers

Once you have determined how many Servers you need, what kind of Servers they will be (batch job or started task), and whether or not you are going to use an Enterprise Server, you are ready to start setting up and configuring these servers. Once this is completed, you can start configuring your client applications to communicate with these Servers and Enterprise Servers.

You can clone the production Server JCL from the one used for development or create brand new ones based on the sample JCL that was provided when eXadas was installed. You also need to create Master Configuration Members for each Server and the Enterprise Server. You may also have to increase service override configuration member(s) for one or more of the data sources that will be serviced in each Server.

CrossAccess has supplied the following two worksheets to assist you in setting up your Server(s) and Enterprise Server and for configuring your client application. The first worksheet can be used when you are only deploying Servers for your

application. The second worksheet should be used when you are going to use an Enterprise Server.

### Data Server Set-up Worksheet

Data Server Name:  Master Configuration Member Name:

Data Source Name	Protocol	Address	# Tasks/Users			Service Override
			Min.	Max.	Users	

### Enterprise Server Set-up Worksheet

Enterprise Server Name:  Master Configuration Member Name:

Data Source Name	Prot.	Address/ # Tasks/Users	DS Name/T	XM Name/ # Tasks/Users	MCM	Override

## Server Set-up Worksheet Instructions

You need to create a separate Server Set-up Worksheet for each of the Servers that you are setting up. Enter the name of the Server batch job name or started task PROC name in the Server Name field. Enter the name of the Master Configuration Member that Server will use in the Master Configuration Member Name field.

**Then, for each data source that you have defined:**

1. Fill in the data source name in the Data Source Name field. In the Master Configuration Member, you have to create a Query Processor SERVICE INFO ENTRY definition for each Data Source. For more information on the Query Processor SERVICE INFO ENTRY parameter see [Appendix A, “Configuration Parameters.”](#)
2. Identify the type of communications protocol that will be used to communicate with the Server in the Protocol field. These will either be TCP (for TCP/IP) for remote clients, or XM1 for OS/390 local clients.
3. Enter the TCP/IP hostname/IP address and port number/port name, or XM Data Space Name and Queue name in the Address field depending upon the type of protocol you entered in the Protocol field.
4. Identify the number of Query Processor Tasks and users that will be serviced by the Query Processor. The Min field should specify how many Query Processor Tasks are to be started during Server initialization. The Max field should specify the maximum number of query Processor Tasks that can be started. The Users field should specify the number of users to be serviced by each Query Processor Task. These values should be specified on fields 4 through 6 of the Query Processor SERVICE INFO ENTRY definition. For more information on the Query Processor SERVICE INFO ENTRY parameter see [Appendix A, “Configuration Parameters.”](#)
5. To use different configuration parameters for the data source, enter the name of the Service Override member that will contain these override parameters. This member name must be specified on field ten (task data field) in the Query Processors SERVICE INFO ENTRY definition.

For each unique Protocol/Address field combination you have to define the appropriate SERVICE INFO ENTRY definition in the Master Configuration Member. Although not shown on the worksheet you need to specify the minimum number of threads, maximum number of threads, and maximum number of users that will be serviced by each communications interface. For more information on defining Connection Handler Services SERVICE INFO ENTRY definitions see [Appendix A, “Configuration Parameters.”](#)

If your Server is accessing IMS data you need to create either a DBB/BMP Initialization SERVICE INFO ENTRY definition or a DRA Initialization SERVICE INFO ENTRY definition in order to be able to access your IMS data. For more information on defining IMS initialization SERVICE INFO ENTRY definitions see [Appendix A, “Configuration Parameters.”](#)

Once you have created the Server's JCL, the Master Configuration Member, and any Service Override members, you can start the Server. You can then start configuring your client applications.

If you are using Preprocessor-based clients, create a DATASOURCE entry in the client configuration file for each data source that the client application will be accessing. On the DATASOURCE definition specify the Data Source Name from the form and fill in the appropriate Protocol type and protocol specific Address information.

Refer to the appropriate client manual on how to specify DATASOURCE definitions for the platform that the client will run on.

For ODBC or JDBC clients, create a data source definition for each Data Source Name (from the form) that the workstation that you are configuring communicates with.

Define the data source name from the worksheet form, select the appropriate protocol (TCP/IP or MQ Series) and fill in the information for the Address field.

For more information on configuring ODBC or JDBC data sources, see the *eXadas Connectors Guide*.

Once these steps have been performed, the client is ready to communicate with the Server and use your new application.

## eXadas Enterprise Server Set-up Worksheet Instructions

When using an Enterprise Server, you need to fill out one Enterprise Server Set-up Worksheet per Enterprise Server you will use. Enter the name of the Enterprise Server batch job or stated task PROC name in the Enterprise Server Name field. Enter the name of the Enterprise Server's Master Configuration Member in the Master Configuration Member Name field.

Then, for each of the data sources that you have defined:

1. Enter the name of the data source in the Data Source Name field. In the Enterprise Server's Master Configuration Member you have to create a DSH SERVICE INFO ENTRY definition. Additionally, you will have to create a Query Processor SERVICE INFO ENTRY definition in the Server(s) Master Configuration Member with the same Data Source Name. See [Chapter 17, "Enterprise Server,"](#) for more information on how to define DSH SERVICE INFO ENTRY definitions and [Appendix A, "Configuration Parameters,"](#) for information on how to define Query Processor SERVICE INFO ENTRY definitions.
2. In the Prot. field, identify the communications protocol that client applications will be using to communicate with the Enterprise Server. The protocol will either be TCP (for TCP/IP) or XM1 for local OS/390 client applications.

3. Enter the TCP/IP hostname/IP address and port number/port name, or XM Data Space Name and Queue name in the Address field, depending upon the type of protocol you entered in the Protocol field.
4. Under the Address information, specify the Minimum/Maximum Number of Tasks and the number of Users that must be serviced for that data source by the DSH. The Minimum Number of Tasks identifies the number of DSHs to be started during Enterprise Server initialization. The Maximum Number of Tasks identifies how many users are serviced by each DSH task. This information must be specified in fields 4 - 6 on the DSH SERVICE INFO ENTRY definition.
5. Enter the name of the Server batch job or started task in the DS Name/T field that the DSH entry will be managing. Also identify the type of Server that the DSH will be managing. If the Servers will be run as batch jobs, the type should be 1; for started tasks, it should be 2.
6. Identify the Cross Memory Data Space Name and Queue Name that the DSH will use to communicate with the Server.
7. Under the SM Name information, specify the Minimum/Maximum Number of Tasks and Number of Users that will be serviced by the Server for that data source. The Minimum Number of Tasks identifies how many Query Processor tasks will be started during Server initialization. The Maximum Number of Tasks identifies how many Query Processors the Server is allowed to start. The Number of Users identifies how many users are serviced by each Query Processor Task. The numbers entered here should be specified in fields 4 through 6 on the Query Processor SERVICE INFO ENTRY definition. Additionally, the Number of Users values must be specified in the Task Data field for the associated DSH SERVICE INFO ENTRY definition. The Number of Users specifies load balancing information the DSH uses to determine when to start new Server instances.
8. Identify the name of the Master Configuration Member that the Server identified in the DS Name field will use.
9. If the Data Source Name is using a Service Override Member to supply custom tuning parameters, identify the name of the Service Override Member in the Override field. This member name must be specified on the Query Processor SERVICE INFO ENTRY definition (field 10) in the Server's Master Configuration Member.

For each unique Prot./Address field combination, you will have to define the appropriate SERVICE INFO ENTRY definition in the Enterprise Server's Master Configuration Member. Although not shown on the worksheet, you must specify the minimum number of threads, maximum number of threads, and maximum number of users that will be serviced by each Connection Handler. For more information on defining Connection Handler Services SERVICE INFO ENTRY definitions, see [Appendix A, "Configuration Parameters."](#)

You then need to create the Enterprise Server's JCL and Master Configuration Member. Then, for each of the Servers that you have identified, create their execution JCL, Master Configuration Member, and any Service Override Member(s). Additionally, if a Server is accessing IMS data create either a



DBB/BMP Initialization SERVICE INFO ENTRY definition or a DRA Initialization SERVICE INFO ENTRY definition in order to access your IMS data. See [Appendix A, “Configuration Parameters,”](#) for more information on defining Query Processor SERVICE INFO ENTRY definitions and for information on how to define IMS initialization SERVICE INFO ENTRY definitions.

Once the above steps have been completed you can start the Enterprise Server. For client configuration, follow the same instructions that were described for the Server set-up process.

## Operations

This section provides an overview of how to run your Server/Enterprise Servers in a production environment. Topics covered include:

- [“Starting the Servers,”](#) on page 33,
- [“Monitoring and Control,”](#) on page 33, and
- [“Dynamic Configuration,”](#) on page 34.

You can find more information about these activities in [Chapter 10, “Server Operations.”](#) For Enterprise Server operations see [Chapter 17, “Enterprise Server.”](#)

### Starting the Servers

If you are not using an Enterprise Server, then each of your production Servers must be pre-started before your client applications can be used. If you are using an Enterprise Server, then only the Enterprise Server must be pre-started. The Enterprise Server automatically starts Servers based on the minimum number of threads information specified on the DSH SERVICE INFO ENTRY definitions in the Enterprise Servers Master Configuration Member and as the number of concurrent users increases.

### Monitoring and Control

The Servers and the Enterprise Server are designed to run continuously. In the current version, eXadas supplies an OS/390 MTO (Master Terminal Operator) interface that can be used to monitor and control Server/Enterprise Server operations. Using the MTO interface, you can issue commands to display the

different services that are active within a Server/Enterprise Server, the number of users that are currently using a Server/Enterprise Server, and the amount of memory that is available. You can also issue commands to start and stop services.

In the current version of eXadas, individual MTO commands must be issued against each Server or Enterprise Server that is executing. For more information on the monitoring and control commands that are available see [Appendix C, “MTO Command Reference.”](#)

## Dynamic Configuration

With eXadas, you can dynamically modify the configuration parameters that a Server/Enterprise Server is using with the MTO interface. Using these commands, you can add new SERVICE INFO ENTRY definitions, and then with the monitoring and control commands, activate these services.

You can also display the individual tuning configuration parameter definitions that are active in the Master Configuration Member and any/all Service Override Members that are currently in use. Using the MTO interface, individual configuration parameter settings can be modified. Some of these settings take effect immediately, while others take effect when the next query is processed for a particular user, or the next time a service is started.

When you modify a configuration parameter definition, the modified value is only stored in core and will be lost when the Server/Enterprise Server is stopped. The MTO interface also allows you to permanently save configuration member updates to disk. This should only be performed after your production system is stable since all comments (including commented-out configuration parameter definitions) are lost when a configuration member is dynamically saved to disk. For more information on the dynamic configuration commands that are available see [Appendix C, “MTO Command Reference.”](#)

# Server Setup for IMS Access

## Introduction to Server Setup for IMS Access

This chapter discusses the options available for accessing IMS data, the set-up requirements for each type of access technique, and the configuration steps that are required depending upon the access technique that you select. The following topics are discussed:

- [“DRA Support,” on page 36,](#)
- [“Setting Up the DRA for Use by eXadas,” on page 37,](#)
- [“Configuration,” on page 38,](#) and
- [“BMP/DBB Support,” on page 40.](#)

You can access IMS data using either the eXadas DRA interface, which is similar to the one used by CICS, or its standard ASMTDLI programming interface. When using a DRA interface, you must configure and activate an IMS DRA Initialization Service in order to access your IMS data. When using the ASMTDLI interface you can access IMS data in a BMP or DBB environment. In these environments you must configure and activate an IMS BMP/DBB

Initialization Service to access your IMS data. For both interfaces, the Server JCL requires alteration.

A description of the setup requirements and the steps required to configure each of the IMS access methods supported in eXadas is described in the sections that follow.

## DRA Support

This overview section discusses:

- the IMS facility DBCTL (Database Control),
- the DRA (Database Resource Adapter), which is the interface between the Server and DBCTL.

Following this section, additional setup requirements for accessing IMS data using the DRA interface are discussed as well as the configuration steps required for the Server to access IMS data using DRA.

### DBCTL

DBCTL is an IMS database manager subsystem that runs in its own address space. DBCTL must be configured and running to use the eXadas IMS DRA interface. For more information on DBCTL, see the IBM IMS documentation.

### DRA

The DRA is the interface between the Server and DBCTL. The DRA start-up/router program is supplied with the IMS product and executes within the Server Address Space. In IMS terms, this address space is known as a **coordinator controller (CCTL)**.

The functions of the DRA include:

- Establishing contact with the DBCTL address space and loading the DRA start-up parameter table. The DRA start-up parameter table provides the parameters needed to define this interface to a DBCTL subsystem.
- Requesting connection to, and disconnection from, DBCTL.

- Managing threads to DBCTL. The minimum and maximum threads defined to DBCTL are specified in the DRA start-up table. The maximum threads specified determines the maximum number of concurrent IMS PSBs that can be scheduled by the Server.
- Informing the Server when a shutdown of DBCTL has been requested or if DBCTL has failed.

## Setting Up the DRA for Use by eXadas

### To enable the DRA for use by eXadas:

1. Make the DRA start-up/router (load module DFSPRRC0) accessible to the Server by either:
  - a. copying DFSPRRC0 from the IMS.RESLIB library (built by the IMS generation process) into the SCACLOAD load library, or
  - b. by concatenating the IMS.RESLIB library to the SCACLOAD STEPLIB.
2. Make the DRA start-up table (load module DFSPZPxx) accessible to the Server.

The SERVICE INFO ENTRY parameter for DRA specifies the suffix xx for the start-up table name.

The default load module, DFSPZP00, is in the IMS.RESLIB library. For an example of DFSPZP00, see the *IBM IMS/ESA Installation Volume 2: System Definition and Tailoring*.

DFSPZP00 contains default values for the DRA initialization parameters. The remainder of the DRA modules reside in a load library that is dynamically allocated by DFSPRRC0 (the start-up/router). The default DDNAME and DSNAME of this load library (IMS.RESLIB) are specified in the default start-up table DFSPZP00.

### To specify values other than the defaults:

1. Code a new start-up table, naming it, for example, DFSPZP01. You may want to use the supplied default module, DFSPZP00, as an example. For a detailed description of each DRA start-up table parameter, see the *IBM IMS/ESA Customization Guide*.
2. Specify the desired values.
3. Copy any unchanged values from the default table.
4. Assemble and link the new module into a load library accessible to eXadas.

If the new module was named DFSPZP01, the IMS DRA Initialization, SERVICE INFO ENTRY parameter would then specify a value of 01 in the DRA Start-up Table Suffix field.

**NOTE:** For discussions on performance tuning issues related to database management that may be controlled using the DRA start-up table, see Topic 4.5 “Database Management” in the *IBM CICS/ESA V4R1 Performance Guide*.

## Configuration

This section of the document discusses how to configure the Server to access IMS data using the DRA interface. You must configure the IMS DRA SERVICE INFO ENTRY parameter in the Server before you can access IMS data. Only one IMS SERVICE INFO ENTRY parameter can be active in the Server.

**NOTE:** All sample JCL, configuration files, and sample application programs referenced in this chapter can be found in the SCACSAMP and SCACCONF libraries included with your eXadas software shipment.

**The following steps identify how to configure the Server for DRA access.**

1. Edit the Server configuration member (SCACCONF member CACDSCF).

The sample configuration member supplied contains default parameter values for the key configuration parameters that are required to execute a Server. The sample configuration member also contains a set of SERVICE INFO ENTRYs that are commented out. In this section we will be activating the DRA Initialization Service, which allows you to access IMS data using a DRA interface.

2. Activate the IMS DRA Initialization Service.

Uncomment the SERVICE INFO ENTRY for the IMS DRA Initialization Service Task. This SERVICE INFO ENTRY can be identified by the comments in the configuration member.

In the Task Data field of the SERVICE INFO ENTRY parameter, you need to specify additional information that is used to initialize the DRA interface. The following information must be supplied:

- DRA Start-up Table Suffix: Modify SF to specify the last two characters of the load module name created in DRA Setup, Step 2. If you are using the default DRA start-up table load module, specify 00.
- DRA user ID: Modify DRAUSER to specify the default DRA user ID that is used to connect to and register with DBCTL. The DRAUSER is the name by which the Server is known to DBCTL.
- Default PSB Name: Modify DEFPSB to specify the name of a PSB to be used when an IMS table is referenced whose Meta Data Grammar contains no PSB name. For more information about PSB scheduling when using a DRA interface, see [Chapter 8, “Mapping Data,”](#) and [Chapter 9, “Optimization.”](#)

3. Save the configuration member.

The following steps describe how to customize the Server JCL that can be used to access IMS data using the DRA interface.

**To customize the JCL to run the Server as a batch job:**

1. Edit the IMS DRA Server JCL stream (SCACSAMP member CACDS).

This member executes the Server as a batch job that is capable of accessing IMS data using the DRA interface.

2. Supply a valid job card.
3. Modify the JCL to conform to site specifications and specify the following parameters at the beginning of the in-stream procedure:
  - eXadas high-level qualifier (CAC).
  - SYSOUT class (SOUT).
4. Supply additional IMS information.

The high level qualifier for IMS data sets (IMS) must be supplied.

5. Save the changes.

You are now ready to run the Server. Before submitting the JCL for execution, ensure that the operational environment has been properly set up by performing the following steps.

1. Ensure that the SCACLOAD library has been APF-authorized.

If the SCACLOAD library is not APF-authorized you will receive a S047 abend when you attempt to run the Server.

**NOTE:** IMS libraries concatenated to SCACLOAD must also be APF-authorized.

2. Ensure that the Server has access and execute authority for the data sets referenced in the Server JCL.

Contact your security administrator and verify that the user ID the Server will run with has been granted access and execute authority for the data sets referenced in the Server JCL.

3. Submit the Server for execution.

If all of the steps have been performed properly, the Server starts up.

To verify that the Server is operational, select the job (while it is executing). You should see the following message near the top of the listing:

```
CAC00103I EXADAS SERVER: V2.2.4 READY
```

To shut down the Server you can use the OS/390 MTO interface and issue a **STOP** command. If you do not have MTO authority simply cancel the Server job.

## BMP/DBB Support

To use the BMP/DBB interface you must perform the following set-up tasks before attempting to access IMS data:

- configure the BMP/DBB initialization service and
- modify the BMP/DBB Server JCL.

The steps are described in the sections that follow.

## Configuration

This section discusses how to configure the Server to access IMS data using either a DBB or BMP interface. You must configure the IMS BMP/DBB Initialization Service in the Server before you can access data. Only one IMS SERVICE INFO ENTRY parameter can be active in the Server.

### To configure the Server for BMP/DBB access:

1. Edit the Server configuration member (SCACCONF member CACDSCF).

The sample configuration member supplied contains default parameter values for the key configuration parameters that are required to execute a Server. The sample configuration member also contains a set of SERVICE INFO ENTRYs that are commented out. In this section we will be activating the BMP/DBB Initialization Service, which allows you to access IMS data.

2. Activate the IMS BMP/DBB Initialization Service.



Uncomment the SERVICE INFO ENTRY for the IMS BMP/DBB Initialization Service Task. This SERVICE INFO ENTRY can be identified by the comments in the configuration member.

3. Save the configuration member.

The following two steps describe how to access IMS data using an IMS/BMP and DBB interface.

4. Modify the JCL for BMP Access.

This step applies to IMS BMP JCL only. For instructions on modifying the JCL for DBB access, skip to [step 5](#).

- a. Edit the IMS BMP Server JCL stream (SCACSAMP member CACBMP).

This member executes the Server as a batch job that is capable of accessing IMS data using a BMP interface.

- b. Supply a valid job card.

- c. Modify the JCL to conform to site specifications and specify the following parameters at the beginning of the in-stream procedure:

- eXadas high-level qualifier (CAC).
- SYSOUT class (SOUT).

- d. Supply additional IMS information.

1. The high level qualifier for IMS data sets (IMS) must be supplied.
2. A PSB name.

**NOTE:** Indicate an Application Group Name (AGN) if your site requires it. Otherwise the server will fail with a U0437 abend.

3. Save the changes.

5. Edit the IMS DBB Server JCL stream (SCACSAMP member CACDBB).

This step applies to IMS DBB JCL only. For instructions on modifying the JCL for BMP access, see [step 4](#).

This member executes the Server as a batch job that is capable of accessing IMS data using a DBB interface.

- a. Supply a valid job card.

- b. Modify the JCL to conform to site specifications and specify the following parameters at the beginning of the in-stream procedure:

- eXadas high-level qualifier (CAC).
- SYSOUT class (SOUT).

- c. Supply additional IMS information.

1. The high-level qualifier for IMS data sets (IMS) must be supplied.
2. Specify a PSB name.

3. Specify the correct ACB library for DBB access.
4. If dynamic allocation is not being used (databases not defined to IMS), include DD statements for the database files.

**NOTE:** When running IMS DBB with IRLM=Y, IMS requires the Server address space be non-swappable. CrossAccess strongly recommends setting the JCL SWAP parameter to SWAP=N so IMS will automatically make the address space non-swappable at initialization time. Otherwise, IMS will automatically make the region non-swappable when the first DL/I call is issued, which may result in a significant delay in processing the call. In addition, the IMS BATCH service definition in the Server configuration file should be placed immediately after the logger service to ensure the shortest possible timeframe in making the region non-swappable.

- d. Save the changes.

# 4

## Server Setup for IDMS Access

### Introduction to Server Setup for IDMS Data Access

This chapter describes IDMS access in eXadas. The following topics are discussed:

- [“APF Authorization of the IDMS.LOADLIB,”](#) on page 44,
- [“Setting up Security for IDMS Access,”](#) on page 45,
- [“Setting up a Server to Access an IDMS Central Version,”](#) on page 46,
- [“Mapping IDMS Data for SQL Access,”](#) on page 46,
- [“How IDMS Paths Are Converted Into SQL Rows,”](#) on page 48,
- [“Accessing Multiple Databases in an IDMS Central Version,”](#) on page 48, and
- [“Accessing Multiple IDMS Central Versions from a Single Server,”](#) on page 49.

The IDMS access component runs under the eXadas Query Processor service in a Server. This component supports multiple users without any special eXadas configuration requirements and there are no special initialization services required.

IDMS access is accomplished using the CrossAccess-supplied batch access module *IDMS*. This module can access IDMS data in either central version or local mode. By default, all JCL and examples provided with eXadas are configured to access IDMS in central version mode. If you have a specific need for local mode access, see the IDMS manuals for the necessary JCL changes necessary to allocate and access IDMS databases in local mode.

The underlying access to IDMS is through native DML calls. Since this access is non-SQL, the eXadas Meta Data Utility must be run to map data from the IDMS database and information placed in the eXadas catalogs.

In central version mode, the IDMS data access component in the Server connects to the IDMS DC/UCF system as external run-units. These run-units are established when the eXadas client issues an SQL OPEN on an IDMS mapped table. At the data access level, the run-unit is created when eXadas issues a BIND request for the subschema name defined in the table mapping.

The number of run-units available in a single IDMS DC/UCF region is an IDMS configurable value. Each active DC/UCF system has a MAXERUS (maximum external run-units) value, which limits the number of concurrent external connections. This value governs ALL external connection sources, such as batch jobs and CICS.

Set up and configuration of the Server requires analyzing the MAXERUS value for IDMS central versions accessed and possibly increasing the value for eXadas' use. In general, the number of Query Processor Services running in a Server is the maximum number of concurrent run-units that may be active to IDMS at any point in time. SQL JOINS of IDMS mapped tables will result in the creation of extra run-units, so additional run-units should be available to handle any expected JOINS. In most cases, JOINS will be infrequent as most IDMS databases usually have very few foreign keys on which to JOIN.

## APF Authorization of the IDMS.LOADLIB

Certain eXadas functions such as cross memory services and the security exit CACSX04 require the Server's STEPLIB to be APF-authorized. The IDMS.LOADLIB is not usually APF-authorized and some utility programs in that library will fail if they are run from an APF-authorized STEPLIB concatenation.

If you plan to use eXadas APF-authorized services, you may want to create a separate authorized copy of the IDMS.LOADLIB for eXadas' use. In this case, you may also comment out the IDMS.DBA.LOADLIB in the Server JCL, as it is not necessary for central version access to IDMS.

---

# Setting up Security for IDMS Access

Securing IDMS data through the Server involves the following:

- ensuring the UserID/Password combination passed from the client is an authorized user of the OS/390 System and
- passing the correct user context in all run-units established with the IDMS central version.

## User ID/Password Validation

The system exit CACSX04 automatically validates all user IDs and passwords when a user establishes a connection to the Server. In addition, this exit will validate that the user has READ access rights to the catalog when catalog queries are issued. See [Chapter 15, “System Exits,”](#) for additional information about system exits.

## Passing the Correct User Context To IDMS In Run-Units

By default, all IDMS batch run-units connect to IDMS with a blank user context. This is interpreted in IDMS as PUBLIC access to the data. To establish the correct user name for each run-unit created under the Server, the IDMS.LOADLIB module IDMSSTRT must be re-linked with a USRIDXIT module supplied by CrossAccess. This IDMSSTRT module is for use by eXadas only and should be placed in a library other than the CA supplied IDMS.LOADLIB.

To create a new IDMSSTRT module, assemble the CACIDUXS assembler source supplied by CrossAccess and re-link the CrossAccess module IDMSSTRT, including the CACIDUXS module using the SAMPLIB member, CACIDUXT.

The new IDMSSTRT module can be linked into the eXadas load library if desired. Otherwise, the library containing the new IDMSSTRT module must be placed in the STEPLIB concatenation above the IDMS.LOADLIB. If a separate APF-authorized copy of IDMS.LOADLIB has been created for eXadas use only, you can replace IDMSSTRT in that library if desired.

**NOTE:** The creation of an IDMSSTRT module only ensures that all run-units established with an IDMS central version have the correct user name associated with them. It does not ensure securing of the IDMS data itself. In order for data to be secure, the IDMS central version must have security enforcement active. See your IDMS System Administrator to validate that IDMS data security is enforced.

# Setting up a Server to Access an IDMS Central Version

The JCL changes required to access an IDMS central version are as follows:

1. Add the IDMS.DBA.LOADLIB and IDMS.LOADLIB to the STEPLIB concatenation.
2. Add a SYSCTL DD statement and allocate the SYSCTL file used by the central version you need access to.
3. Add a SYSIDMS DD statement with a 'DBNAME=*default database name*' card so tables mapped without an explicit database name have a default.

**NOTE:** The IDMS.LOADLIB is a non-APF-authorized library, and will remove authorization from STEPLIB if allocated to STEPLIB. If the Server is using any eXadas authorized services (Cross Memory Services and any SAF Exits such as CACSX04 and CACSX07), S047 abends will occur.

## Mapping IDMS Data for SQL Access

Mapping IDMS data for use by eXadas includes defining logical tables which access single records or a specific path through an IDMS database. To define a mapping, the eXadas DataMapper loads an IDMS schema and subschema report and converts record layouts into SQL columns definitions. When mapping a path of records, mapping starts with a single record and traverses sets to additional records in the schema.

IDMS schema and subschema reports are produced by running the IDMS schema and subschema compilers and capturing the punched output into an OS/390 data set. JCL to punch these reports can be found in SAMPLIB with the member name CACIDPCH.

**NOTE:** Before running the supplied JCL, you must know which databases, schemas, and subschemas you want to map.

The basic steps required to map IDMS Data are:

1. Punch IDMS schema and subschema reports on the mainframe.
2. Transfer schema/subschema reports to the PC on which the eXadas DataMapper is installed\*.
3. Start the DataMapper.
4. Load the schema/subschema report for which you want to create logical mappings.

5. Create a Data Catalog of type IDMS.
6. Create logical tables for any desired records/paths through the schema.
7. Import the columns from the schema report.
8. Generate the Meta Data Grammar statements.
9. Transfer the generated Meta Data Grammar back to the mainframe\*.
10. Run the eXadas Meta Data Utility to populate the system catalogs used by the Server.

\* TCP/IP users can transfer mainframe files between host systems and the PC from within the DataMapper using an FTP facility included in the DataMapper.

See the IDMS tutorial in the *eXadas DataMapper Guide* for more specific details about using the DataMapper to map IDMS Data.

## Running the Meta Data Utility (METAU) with IDMS Meta Data Grammar

In addition to the Meta Data Grammar created by the DataMapper, the Meta Data Utility requires access to the IDMS schema/subschema reports and access to the IDMS central version. IDMS central version access is required only if mapping schema records have SYNONYMS for a SHARED STRUCTURE. These SYNONYMS require central version access to determine whether any prefixes or suffixes are defined for the elements in a record. For the prefix/suffix look up to work correctly, the SYSIDMS DD statement in the Meta Data Utility JCL must contain a 'DBNAME=xxxxxxx' specification for the dictionary database name that contains the mapped schemas.

**To set up the Meta Data Utility for IDMS mapping, make the following changes to the standard Meta Data Utility JCL:**

1. Add a DDLPUNCH DD statement and allocate all schema and subschema report files referenced in the Meta Data Grammar. Multiple schemas and subschemas can be concatenated to this DD name if the Meta Data Grammar apply to more than one schema/subschema combination.
2. Add a SYSCTL DD statement referencing the central version from which the schema/subschema reports were produced.
3. Add a SYSIDMS DD statement with a DBNAME=dictionary-name specification where dictionary name is the name of the dictionary containing the schemas reports included in the DDLPUNCH DD statement.

While the server itself can access multiple IDMS central versions, the Meta Data Utility is restricted to a single dictionary in a single central version for SYNONYM lookup purposes. Therefore, you must restrict the schemas and table

definitions for each run of the Meta Data Utility to a single dictionary in a single IDMS central version.

## How IDMS Paths Are Converted Into SQL Rows

Each IDMS Table represents a single record or path through an IDMS schema. A path is defined by starting with a single record, and then navigating sets to additional records in the schema.

When the server returns SQL rows for a logical table mapped to a path, it returns an instance of the first record type mapped with each instance of related records down the defined path. For example, given the following path:

Employee RECORD → Empl\_Dep SET → Dependent RECORD

if the database contains the following records:

```
Employee Empl_Dep SETDependent
SET relationships
BILL SMITH→MARTHA
           BILLY
           SALLY
JANE WHELAN→
SANDRA JONES→ROBERT
```

The Query to retrieve all the rows in the mapped table returns:

EMPL_NAME	DEPENDENT NAME
BILL SMITH	MARTHA
BILL SMITH	BILLY
BILL SMITH	SALLY
JANE WHELAN	-----
SANDRA JONES	ROBERT

Total Rows Fetched: 5

## Accessing Multiple Databases in an IDMS Central Version

Each active IDMS central version generally provides access to multiple databases. As described earlier, each SQL OPEN of an IDMS mapped table in eXadas



creates a run-unit by binding to a subschema in an IDMS central version. A database name is also passed in the BIND if it is specified in the Meta Data Grammar in the logical definition.

By explicitly specifying the appropriate database name for IDMS mapped tables, you can access as many databases as desired in a single IDMS central version. Tables without an explicit database name require a default database name defined for the server. This name can be specified in the SYSIDMS JCL DD allocation as a 'DBNAME=xxxxxxx' card or in a custom IDMS access module containing an IDMSOPTI specification.

## Accessing Multiple IDMS Central Versions from a Single Server

The SYSCTL data set allocated to the server identifies the default IDMS central version to communicate with. If desired, multiple SYSCTL data sets with unique DD names can be allocated to a single Server and selected on a table-by-table basis using custom built IDMS ACCESS LOADMODs that reference the appropriate SYSCTL DD name.

**To build and reference a custom access load module whose SYSCTL DD name is SYSCTL1 instead of the default value of SYSCTL:**

1. Code up an assembler IDMSOPTI module containing the following assembler macro statement.

```
IDMSOPTI  CENTRAL=YES , SYSCTL=SYSCTL1
END
```

2. Assemble the IDMSOPTI module using the supplied SAMPLIB member CACIDACM.
3. Re-link the CrossAccess-supplied batch access module named IDMS and include the IDMSOPTI module assembled in step 2. Sample link-edit control statements to build the new access module are:
  - INCLUDE IDMSLOAD (IDMS)
  - ENTRY IDMS
  - NAME CACIDMS (R)

Be sure to create a new name for the IDMS module as the default module should be left as-is for other IDMS batch applications.

4. Use the eXadas DataMapper to specify the new batch access module name by specifying an Access Module Name of IDMSCTL1.

5. Generate the new Meta Data Grammar and rerun the eXadas Meta Data Utility to update the eXadas catalogs with the new access module.

IDMSOPTI modules can also be used to manage default databases and other IDMS specific parameters as well. For more information, see the *CA/IDMS System Operations* manual.

# 5

## Server Setup for CA-DATACOM/DB

### Introduction to Server Setup for CA-DATACOM/DB

This chapter describes how to set up the Server to enable CA-DATACOM/DB support.

Configuring CA-DATACOM/DB support in eXadas requires you to:

- Define the Datacom Initialization Service for the Server.
- Ensure the CA-DATACOM/DB Multi-User Facility is running authorized. See the *CA-DATACOM/DB Database and System Administrator Guide* for more details.
- Set up CA-DATACOM/DB data access.
- Set up security for CA-DATACOM/DB access (this is an optional step). For more information, see the *CA-DATACOM/DB SQL User Guide*, *CA-DATACOM/DB Security Guide* and other appropriate CA-DATACOM/DB documentation.

These steps are described in the sections that follow.

# Define the Datacom Initialization Service to the Server

CA-DATACOM/DB access requires a SERVICE INFO ENTRY in the Server configuration file for the Datacom Initialization Service.

As mentioned previously, CA-DATACOM/DB database connections are created and managed by a separate Datacom Initialization Service. The following example shows the configuration of this service:

```
SERVICE INFO ENTRY = CACDCI DCOMIS 2 1 1 50 4 5M 5M 4
```

This service has a minimum task count of 1, a maximum task count of 1 and a maximum connections count of 50. The initialization task requires the maximum tasks (field 5) to be 1 as only one instance of this service is required to handle any number of connections.

Field 10 contains a numeric value. It is the total number of CA-DATACOM/DB task areas to be acquired and initialized for use by the Query Processor. This value defines how many concurrent connections can be attained with CA-DATACOM/DB. Based on the example definitions shown above, a maximum of 4 connections can exist with the CA-DATACOM/DB system at any given time.

# Ensure the Multi-User Facility Is Running Authorized

CA recommends authorizing the CA-DATACOM/DB load library because certain Multi-User Facility options are operational only when the Multi-User Facility is authorized. If authorization is not present when the Multi-User Facility is started, the MUF issues the following message:

```
DB00210I - MULTI-USER NOT RUNNING AUTHORIZED.
```

See the *CA-DATACOM/DB Database and System Administrator Guide* for more detail.

---

# CA-DATACOM/DB Data Access

In the Query Processor, CA-DATACOM/DB table information is mapped into eXadas system catalogs by the Meta Data Utility. Access to the data is processed through the Query Processor. This processing parses SQL queries and converts them into native CA-DATACOM/DB commands for database access. The benefit of this approach is that it provides the capability of joining CA-DATACOM/DB data with other data types.

The Query Processors in a single Server are restricted to accessing the same CA-DATACOM/DB Multi-User Facility. To access a different Multi-User Facility requires a second server referencing a different CA-DATACOM/DB control library. For information on how to access a different CA-DATACOM/DB MUF, see “Modifying DBSIDPR Parameters” in the *CA-DATACOM/DB Database and System Administrator Guide*.

Access to CA-DATACOM/DB data is handled by a series of CA-DATACOM/DB supplied interface modules. The actual database connections are provided using pre-allocated task areas. Each separate task area represents a TCB communicating between the eXadas Query Processor and the CA-DATACOM/DB Multi-User Facility. The number of task areas for use by the Query Processor is defined in the SERVICE INFO ENTRY for the Datacom Initialization Service and it limits the number of concurrent connections to a CA-DATACOM/DB MUF.

Task areas assigned for use by the Query Processor can be shared by non-update queries requiring the same User Requirements Table. A query containing update type statements requires a single non-shared task area. It is difficult, based upon the possible mix of queries, to calculate the limiting factor that controls the maximum total number of users allowed a connection to CA-DATACOM/DB at any given instant.

Assuming all queries are non-update and all queries require a different User Requirements Table, the maximum total number of users allowed a connection would be the smaller of the following:

- the number of Query Processor instances times the maximum number of connections per instance or
- the number of task areas assigned for use by the Query Processor.

Attempts to connect to either the Query Processor or to CA-DATACOM/DB that exceed the maximum thresholds you specify in your configuration file values will be rejected with an error message.

# Setting Up CA-DATACOM/DB Security

Your choices for security implementation include internal CA-DATACOM/DB security (not recommended by CrossAccess), external security (RACF, ACF-2 or Top Secret), or no security at all.

If you chose an external type security package, each user must have a unique identity. Servers run as either an OS/390 started task or a batch job. The primary authorization ID, from either of these types of tasks, is the user of the started task or batch job and in many installations, the user for started tasks is specified as plus signs ('+++++++'). Since the Server is a multi-user system, accessing CA-DATACOM/DB under the authorization ID of the Server itself will not provide adequate security for your installation's data.

Therefore, eXadas has a mechanism for each user of the server to be identified by the user ID supplied when they connect to the Server. This user ID and the associated password must be verified using RACF, or the security system installed at your site, as part of the connection processing. This validation is enabled by specifying SAF Exit = CACSX04 in the master configuration file for the Server. To extend user level security checking to each database request, the ACEE created during this security checking must be made available outside the security exit. All database processing is then done under the particular user ID identified in the ACEE.

If you chose no security at the database level and the SAF Exit is active in the Server configuration file, you must indicate to the SAF Exit that the Query Processor should not be provided with security information. You do this by providing a keyword parameter (EXCLUDE = n) in the SAF Exit configuration file entry. For detail information on how to specify this parameter see [“SAF Exit API Overview,” on page 224](#).

# 6

## Communication Configuration

### Introduction to Communication Configuration

This chapter describes the communication configuration options as well as the Server configuration required to achieve Cross Memory, IBM MQ Series, or TCP/IP communications. The following topics are discussed:

- [“Communications Options,”](#) on page 56,
- [“Selecting a Communications Option,”](#) on page 64, and
- [“Server Configuration,”](#) on page 65.

A Server can accept communication connections from both local and/or remote client applications. Configuration parameters, on the server and the client, provide for the connections.

Three parameters are involved in configuring a client application to Server connection, the DATASOURCE parameter in the client and two SERVICE INFO ENTRY parameters in the Server, one for a Query Processor (CACQP), and the other for a Connection Handler Service (CACINIT).

The client's DATASOURCE parameter specifies two subparameters:

- a data source name and
- a communications compound address field.

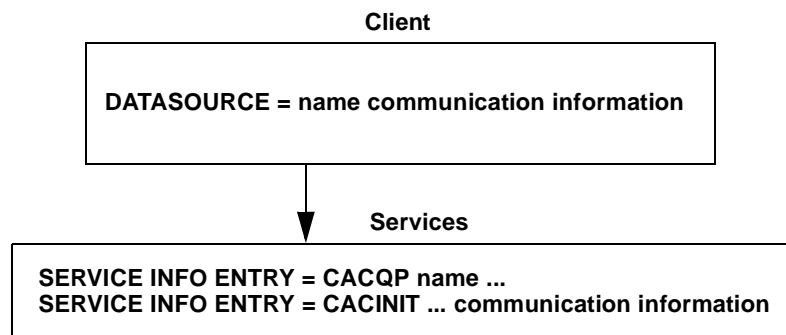
The parameter descriptions follow.

The Data Source Name field is used to identify which Query Processor service task in the targeted server handles requests for this client application. The server configuration must include a SERVICE INFO ENTRY for a Query Processor, with a matching data source name specified as the Service Name for a connection to be established.

The Communication's Compound Address field is used to establish a communications path to the server that contains the targeted Query Processor task. The Server configuration must include a SERVICE INFO ENTRY for a Connection Handler task (CACINIT) that specifies this compound address in its field 10. A Server configured in this way will listen for connections on that address. A client application then connects to that address and a communication session results.

The following table illustrates the relationship between the three parameters.

**Figure 6: Communication Configuration Parameter Relationships**



## Communications Options

eXadas supports the following communications options:

- Cross Memory,
- IBM MQ Series,
- TCP/IP.



## Cross Memory

There are no additional set-up requirements to use the OS/390 Cross Memory interface, such as those required for TCP/IP. This interface uses OS/390 data spaces and OS/390 token naming services for communications between client applications and Servers. The components of the communications compound address field specific to Cross Memory are the Cross Memory protocol identifier (XM1), the data space name, and the queue name. This compound string represents a unique address that must match in the corresponding parameters within the client application and Server configurations (as described previously) for a connection to take place. The data space and the queue name fields each have a maximum length of four characters.

Each Cross Memory Data Space can support up to a maximum of 400 concurrent users, although in practice this number may be lower due to resource limitations. To support user populations in excess of this number on a Server, configure multiple Connection Handler services, each with a different data space name.

The **communications compound protocol field** consists of several fields. The first field identifies the protocol. The remaining fields are particular to that protocol. An example of the communications compound address field follows:

```
XM1/DataSpace/Queue
```

## IBM MQ Series

eXadas supports IBM MQ Series messaging middleware as an alternative to the existing TCP/IP or OS/390 Cross Memory Services transport services that are used for client-server intercommunications. MQ Series support is provided between OS/390 servers/clients and NT ODBC clients.

CrossAccess assumes that you are familiar with MQ Series concepts and terminology. You should have a working knowledge of how to configure and operate MQ Series on OS/390. For additional information on MQ Series, see the IBM MQ Series documentation.

The following are supported MQ Series release levels for OS/390 and NT:

- OS/390: eXadas supports MQ Series Release 1 Version 2, or above.
- NT: eXadas supports MQ Series Clients or Servers running MQ Series Release 5 Version 0, or above.

The following topics are discussed in this section:

- [“Conceptual Overview,” on page 58](#), describes how eXadas uses MQ Series as a transport mechanism.
- [“Prerequisites to Using MQ Series,” on page 62](#), describes the infrastructure that must be in place before eXadas can use MQ Series.
- [“OS/390 Queue Manager Definitions,” on page 62](#), provides sample queue definitions that allows an eXadas Server, Enterprise Server, or MQ Series Client to communicate with an OS/390 Server using MQ Series.

## Conceptual Overview

eXadas uses MQ Series as a transport mechanism between clients and Server. Clients can use MQ Series to communicate with an OS/390 Server and can be deployed on OS/390 or on NT.

The eXadas MQ Series implementation is referred to as a transport layer since eXadas does not use any advanced MQ Series facilities, such as message persistence or two-phase commit protocols. For NT clients, using MQ Series as a transport mechanism still uses TCP/IP as the underlying transport mechanism between the client and Server.

The advantage of using MQ Series as a transport mechanism is that you use MQ Series to configure connectivity between the client and server in the same manner that you configure other applications that use MQ Series. Additionally, MQ Series provides protocol independence between client and server. For example, you can use TCP/IP for communications between an eXadas client and an NT MQ Series server. You can also multi-hop over multiple OS/390 MQ Series servers, if desired. In all instances, eXadas is not aware of the underlying protocols being used.

For eXadas to use MQ Series as a transport vehicle, at a minimum two queue definitions are required. One of these queue definitions is a local queue that the server listens on for requests from clients. The other queue is a temporary dynamic model queue that is used by clients. The client connects to the local queue definition and sends messages to that queue for processing by the server. The server puts response messages on the instance of the temporary dynamic queue (that is created when the client opens the temporary dynamic model queue) which the client subsequently retrieves and processes. [Figure 7: “Basic eXadas MQ Series Architecture,” on page 59](#), shows how clients and servers communicate using MQ Series.

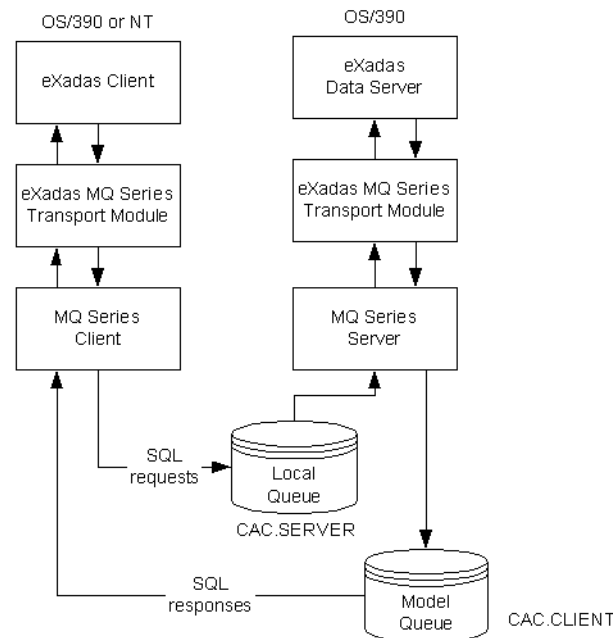
**Figure 7: Basic eXadas MQ Series Architecture**

Figure 7: “Basic eXadas MQ Series Architecture,” shows the basic MQ Series architecture that is required for a local OS/390 eXadas client or remote NT eXadas client (either ODBC or JDBC) to communicate with an OS/390 server using MQ Series. The diagram shows that two queues are defined to the OS/390 MQ Series Queue Manager. The queue named CAC.SERVER is the local queue definition that the CAC MQ Series Transport Module accesses to receive SQL requests from OS/390 or NT clients.

In the diagram, the CAC.CLIENT queue is the temporary dynamic model queue that the Server places messages on in response to SQL Requests from a client. During initialization processing, the client opens the CAC.CLIENT temporary dynamic queue that causes MQ Series to create a unique queue name for use by the client. When the client sends a message to the CAC.SERVER local queue, the MQ message header contains the name of the reply-to queue, which in this case is the unique name of the CAC.CLIENT queue assigned to the client by MQ Series. Once the Server has finished processing a client SQL request, the OS/390 MQ Series Transport Module sends a message to the reply-to queue name identified in the originating message from the client.

**NOTE:** You can use any queue name that you like for the local and temporary dynamic queue names. The use of CAC.CLIENT and CAC.SERVER are for illustrative purposes only.

Figure 7: “Basic eXadas MQ Series Architecture,” shows MQ Series Clients directly connecting to the OS/390 MQ Series Queue Manager. The following figure shows another common implementation where an intermediate queue manager is used.

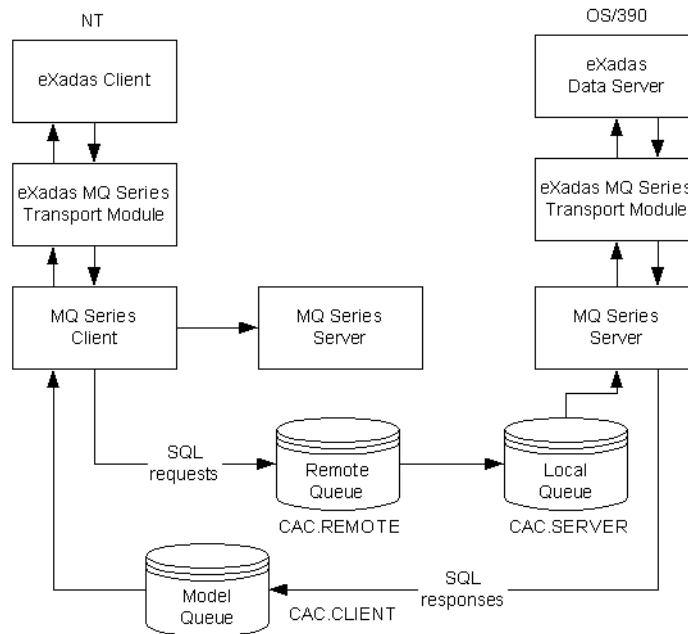
**Figure 8: Using Two Queue Managers**

Figure 8: “Using Two Queue Managers,” shows an implementation that uses an intermediate NT MQ Series Queue Manager. In this diagram, the MQ Series Client would connect to the NT MQ Series Queue Manager using TCP/IP or a LAN-based protocol like NetBIOS or SPX. As can be seen in this diagram, there are three queue definitions that are required. Additionally, the temporary dynamic model queue is now defined to the NT MQ Series Queue Manager. In this implementation, a remote queue definition is also required at the NT MQ Series Queue manager that references the CAC.SERVER local queue that is defined on OS/390.

Communications between the eXadas Client and Server is similar to that shown in Figure 7: “Basic eXadas MQ Series Architecture.” In this scenario the eXadas Client is configured to open and send messages to the CAC.REMOTE queue. This causes messages to be sent to the CAC.SERVER queue on OS/390 where the Server can pick these messages up for processing. The Server sends replies to the SQL requests sent by the client. Using standard MQ Series routing protocols the SQL Responses are sent to the instance of the temporary dynamic model queue that was created on NT when the client opened the CAC.CLIENT queue.

As in the previous diagram, the queue names CAC.CLIENT, CAC.LOCAL and CAC.REMOTE are for illustrative purposes only. You can assign any name to these queue definitions.

The following figure shows one possible configuration when an Enterprise Server has been deployed and you want to use MQ Series for communications between the eXadas Client, the Enterprise Server and an eXadas server.

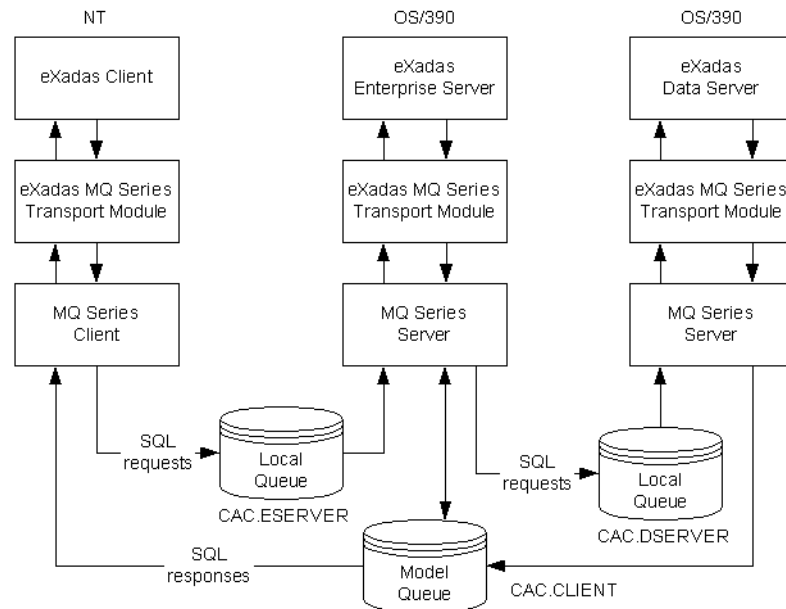
**Figure 9: Sample Enterprise Server Configuration**

Figure 9: “Sample Enterprise Server Configuration,” shows that an NT (or OS/390) MQ Series Client is directly connecting to the Enterprise Server. In this configuration, two local queues are defined to the OS/390 MQ Series Queue Manager. Clients send messages to the Enterprise Server using the CAC.ESERVER local queue. The CAC.DSERVER local queue is used by the Data Server to receive and process messages sent by the Enterprise Server.

In the diagram, a single temporary dynamic model queue has been defined on OS/390. The eXadas Client and the eXadas Server/Enterprise Server return SQL Responses and both use the CAC.CLIENT queue. In actuality two queue instances are used. The first is for messages sent by the Enterprise Server for pick-up by the client. The second is for eXadas Server responses to the Enterprise Server.

In this scenario, the client sends messages to local queue CAC.ESERVER. When the client opens the CAC.CLIENT queue a unique queue name is assigned that the Enterprise Server uses to route SQL responses back to the client. Based on configuration information in the Enterprise Server’s master configuration member, when a client connects to the Enterprise Server, the Enterprise Server will:

1. Open a temporary dynamic model queue (in the diagram CAC.CLIENT) for SQL Responses from the eXadas Server.
2. Forward the message to the Server’s local queue (in the diagram CAC.DSERVER).
3. Wait for responses on the temporary dynamic model queue from the Server.

When a response is received from the Server, the message is picked up from the temporary dynamic model queue. The Enterprise Server then puts the message on the client’s temporary dynamic model queue so that the client can pick up the SQL Response.

**NOTE:** When MQ Series is used for communications between an Enterprise Server and an eXadas Server, the Enterprise Server can start a single instance of the eXadas Server. If multiple instances of the Server are started then client messages placed on the Server's local queue definition are processed randomly by the different Server instances that are running. This causes failures in the client and unpredictable results in the eXadas Server.

It is also possible to create a configuration, using the Enterprise Server and an intermediate queue manager. Using [Figure 8: "Using Two Queue Managers,"](#) on [page 60](#) as an example, define a temporary dynamic model queue (CAC.CLIENT) at the NT MQ Series Queue Manager. Then create a remote queue definition (CAC.REMOTE) that references the local queue used by the Enterprise Server (CAC.ESERVER). At the OS/390 MQ Series Queue Manager, you would define the queues shown in ["Sample Enterprise Server Configuration,"](#) on [page 61](#).

## Prerequisites to Using MQ Series

In the previous section we discussed and showed several typical configurations that you may use to establish connectivity between an eXadas Client and OS/390 Server/Enterprise Server using MQ Series. Before you attempt to implement any of these configurations, you must have the infrastructure in place to allow communications between the different MQ Series components.

The following should be in place prior to implementing a configuration using MQ Series:

1. The MQ Series OS/390 Queue Manager has been installed and configured for communications between any other queue managers that eXadas will be using (or passing through) and/or the MQ Series clients that will be connecting to the OS/390 Queue Manager.
2. The MQ Series Client (or an NT MQ Series Server) has been installed on the NT workstation where the eXadas client is/will be installed.
3. You have tested connectivity between all MQ Series components by putting and getting messages with the MQ Series supplied utility programs.

**NOTE:** If you do not have your own queues for test purposes you can put and get test messages from the local queue that the eXadas Server will be using. When the server starts up, it will retrieve the test messages, determine that a reply-to queue does not exist, and then discard the message(s) from the local queue.

## OS/390 Queue Manager Definitions

eXadas requires two queues to be defined on the OS/390 MQ Series Server:

- a local queue definition, and
- a model queue definition.

The names of these queues are not restricted. In the following example the queue named CAC.SERVER is used for the local queue definition and CAC.CLIENT for the model queue definition.

The command line definitions for both of these queues are as follows:

```
DEFINE QLOCAL(CAC.SERVER) DEFISOFT(SHARED) SHARE

DEFINE QMODEL(CAC.CLIENT)
```

You can either use the default STGCLASS or set this parameter to one of your custom STGCLASS definitions.

**NOTE:** If you are using intermediate queue managers (as shown in [Figure 9: “Sample Enterprise Server Configuration,”](#) on [page 61](#)), the CAC.CLIENT model queue definition is the same. Additionally, you are responsible for defining the remote queue definition that is associated with the CAC.SERVER local queue definition. For the remote queue definition you must identify a transmission queue that has connectivity to the OS/390 MQ Series Queue Manager that eXadas is using.

After issuing the two DEFINE commands shown above, you can use the **DISPLAY QLOCAL(CAC.SERVER)** and **DISPLAY QMODEL(CAC.CLIENT)** MQ Series MTO commands to verify that queues have been defined properly.

## TCP/IP

TCP/IP communications requires the definition of the IP address of the TCP/IP communications stack that the Server is running on and the specification of a listen port number, in addition to the TCP/IP protocol identifier, for example, TCP. The IP address can be specified using dot-decimal notation, for example, 011.022.033.044, or as a hostname. The port number can be specified as a four digit value, for example 9999, or as a service name.

A single TCP/IP Connection Handler Service can service 255 concurrent users.

Multiple sessions are created on the specified port number. The number of sessions carried over the port is the number of concurrent users to be supported plus one for the listen session the Connection Handler uses to accept connections from remote clients. If the TCP/IP implementation you are using requires the specification of the number of sessions that can be carried over a single port, you must ensure that the proper number of sessions have been defined. Failure to do so will result in a connection failure when a client application attempts to connect to the server.

The IP address and port number are specified in the service information field of the SERVICE INFO ENTRY parameter in the Server. Failure to define a correct IP address causes the Connection Handler Service (in the Server) to fail during initialization. Specification of a port number that has been assigned to another application causes unpredictable results for both the Server and the application that is using the port.

The following example shows a TCP/IP communications compound address field:

```
TCP/119.23.1.2/5000
```

## Selecting a Communications Option

Proper selection and configuration of a Connection Handler Service can have a dramatic affect on query performance. If you are accessing an OS/390 Server locally, then CrossAccess recommends you use a Cross Memory Connection Handler Service. If your application is accessing a Server remotely you need to consider the following options in order to obtain the best query performance.

### Bandwidth

The more bandwidth that is available the faster a query will be processed. This is especially true when the number of concurrent users is large. If the client applications are physically close, use of an ESCON channel is recommended. For clients that are farther away, use of ATM or frame relay is recommended. For dial-up clients, use of ISDN or the fastest modem that is available is recommended.

### TCP/IP Use Of Hostnames vs. IP Addresses

Using a hostname requires the availability of a configured local HOSTS file or a domain name server. If a domain name server is involved, then there is some overhead required to resolve the HOST name to the correct IP address. However, CrossAccess recommends the use of hostnames in remote client configuration files for readability and ease of future configuration changes.

Specifying a hostname for the Server's SERVICE INFO ENTRY for the Connection Handler Service that the remote clients will use to connect to the Server does not affect overall Server performance. CrossAccess recommends using hostnames for Server Connection Handler Service configuration since it makes it easier to verify that the remote clients are communicating with the correct OS/390 Initiator instance.

Using hostnames also makes it easier to change IP addresses if the environment changes. If hostnames are used, frequently the Server/remote clients will not have to be reconfigured. eXadas can be brought down and the network administrator can change the IP address for a hostname in the OS/390 and client domain name



server(s). When the Server is restarted it will automatically listen on the new IP address for connection requests from remote clients. When a remote client connects to the Server it will automatically use the new IP address that has been assigned to the hostname.

## Server Configuration

This section describes how to configure a Server to allow remote access from client applications. Step-by-step instructions on how to configure Cross Memory, IBM MQ Series, and TCP/IP connectivity are provided.

**WARNING:** Remote access can only be configured after your network administrator has set up the network infrastructure to allow remote clients to communicate with the OS/390 host system.

### Cross Memory

**To configure the server for Cross Memory:**

1. Edit the Server master configuration member (SCACCONF member CACDSCF).

**NOTE:** All sample JCL, configuration files, and sample application programs referenced in this chapter can be found in the SCACSAMP and SCACCONF libraries included with your eXadas software shipment.

The sample master configuration member supplied contains default parameter values for the configuration parameters that are required for Server installation verification. The sample master configuration member also contains a number of SERVICE INFO ENTRIES that are commented out. These SERVICE INFO ENTRIES are for use by the eXadas sample application (CACCLNT). Once fully-activated, you will be able to access the eXadas database/files both locally and remotely.

2. Activate the default Cross Memory Connection Handler Service.

Uncomment the SERVICE INFO ENTRY for the Cross Memory Connection Handler Service. This SERVICE INFO ENTRY can be identified by the comments in the master configuration member.

In the task data field the name of the Cross Memory compound address (XM1/data space/queue names) that the Connection Handler Service uses to communicate with client applications is specified. In the sample, the queue name is CAC. This name must also be specified in the local client's configuration file. The data space/queue name pair must be unique for each

configured connection with a maximum length of four characters for each field.

3. Save the master configuration member.

The Server is now configured for Cross Memory communication. In order for the Server to pick up the new definition you must stop/start the Server. Alternately, you can use the MTO interface to activate the connection.

## IBM MQ Series

The following steps describe how to configure the Server to support IBM MQ Series communications from a local OS/390 client or a remote client application.

1. Modify the Server JCL

You may have to modify the eXadas-supplied sample JCL for the Server. Any OS/390 component using MQ Series must have access to the following MQ Series libraries:

- *thlqual.SCSQANLx* and
- *thlqual.SCSQAUTH*

where *thlqual* is replaced with the high-level qualifier for your MQ Series installation and *x* is replaced with the language letter to be used (probably E).

If these libraries are globally available (in LPA) then no JCL modifications are required. Otherwise the eXadas Server/Enterprise Server must have the previous two libraries concatenated into the STEPLIB DD statement for the first step in the Server's JCL (EXEC PGM=CACCNTL).

**NOTE:** If you are planning to configure CACCLNT to use MQ Series, the above libraries must also be accessible. If the libraries are not globally available modify the JCL and concatenate the MQ Series libraries on the STEPLIB DD statement.

2. Obtain the name of the Local Queue name that the Server will use.

Contact your MQ Series administrator and obtain the names set up for use by eXadas. A different Local Queue name is required for each MQ Series Connection Handler Service that you configure in the Server.

3. Edit the Server master configuration member (SCACCONF member CACDSCF)

The sample master configuration member supplied contains default parameter values for the key configuration parameters that are required to execute a Server. The sample master configuration member also contains a set of SERVICE INFO ENTRIES that are commented out. These SERVICE INFO ENTRIES apply to the eXadas sample applications. Once fully activated, you will be able to access eXadas database/files both locally and remotely.

4. Activate the IBM MQ Series Connection Handler Service

Uncomment the SERVICE INFO ENTRY for the IBM MQ Series Connection Handler Service. This SERVICE INFO ENTRY can be identified by the comments in the master configuration member.

The last field in the SERVICE INFO ENTRY (the task data field) parameter specifies the name of the local queue used to accept connections from local and remote client applications. This name, in addition to a model queue name, must also be specified in the local or remote client's configuration file on the DATASOURCE parameter.

For more information on the SERVICE INFO ENTRY parameter see [Appendix A, "Configuration Parameters."](#)

5. Save the master configuration member.

The Server is now configured for IBM MQ Series connectivity. In order for the Server to pick up the new Connection Service Handler definition you must stop and restart the Server.

## TCP/IP

### To configure the server to support TCP/IP communications from a remote client application:

1. Obtain the name/IP address of the OS/390 machine and the port number that the Server will use.

Contact your network administrator and obtain the name or IP address of the OS/390 system that the server is running on. Also obtain the port number or service name that has been assigned for use by eXadas. A separate service name/port number is required for each TCP/IP Connection Handler Service that you configure in the Server.

2. Edit the Server master configuration member (SCACCONF member CACDSCF).

The sample master configuration member supplied contains default parameter values for the key configuration parameters that are required to execute a Server. The sample master configuration member also contains a set of SERVICE INFO ENTRIES that are commented out. These SERVICE INFO ENTRIES are for the eXadas sample applications. Once fully activated, you will be able to access the eXadas database/files both locally and remotely.

3. Activate the TCP/IP Connection Handler Service.

Uncomment the SERVICE INFO ENTRY for the TCP/IP Connection Handler Service. This SERVICE INFO ENTRY can be identified by the comments in the master configuration member.

The last two fields in the SERVICE INFO ENTRY (the task data field) parameter specify the name/IP address of the OS/390 host machine and the service name/port number that is used to accept connections from remote client applications. The same name/IP address and service name/port number

must also be specified in the remote client's configuration file on aDATA SOURCE parameter.

For more information on the SERVICE INFO ENTRY parameter see [Appendix A, "Configuration Parameters."](#)

**NOTE:** The port number or service name must not be in use by any other application, and should be greater than 1024. (The port numbers 1 through 1024 are reserved by convention for well-known TCP/IP systems services.)

If your OS/390 TCP/IP system is using off-load gateways ensure that the IP address that is specified reflects the IP address of the OS/390 TCP/IP stack, not the address of an off-load gateway device's IP stack.

If you are using Interlink TCP/IP then the hostname must be specified as an IP address in dot-decimal notation and must be all zeros (000.000.000.000). See your Interlink documentation for information on configuring a server application.

4. Save the master configuration member.

You have completed configuring the Server for TCP/IP connectivity. For the Server to pick up the new Connection Handler Service definition you need to stop and restart the server. Alternately, you can use the MTO interface to activate the Connection Handler Service. For more information on using the MTO interface to perform dynamic configuration, see [Chapter 10, "Server Operations."](#)

# 7

## SQL Security

### Introduction to SQL Security

Security of your data is of primary importance, particularly in an SQL-based DBMS, because interactive SQL makes database access very easy. The security requirements of production databases can include such things as data in any given table accessible to some users, but denied to others and some users allowed to update data in a table, while others are only able to view data.

SQL security provides protection for these scenarios. This chapter describes:

- [“eXadas Security Concepts,”](#) on page 70,
- [“User Types,”](#) on page 70,
- [“Database Objects,”](#) on page 71,
- [“Defining User Privileges,”](#) on page 72,
- [“Authorization Requirements,”](#) on page 81,
- [“SQL Security and the eXadas SAF Exit,”](#) on page 82, and
- [“Summary,”](#) on page 83.

# eXadas Security Concepts

SQL security in eXadas is similar to DB2 security. Implementing security and enforcing security restrictions are the responsibility of the DBMS software. SQL defines an overall framework for database security and SQL statements are used to specify security access and restrictions.

There are several key concepts to understanding SQL security:

- **Users** are the actors in the database. When the DBMS retrieves, inserts, deletes, or updates data, it does so on behalf of a user or group of users. The DBMS permits or denies user actions depending on which user makes the request. eXadas allows you to define users and user groups based on five user types.
- **Database objects** are the items to which SQL security can be applied. Security is usually applied to tables, views, and stored procedures. Most users have permission for certain database objects, but are denied access to others.
- **Privileges** are the actions that a user is permitted to carry out for a particular database object. For example, a user may have permission to SELECT and INSERT rows in one table, but lack permission to DELETE or UPDATE rows in that table. These privileges are allowed or disallowed by using the GRANT and REVOKE SQL statements.
- **SQL Security and the SAF Exit** work together in eXadas to ensure that the user ID and its password are checked prior to allowing access to particular database objects.
- **Enabling SQL Security in eXadas** is required since eXadas is delivered with Security completely disabled.

These concepts are described in detail in the remainder of this chapter.

## User Types

Each user or group of users in the database is assigned a user ID that identifies the user to the DBMS. The user ID is the heart of security in SQL. In eXadas, each user ID is associated with a particular user type.

There are five categories of user types you may assign in eXadas:

- **SYSADM.** This is the system administrator and has privileges for all objects including the ability to grant authorizations to other users. The first user to run the Meta Data Utility is granted SYSADM authority. See [“Summary,” on page 83](#), for additional information about enabling security.
- **SYSOPR.** This user type has remote operator privileges to display and manage an active Server.

- **DISPLAY.** This user type has remote operator privileges for display commands only on an active Server.
- **DBADM.** This user type has mapping/VIEW creation privileges for specific database types.
- **PUBLIC.** Limited to privileges explicitly granted to their user names or to PUBLIC.

User types are assigned to individual users or groups of users by the SQL GRANT command of the type authority (GRANT SYSADM TO user\_or\_group\_name, GRANT SYSOPR TO user\_or\_group\_name, GRANT DBADM ON DATABASE database\_name to user\_or\_group\_name, etc.). The user ID determines whether the statement will be permitted or prohibited by the DBMS. In production databases, user IDs are assigned by the database administrator. For additional information about the user privileges, see [“Defining User Privileges,” on page 72.](#)

## Database Objects

Catalog database types are database objects to which security can be applied. In order to manage or secure the eXadas Meta Data Utility for mapping purposes, the following implicit database names were added to the system catalogs. These system catalogs map one-to-one with the eXadas Data Savants.

**NOTE:** The \$CFI type does not map to a Data Savant because it is the SYSTEM CATALOG.

The eXadas catalog database types are:

- \$ADABAS for ADABAS database mappings.
- \$CFI for SYSTEM CATALOG.
- \$DATACOM for Datacom database mappings.
- \$IDMS for IDMS database mappings.
- \$IAM for IAM database mappings.
- \$IMS for IMS database mappings.
- \$SEQUENT for Sequential database mappings.
- \$SP for Stored Procedure definitions.
- \$VSAM for VSAM database mappings.

Database types are specified in the GRANT DBADM SQL command. For example:

```
GRANT DBADM ON DATABASE $IMS TO USER1
```

For eXadas, users mapping tables must have SYSADM authority to run the Meta Data Utility as the Meta Data Utility does not have database level security access checking. DBADM granting is only used for DROP commands.

## Defining User Privileges

Privileges are the set of actions that a user can carry out against a database object. Privileges are defined for users with the SQL commands GRANT and REVOKE. GRANT assigns one or more privileges to specific users. REVOKE removes one or more privileges from specific users. eXadas allows you to GRANT or REVOKE the following privileges for a user:

- System,
- Database,
- Stored Procedures, and
- Tables or Views.

GRANT and REVOKE statements can be issued in the Meta Data Utility or issued interactively. They are executable statements that can be dynamically prepared.

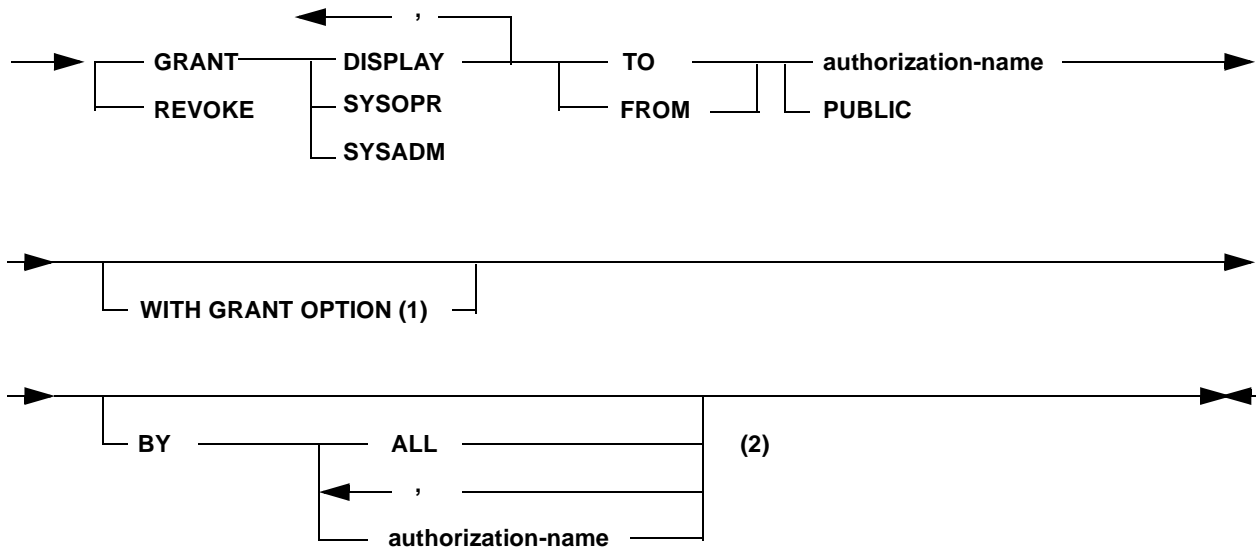
A GRANT is the granting of a specific privilege by a specific grantor to a specific user. By the same token, a REVOKE is the revoking of specific privileges by a specific grantor from a specific user, with the restriction that a privilege can only be REVOKEd if it has first been GRANTEd.

## System Privileges

System privileges allow or deny access to a specific set of catalogs within a Server. The syntax for a GRANT or REVOKE of user types for system privileges is shown in the figure that follows.



Figure 10: GRANT/REVOKE System Privileges Syntax Diagram



(1) GRANT ONLY

(2) REVOKE ONLY (only revokes privileges granted by that user)

The statements and their descriptions are included in the table that follows.

Table 2: GRANT/REVOKE System Privileges Syntax Statement Descriptions

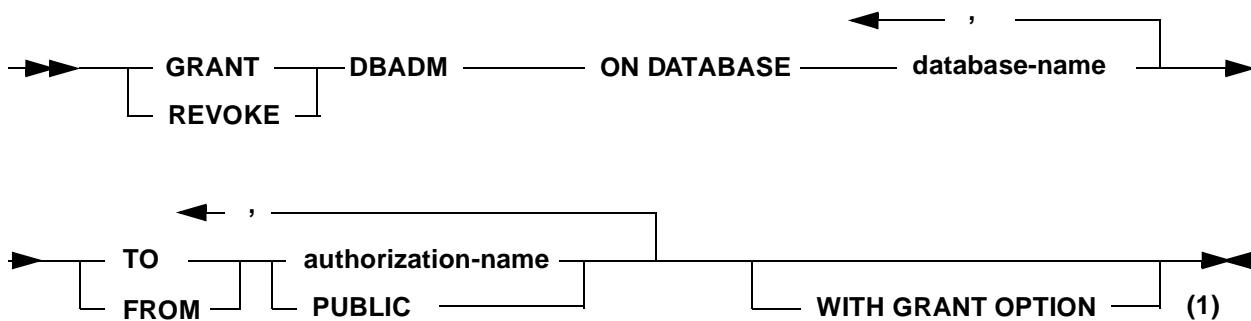
Statement	Description
GRANT	GRANT privileges to user IDs or groups of user IDs.
REVOKE	REVOKE privileges from user IDs or groups of user IDs.
DISPLAY	GRANT/REVOKE the privileges to remotely issue all forms of the DISPLAY command to a Server.
SYSOPR	GRANT/REVOKE the privilege to remotely issue all commands to a Server including the commands to start/stop services and shutdown the Server. <b>NOTE:</b> Commands issued from the system console are not secured.
SYSADM	GRANT/REVOKE system administrator authority.
TO <i>authorization-name</i>	GRANT authority to a particular user or group of users.
FROM <i>authorization-name</i>	REVOKE authority from a particular user or group of users.
PUBLIC	GRANT/REVOKE authority to all users or groups of users on a system.

**Table 2: GRANT/REVOKE System Privileges Syntax Statement Descriptions**

Statement	Description
WITH GRANT OPTION	<p>GRANT authority to a particular user or group of users to GRANT authority to other users or other groups of users in the system.</p> <p>Although this option can be specified when granting the SYSADM privilege, it has no effect on the SYSADM's privileges as the SYSADM privilege has ALL access privileges available in the Server.</p>
BY ALL authorization-name	<p>BY revokes each named privilege that was explicitly granted to some named user or group of users by one of the named grantors. Only an authorization ID with SYSADM authority can use BY, even if the authorization ID names only itself in the BY clause.</p> <p>ALL then revokes each named privilege from all named users or group of users.</p> <p><i>authorization-name</i> lists one or more authorization IDs of users or group of users who were the grantors of the privileges named.</p> <p>Do not use the same authorization ID more than once. Each grantor listed must have explicitly granted some named privilege to all named users or group of users.</p>

## Database Privileges

You can GRANT or REVOKE specific privileges within a particular database. The syntax for GRANTing/REVOKEing database privileges for the DBADM user type is shown in the following figure.

**Figure 11: GRANT/REVOKE Database Privileges Syntax Diagram**

The statements and their descriptions are included in the table that follows.

**Table 3: GRANT/REVOKE Database Privileges Syntax Statement Descriptions**

Statement	Description
GRANT DBADM	GRANT database administrator authority to a user.
REVOKE DBADM	REVOKE database administrator authority to a user.
GRANT ON DATABASE database-name	<p>Identifies database types on which privileges are to be GRANTED. For each named database types, the grantor must have all the specified privileges with the GRANT option. This privilege is used to secure the mappings of tables and dropping of mapped tables. The types are as follows:</p> <ul style="list-style-type: none"> <li>• \$ADABAS (for Adabas mappings),</li> <li>• \$CFI (for system catalog mappings)</li> <li>• \$DATACOM (for Datacom mappings),</li> <li>• \$IAM (for IAM mappings),</li> <li>• \$IDMS (for IDMS mappings),</li> <li>• \$IMS (for IMS mappings),</li> <li>• \$SEQUENT (for Sequential),</li> <li>• \$SP (for Stored Procedures), and</li> <li>• \$VSAM (for VSAM).</li> </ul>
REVOKE ON DATABASE database-name	<p>Identifies database type on which you are revoking the privileges. For each database type you identify, you (or the indicated grantors) must have granted at least one of the specified privileges on that database to all identified users (including PUBLIC, if specified). The same database type must not be identified more than once. The database-names are the same as those listed for the GRANT ON DATABASE database-name option.</p>
TO authorization-name PUBLIC	<p>Specifies to what authorization IDs the privileges are granted. <i>authorization-name</i> lists one or more authorization IDs.</p>
FROM authorization-name PUBLIC	<p>Specifies to what authorization IDs the privileges are revoked. <i>authorization-name</i> lists one or more authorization IDs.</p>

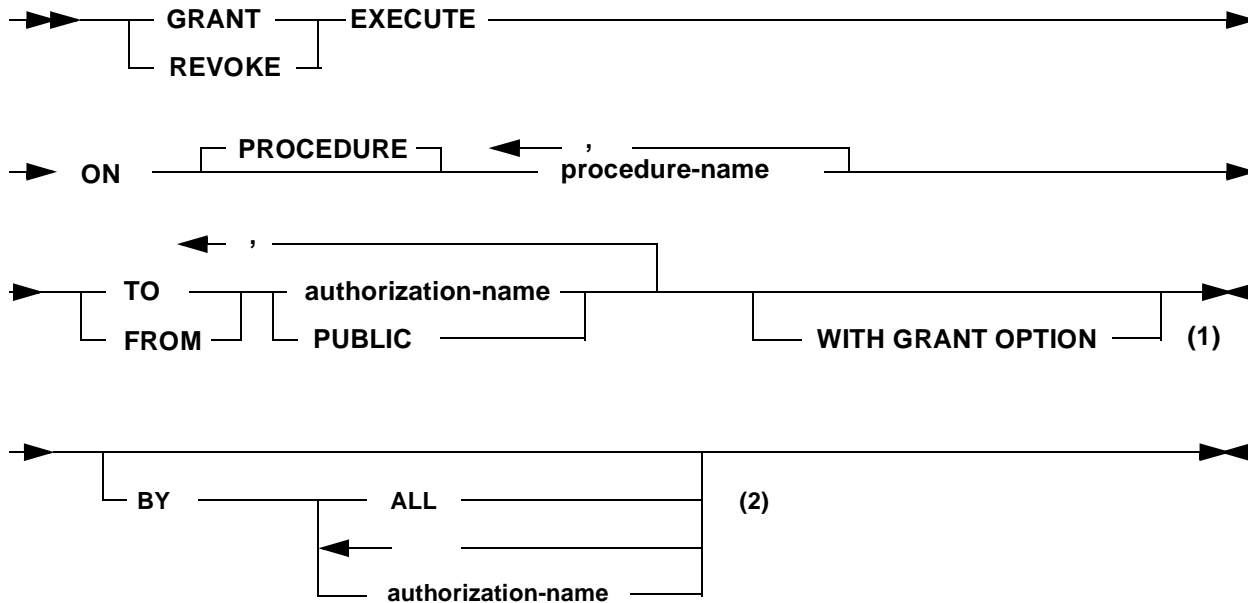
**Table 3: GRANT/REVOKE Database Privileges Syntax Statement Descriptions**

Statement	Description
WITH GRANT OPTION	Allows the named users to grant the database privileges to others. Granting an administrative authority with this option allows the user to specifically grant any privilege belonging to that authority. If you omit WITH GRANT OPTION, the named users cannot grant the privileges to others unless they have that authority from some other source.
BY ALL authorization-name	<p>BY revokes each named privilege that was explicitly granted to some named user or group of users by one of the named grantors. Only an authorization ID with SYSADM authority can use BY, even if the authorization ID names only itself in the BY clause.</p> <p>ALL then revokes each named privilege from all named users or group of users.</p> <p><i>authorization-name</i> lists one or more authorization IDs of users or group of users who were the grantors of the privileges named.</p> <p>Do not use the same authorization ID more than once. Each grantor listed must have explicitly granted some named privilege to all named users or group of users.</p>

## Stored Procedures Privileges

Stored Procedure privileges allow or deny access to a particular stored procedure. The syntax for a GRANT or REVOKE of user types for stored procedure privileges is shown in the figure that follows.

Figure 12: GRANT/REVOKE Stored Procedures Privileges Syntax Diagram



(1) GRANT ONLY

(2) REVOKE ONLY (only revokes privileges granted by that user)

The statements and descriptions are included in the table that follows.

Table 4: GRANT/REVOKE Stored Procedure Syntax Statement Descriptions

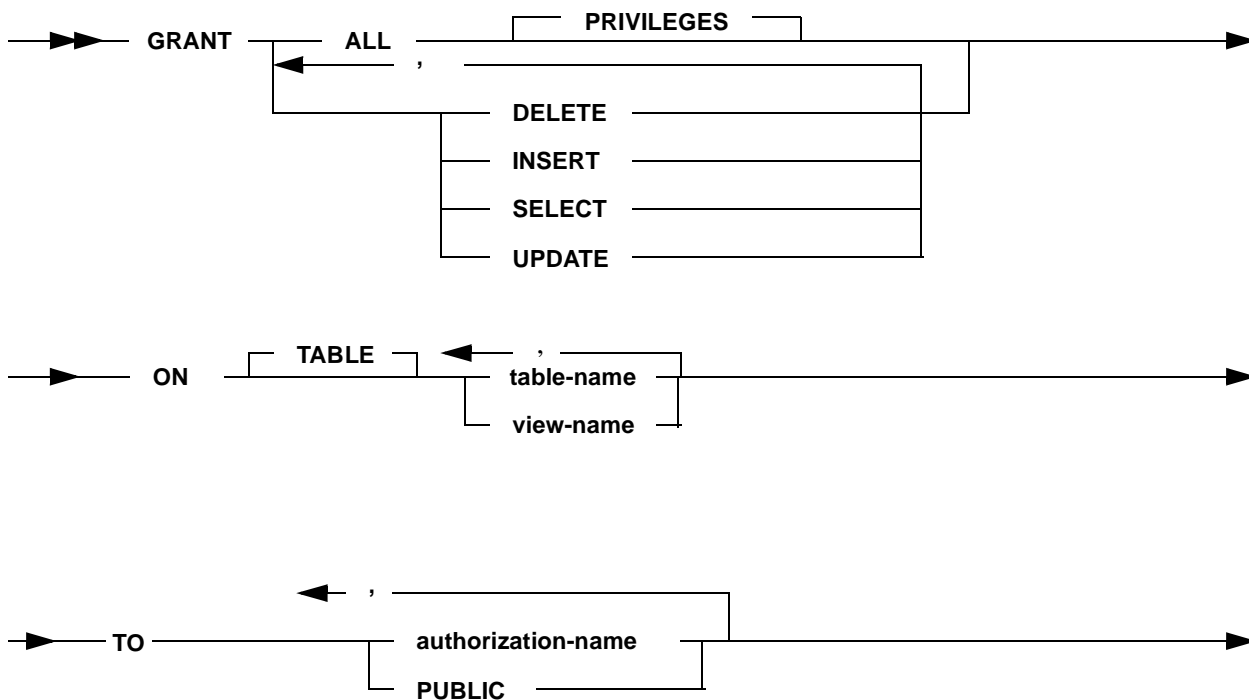
Statement	Description
GRANT EXECUTE	Grants authority to execute a stored procedure.
REVOKE EXECUTE	Revokes authority to execute a stored procedure.
ON PROCEDURE procedure-name	Identifies the procedure for which you are granting or revoking privileges.
TO authorization-name PUBLIC	Specifies to what authorization IDs the privileges are granted. <i>authorization-name</i> lists one or more authorization IDs.
FROM authorization-name PUBLIC	Specifies to what authorization IDs the privileges are revoked. <i>authorization-name</i> lists one or more authorization IDs.
WITH GRANT OPTION	Allows the named users to grant the stored procedure privileges to others. Granting an administrative authority with this option allows the user to specifically grant any privilege belonging to that authority. If you omit <code>WITH GRANT OPTION</code> , the named users cannot grant the privileges to others unless they have that authority from some other source.

## Table and View Privileges

Table and View privileges allow or deny access to specific tables and views.

The syntax for a GRANT of user types for table or view privileges is shown in [Figure 13: “GRANT Table and View Privileges Syntax Diagram,”](#) and described in [Table 5, “GRANT Table and View Syntax Statement Descriptions.”](#) The syntax for a REVOKE of user types for table or view privileges is shown in [Figure 14: “REVOKE Table and View Privileges Syntax Diagram,”](#) and described in [Table 6, “REVOKE Table and View Syntax Statement Descriptions.”](#)

**Figure 13: GRANT Table and View Privileges Syntax Diagram**



The statements and descriptions are included in the table that follows.

**NOTE:** ON table-name/view-name can be specified as a list of tables and views, separated by commas. The same is true for authorization name.

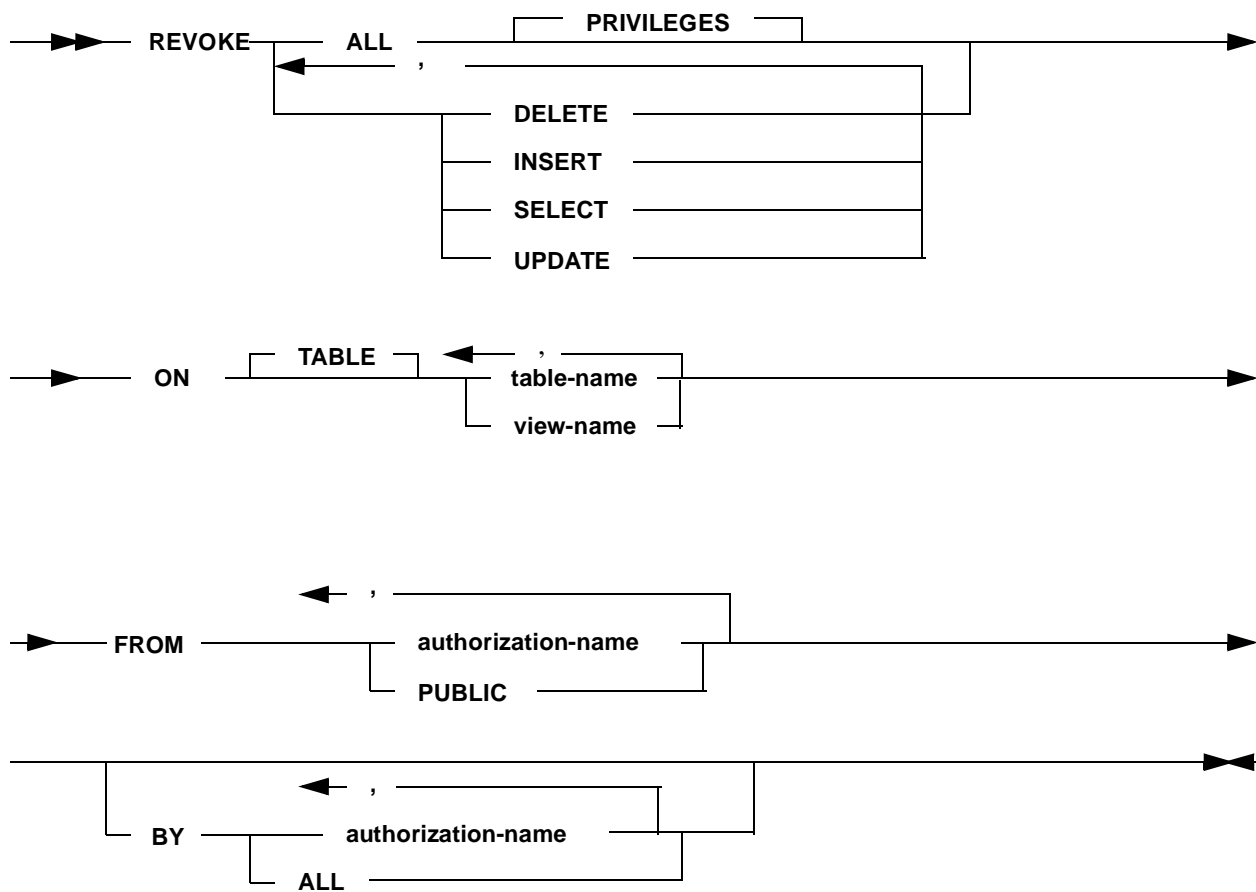
For example:

```
GRANT SELECT ON TABLE SYSIBM.SYSTABLES, SYSIBM.SYSCOLUMNS,
SYSIBM.SYSPROCEDURES TO USER1, USER2, USER3, USER4;
```

**Table 5: GRANT Table and View Syntax Statement Descriptions**

Statement	Description
GRANT ALL PRIVILEGES	Grants all table or view privileges for which you have GRANT authority, for the tables and views named in the ON clause.
DELETE	Grants privileges to use the DELETE statement.
INSERT	Grants privileges to use the INSERT statement.
SELECT	Grants privileges to use the SELECT statement.
UPDATE	Grants privileges to use the UPDATE statement.
ON TABLE table-name view-name	Names the tables or views on which you are granting the privileges. The list can be a list of table names or view names, or a combination of the two.
TO authorization-name PUBLIC	Specifies to what authorization IDs the privileges are granted. <i>authorization-name</i> lists one or more authorization IDs.
FROM authorization-name PUBLIC	Specifies to what authorization IDs the privileges are revoked. <i>authorization-name</i> lists one or more authorization IDs.
WITH GRANT OPTION	Allows the named users to grant the table/view privileges to others. Granting an administrative authority with this option allows the user to specifically grant any privilege belonging to that authority. If you omit WITH GRANT OPTION, the named users cannot grant the privileges to others unless they have that authority from some other source.

Figure 14: REVOKE Table and View Privileges Syntax Diagram



The statements and descriptions are included in the table that follows.

Table 6: REVOKE Table and View Syntax Statement Descriptions

Statement	Description
REVOKE ALL PRIVILEGES	Revokes all table or view privileges for which you have GRANT authority, for the tables and views named in the ON clause.
DELETE	Revokes privileges to use the DELETE statement.
INSERT	Revokes privileges to use the INSERT statement.
SELECT	Revokes privileges to use the SELECT statement.
UPDATE	Revokes privileges to use the UPDATE statement.
ON TABLE table-name view-name	Names the tables or views on which you are granting the privileges. The list can be a list of table names or view names, or a combination of the two.



**Table 6: REVOKE Table and View Syntax Statement Descriptions**

Statement	Description
FROM authorization-name PUBLIC	Specifies to what authorization IDs the privileges are revoked. <i>authorization-name</i> lists one or more authorization IDs.
BY ALL authorization-name	<p>BY revokes each named privilege that was explicitly granted to some named user or group of users by one of the named grantors. Only an authorization ID with SYSADM authority can use BY, even if the authorization ID names only itself in the BY clause.</p> <p>ALL then revokes each named privilege from all named users or group of users.</p> <p><i>authorization-name</i> lists one or more authorization IDs of users or group of users who were the grantors of the privileges named.</p> <p>Do not use the same authorization ID more than once. Each grantor listed must have explicitly granted some named privilege to all named users or group of users.</p>

## Authorization Requirements

In order to issue GRANT, REVOKE, SELECT, CALL, INSERT, UPDATE, or DELETE statements, users must have proper authorization. The following is a list of the user authorizations required to issue each of these statements.

**Table 7: Authorization Requirements for SQL Statements**

Statement	Authorization Required
GRANT	To GRANT a privilege, you must have SYSADM authority or you must have been granted the privilege itself with the WITH GRANT option.
REVOKE	To REVOKE a privilege, you must have SYSADM authority or be the user who originally granted the privilege being revoked. The BY ALL clause requires SYSADM authority because you are revoking privileges granted by users other than yourself.
SELECT	SYSADM authority or specific privilege is required; SELECT authority on all tables and views referenced in the SELECT statement.
CALL	SYSADM authority or specific privilege is required; EXECUTE authority on the procedure being called.

**Table 7: Authorization Requirements for SQL Statements**

Statement	Authorization Required
INSERT	SYSADM authority or specific privilege is required; INSERT authority on the Table being inserted into.
UPDATE	SYSADM authority or specific privilege is required; UPDATE authority on the Table being updated and SELECT authority on all tables reference in the WHERE clause.
DELETE	SYSADM authority or specific privilege is required; DELETE authority on the table in which rows are to be deleted; SELECT authority on all tables referenced in the WHERE clause.

## SQL Security and the eXadas SAF Exit

SQL Security is designed to be used in conjunction with the eXadas SAF security exit. SQL security only validates which user IDs have access to which catalog information. It does not validate that the user names are valid users of the system. It also does not validate user passwords. The SAF Exit is required to validate passwords at Server connection. Explicit resource checking between user names and system resources, such as VSAM files or IMS PSBs, can be disabled in the SAF Exit if SQL security is used.

Use both types of security to ensure that security is maintained. See [“Security: SAF Exit Specifics,” on page 220](#), for additional information about the SAF Exit and how it can be used to implement external security to validate user passwords.

Activation of the SAF Exit also allows you to administer SQL security at a group level as opposed to the individual user (ID) level. The SAF Exit has been modified to return a group name when it is called to perform initialization processing. This name, in addition to the user ID, is used to determine whether the user is authorized to perform the operation they are attempting.

The supplied SAF Exit returns the ACEEGRPN field from the ACEE created during initialization processing, if the ACEEGRPL field (in the ACEE) is not zero. The value in ACEEGRPN is the default group name for the user ID that was passed to the SAF Exit. You can customize this exit if you want it to return a different name.

Security administration is easier since you only need to GRANT privileges to the group name and all users associated with that group name automatically pick up those privileges. Likewise, to remove a privilege, just remove it from the group name and then all users associated with that group name also lose that privilege. When you need to authorize access to new users, use your external security package and assign the new user to the default group name.

Additionally, in this release of eXadas, the SMF Exit has been enhanced to be called from the Query Processor when an authorization violation is detected. The eXadas-supplied SMF Exit generates an SMF record that logs the user ID, type of authorization failure, and the name of the object for which authorization failed. These records use the same SMF record type as the standard accounting record generated by previous versions of the SMF Exit. The USRTYPE field contains the type of authorization violation and instead of containing the aggregate CPU time, a new field (OBJNAME) has been added to the record definition that identifies the name of the object for which authorization failed. You can modify the sample exit to generate a different type of SMF record if desired.

See [“Accounting: SMF Exit Specifics,” on page 228](#), for additional details on how the SMF Exit works.

## Summary

The SQL language is used to specify the security restrictions for an SQL-based database. The SQL security is built around privileges that can be granted to specific users and to particular database objects, like tables or views. Views can be used to restrict access to data for specific users. All of this is accomplished through GRANTing to or REVOKEing privileges from particular users or groups of users.



# 8

## Mapping Data

### Introduction to Mapping Data

This chapter discusses the following topics:

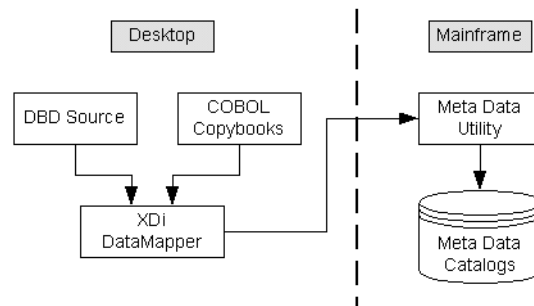
- [“The Data Mapping Process,”](#) on page 86,
- [“DataMapper,”](#) on page 87,
- [“Meta Data Utility,”](#) on page 88, and
- [“Advanced Mapping Considerations,”](#) on page 89.

**NOTE:** For information on why you would want to map data, see [Chapter 1, “Overview”](#) and [Chapter 2, “Concepts,”](#) in the *eXadas OS/390 Getting Started Guide*.

# The Data Mapping Process

The data mapping process involves gathering information about your existing database(s) and assigning SQL names to the data. The information is stored in a DB2-like system catalog where it can be queried in the same fashion as a DB2 catalog itself. The following figure shows the process of mapping data with eXadas.

**Figure 15: Mapping Data With eXadas**



**NOTE:** The DataMapper contains embedded FTP support to facilitate file transfer to and from the mainframe.

The eXadas DataMapper is a Windows application that takes information about your databases and generates Meta Data Grammar into an ASCII text file. The Meta Data Grammar is then processed by a Meta Data Utility to populate the DB2-like catalog files. The catalog files contain the DB2 system tables. These files are used by the server's Query Processor component to identify tables and columns referenced in SQL statements.

In addition to DB2 table and column information, these catalogs also store internal information such as OS/390 file names and record offsets necessary to convert data into SQL column data. These database-specific fields are not visible to the user in the catalog interface, as the DB2 catalogs themselves do not have column definitions for this type of information.

The Meta Data Utility is a program that runs on the server host and populates the DB2-like catalogs. The input to this utility is a text file containing grammar for defining DB2-like tables and columns and database-specific information necessary to successfully convert database fields into SQL columns. While it is possible to key in this grammar using any standard text editor, the process is extremely time-consuming and error prone. Therefore, it is recommended that you use the DataMapper application to create all Meta Data Grammar. This application can quickly convert COBOL copybooks into Meta Data Utility Grammar. Even if you do not have copybooks for your application, the DataMapper is a full function editor that will assist you to define tables and columns as well as to generate syntactically-correct Meta Data Grammar for input to the Meta Data Utility.

---

# DataMapper

This section describes the DataMapper utility and how to use the utility to map data to relational views.

## General Data Mapping

The eXadas DataMapper is a Microsoft Windows application that automates many of the tasks required to create a DB2-equivalent logical table for data structures. The DataMapper accomplishes this by creating Meta Data Grammar from existing data definitions (copybooks and DBDs). The Meta Data Grammar is used as input to the eXadas Meta Data Utilities to create Meta Data Catalogs that define how the data structure is mapped to an equivalent logical table. The Meta Data Catalogs are used by a server to facilitate translation of the data from the data structure into relational columns.

The DataMapper import utilities create initial logical tables from COBOL copybooks and IMS DBD source. You then use the DataMapper graphical user interface to refine these initial logical tables to create as many views of your data as your facility requires.

The DataMapper functions include:

- Creating data catalogs. A **data catalog** is a collection of tables for a particular database type, for example VSAM or IMS.
- Creating a table. A **table** is created by mapping one or more data structures from a data structure into a single eXadas table.
- Creating a column (optional). A **column** can represent one or more data items in the corresponding data structure. Columns can be defined manually or automatically created when importing a copybook.
- Importing a copybook. A copybook refers to a COBOL copybook that is transferred from the mainframe to the workstation for use by the DataMapper. Importing COBOL copybooks automatically creates column definitions.
- Loading DBDs for reference (IMS only). DBD refers to data base definitions that are transferred from the mainframe to the workstation for use by the DataMapper. This allows the DataMapper to use the information as reference when creating relational data. The DataMapper does not store IMS DBD source, so you must reload the source each time you reopen a repository.
- Generating Meta Data Grammar. **Meta Data Grammar**, also known as USE statements or USE Grammar, is generated by the DataMapper for all of the tables in a specific data catalog. When Meta Data Grammar has been created, it is subsequently transferred from the workstation to the mainframe. It is required as input to the eXadas Meta Data Utility that is run on the mainframe to create tables used as relational-equivalent data maps for their corresponding non-relational files.

- File transfer of data from the workstation to the mainframe. The DataMapper facilitates file transfers from the workstation to the mainframe through its built-in FTP facility. Transfer of copybooks or DBD source and generated USE Grammar can be performed.
- Creating a relational view. By transferring the Meta Data Grammar to the host system and running the Meta Data Utility with this input, you are able to create a relational view from non-relational data. This relational view is ultimately used by a server to enable SQL access to the mapped data structure.
- Creating an index. An eXadas index is a logical SQL index that maps an existing physical index on a target database or file system, for example IMS and VSAM. A DataMapper index is the mapping of an eXadas index.

The tutorial chapters of the *eXadas DataMapper Guide* include step-by-step information on how to use the DataMapper utility to map data to a relational view.

## Meta Data Utility

When the data mapping process is complete, Meta Data Grammar must be generated for input to the Meta Data Utility. This grammar is passed as input to the utility for updating the DB2-like Meta Data Catalogs on the system running the Server. Since the DataMapper repository stores all information gathered at mapping time, any changes or additions to the mapping can be easily made in the DataMapper and regenerated for updating the system catalog information.

The DataMapper generates Meta Data Utility Grammar for each defined data catalog separately. Meta data catalogs on a Server host can include tables defined for as many DataMapper data catalogs as desired. Each run of the Meta Data Utility only updates those tables for which Meta Data Grammar information is supplied. For example, if you have built the host Meta Data Catalog from 3 separate DataMapper data catalogs, you can update the tables for one of the data catalogs by changing the tables and regenerating the Meta Data Grammar for that data catalog. You then run the Meta Data Utility with the new generated output to update only those table definitions in the DataMapper data catalog. Table definitions contained in the host Meta Data Catalog from other DataMapper data catalogs will remain unchanged.

For more information on the Meta Data Utility, see [Chapter 13, “Utilities.”](#)



---

# Advanced Mapping Considerations

This section describes the considerations you should make prior to mapping databases into SQL tables and columns.

Before mapping any data, CrossAccess strongly recommends that you gather and fully understand the end-user application's requirements for data. As in building any end-user application, analysis is required to ensure the appropriate correlation between existing application data and the new uses you will be creating for the data. Given existing non-relational data, there are numerous ways to map data for the use of new applications. Attempts to map data simply by importing COBOL copybooks into the eXadas DataMapper application are likely to produce mappings that are unwieldy or totally unusable, depending upon the complexity of the copybooks.

This section includes the following topics:

- [“Defining Indexes,” on page 89,](#)
- [“Occurs Processing,” on page 92,](#) and
- [“IMS Segment Mapping Considerations,” on page 96.](#)

## Defining Indexes

Indexes will dramatically speed up queries that contain qualification information on indexed data. In most cases, indexes must be defined to eXadas during data mapping before they can be used by the server.

Most non-relational databases include support for indexing. eXadas also supports the definition of indexes in the DataMapper and in the generated Meta Data Grammar.

**NOTE:** eXadas does not actually index non-relational data, it only utilizes existing indexes on the data. Logical indexes defined in the DataMapper must match existing physical indexes placed on the underlying database system, itself.

After defining the columns for a logical table, you can add index definitions to the table to map existing indexes on the underlying non-relational data. Each logical index consists of an SQL name for the index and a list of one or more columns that comprise the index. Like table and column definitions, index definitions also have database-specific information required at definition time. This information varies based on the underlying database type.

### Multi-Part Keys

Like DB2, eXadas supports index definitions that are made up of multiple fields in the database record. These fields must first be mapped as logical SQL columns, which can then be referenced as index columns in an index definition. Generally, the Meta Data Utility automatically compares the column definitions, based on

their offset and length definitions, against the underlying database index to validate that the index definition exactly matches the database index.

In many cases, the columns comprising the index do not have to match one-for-one the database field definitions implementing the index. For example, if a 12 character database field is indexed, eXadas allows you to map the field as three CHAR(4) columns and then referencing these columns as an index. It is also possible to mix data types in the columns as long as the column definitions match the data type in the underlying database record. For example, a 12 byte index may be sub-defined as DECIMAL(7,2), followed by a CHAR(8) column, and then referenced in an index definition.

Optimization of queries using partial key information is automatically performed providing there is key information available for the first column referenced in the index definition. In cases where multiple indexes could potentially be used to optimize a query, precedence is given to any index that has qualification of the whole key over partial key information.

**NOTE:** Unlike DB2, eXadas supports re-mapping fields in a database multiple times. For example, an 8-byte character field can be mapped as a single 8-byte column and two 4-byte columns. If you re-map the columns that comprise an index, you must define a separate logical index definition for each re-mapping in order to ensure index optimization for all mappings.

## VSAM Indexes

Index definitions on VSAM files can be performed on the primary index of a KSDS and on alternate indexes for any VSAM file. When defining a VSAM alternate index, the PATH name for the index must be supplied in the Meta Data Grammar in order to process the index at query execution time.

## IMS Indexes

Index definitions for an IMS table must map either the primary index of HIDAM database or a secondary index on any IMS database. The key names in a primary index DBD cannot match the key names in a secondary index DBD. The mapped columns in an index are compared to the IMS FIELDS in a primary HIDAM index or XDFLD based on offset and length information to ensure that the index definition is valid. FIELDS in an IMS DBD can be subdivided into multiple column definitions as long as the columns map the complete underlying database FIELD and are listed in the index in the correct order.

For example, in the DBD:

```
DBD      NAME= . . .
DATASET DD1= . . .
SEGM     NAME=ROOT , PARENT=0 , BYTES=20 . . .
FIELD    NAME=FLD1 , TYPE=C , BYTES=3 , START=1
FIELD    NAME=FLD2 , TYPE=C , BYTES=7 , START=4
FIELD    NAME=FLD3 , TYPE=C , BYTES=10 , START=11
XDFLD    NAME=IDX1 , SRCH=( FLD3 , FLD2 ) . . .
. . .
```

The columns mapping DBD FIELDS FLD3 and FLD2 might be:

```

FLD2 SOURCE DEFINITION ENTRY ROOT
  DATAMAP OFFSET 3 LENGTH 7 DATATYPE C
  USE AS CHAR(7),
FLD31 SOURCE DEFINITION ENTRY ROOT
  DATAMAP OFFSET 10 LENGTH 3 DATATYPE C
  USE AS CHAR(3),
FLD32 SOURCE DEFINITION ENTRY ROOT
  DATAMAP OFFSET 13 LENGTH 3 DATATYPE UP
  USE AS DECIMAL(5),
FLD34 SOURCE DEFINITION ENTRY ROOT
  DATAMAP OFFSET 16 LENGTH 4 DATATYPE C
  USE AS CHAR(4)

```

The index definition for the XDFLD would be:

```

USE INDEX IMS_INDEX ON IMS_TABLE
( FLD31, FLD32, FLD33, FLD2 )

```

**WARNING:** Some IMS indexes contain a different number of index entries than there are actual target segments in the database. Mapping these indexes can result in a different number of rows being retrieved when accessed through the index, rather than through the primary database DBD. These indexes include sparse indexes and indexes whose source data is derived from segments other than the target segment of the index. To ensure a consistent result set whenever accessing the logical table, CrossAccess strongly recommends that *all* access to the logical table be through the specified index whenever you are mapping these types of indexes. The PCBPREFIX parameter on the table can be used to make all table access use a PCB with the correct secondary index PROCSEQ.

## IDMS Indexes

By default, the Meta Data Utility automatically creates index definitions for SYSTEM owned sets indexing the first record of an IDMS mapped table. If the fields listed in the KEY IS clause of the index set definition are mapped as columns in the table definition, then SQL queries referencing these fields will automatically use the index. However, if any explicit USE INDEX statements are defined for a table, only those explicitly-defined indexes are used for SQL access.

If sub-fields or group-level fields of those listed in the KEY IS clause are mapped, then you must include an explicitly USE UNIQUE INDEX definition in the mapping grammar for the SQL queries to index access using these fields.

For example, PART-NUMBER is an index defined as follows:

```

ADD
SET NAME IS ...
  OWNER IS SYSTEM
  MEMBER IS RECORD1
  KEY IS(PART-NUMBER ASCENDING)...

```

Its record layout definition is:

```

03 PART-NUMBER

```

```
05 PART-GROUP    PICXX.  
05 PART-NUM     PICX(5).
```

Its mapped columns include:

```
USE TABLE ...  
PART_NUMBER SOURCE DEFINITION ENTRY  
  RECORD1 PART-NUMBER  
  USE AS CHAR(7),  
PART_GROUP SOURCE DEFINITION ENTRY  
  RECORD1 PART-GROUP  
  USE AS CHAR(2),  
PART_NUM SOURCE DEFINITION ENTRY  
  RECORD1 PART-NUM  
  USE AS CHAR(5),  
.  
.  
.
```

By default, the following query will result in keyed access:

```
SELECT * FROM CAC.PARTS WHERE PART_NUMBER LIKE "01%";
```

The following query will not result in keyed access unless `PART_GROUP` and `PART_NUM` are explicitly defined as an index with a `USE INDEX` statement:

```
SELECT * FROM CAC.PARTS WHERE PART_GROUP = "01";
```

To ensure both index mappings can be used in SQL queries, the following `USE UNIQUE INDEX` definition must be defined when mapping the IDMS table:

```
USE UNIQUE INDEX index1 ON tablename (PART_NUMBER);  
USE UNIQUE INDEX index2 ON tablename (PART_GROUP, PART_NUM);
```

## Occurs Processing

Occurs processing for this release of eXadas is achieved by adding the following statements to the existing Meta Data Grammar.

- `BEGINLEVEL` starts the definition of a group of repeating fields.
- `ENDLEVEL` ends the definition of a group of repeating fields.

The begin and end level statement block column definitions that can occur more than one time in a data record. These groups of repeating fields are referred to as **record arrays**.

### Record Arrays

A group of data items in a database that have multiple occurrences within a single record in the database are referred to as **record arrays**. For example, a record could be defined for an employee to include the employee's dependent information (spouse and children) in the record. Since an employee may have

multiple dependents, you can declare an array of dependent information within the actual employee record by specifying a COBOL OCCURS clause. In the following example, an array of exactly 20 dependents has been declared in the record.

```

01  EMPLOYEE-RECORD.
05  EMP-LAST-NAME          PIC X(20).
05  EMP-FIRST-NAME        PIC X(20).
05  EMPSSN                PIC 9(9).
.....
05  DEPENDENTS-ARRAY OCCURS 20 TIMES
10  DEP-SSN               PIC 9(9).
10  DEP-NAME              PIC X(20).
10  DEP-DOB               PIC 9(6).
10  DEP-RELATIONSHIP-TO-EMPL PIC X.

```

Another common record array construct defines variably recurring data. The record contains up to 20 occurrences of the array data. The actual number of occurrences is dependent on the value of some other data item. A variably recurring dependent array is defined as follows:

```

01  EMPLOYEE-RECORD.
05  EMP-LAST-NAME PIC X(20).
05  EMP-FIRST-NAME PIC X(20).
05  EMP-SSN PIC 9(9).
.....
05  NUMBER-OF-DEPENDENTS PIC 9(4) COMP.
05  DEPENDENTS-ARRAY OCCURS 1 TO 20 TIMES
    DEPENDING ON NUMBER-OF-DEPENDENTS.
10  DEP-SSN PIC 9(9).
10  DEP-NAME PIC X(20).
10  DEP-DOB PIC 9(6).
10  DEP-GENDER PIC X.
.....

```

An example of the Meta Data Grammar that maps a subset of the data items in the preceding COBOL copybook is:

```

USE TABLE CAC.EMPL .....

(
  /* COBOL Name EMP-SSN */
  EMP_SSN SOURCE DEFINITION
    DATAMAP OFFSET 40 LENGTH 9 DATATYPE C /* Zoned Decimal */
    USE AS CHAR(9),
  /* COBOL Name NUMBER-OF-DEPENDENTS */
  NUMBER_OF_DEPENDENTS SOURCE DEFINITION
    DATAMAP OFFSET 49 LENGTH 2 DATATYPE H
    USE AS SMALLINT,
  BEGINLEVEL 1 OFFSET 51 LENGTH 36 OCCURS 20
    DEPENDING ON COLUMN NUMBER_OF_DEPENDENTS,
  /* COBOL Name DEP-SSN */
  DEP_SSN SOURCE DEFINITION
    DATAMAP OFFSET 0 LENGTH 9 DATATYPE C
    USE AS CHAR(9),
  /* COBOL Name DEP-NAME */
  DEP_NAME SOURCE DEFINITION
    DATAMAP OFFSET 9 LENGTH 20 DATATYPE C
    USE AS CHAR(20),
  ENDDLEVEL 1
)

```

)

The example only maps the employee's social security number field, the COBOL depending on variable, the dependent social security number field, and the dependent name field. The items in the OCCURS clause DEPENDENTS-ARRAY are contained within a BEGINLEVEL, ENDLEVEL block of statements. This identifies a group of data items that repeat. When converting this mapping from a COBOL record to SQL columns, eXadas takes each occurrence of the DEPENDENT-ARRAY and combines it with the non-array data items to create SQL rows. In this case, each occurrence of the array data items DEP-SSN and DEP-NAME is combined with the non-array data items EMP-SSN and NUMBER-OF-DEPENDENTS.

For example, if the record in the database for the employee SSN '123456789' contains three dependents, three distinct rows are returned for that particular record.

The query:

```
SELECT EMP_SSN, NUMBER_OF_DEPENDENTS, DEP_SSN, DEP_NAME FROM
CAC.EMPL WHERE EMP_SSN = '123456789' ;
```

returns the result set:

EMP_SSN	NUMBER_OF_DEPENDENTS	DEP_SSN	DEP_NAME
123456789	3	111223333	Depen1
123456789	3	222334444	Depen2
123456789	3	333445555	Depen3

eXadas does not support nested OCCURS DEPENDING ON definitions. In addition, you cannot define a table that contains fixed columns after an OCCURS DEPENDING ON construct.

Record arrays containing a fixed number of occurrences can include a NULL IS definition, which results in array occurrences being skipped as SQL ROW candidates at runtime. The NULL IS definition identifies a comparison value for the array itself or a column in the array that identifies an instance of the array as NULL. NULL instances of a record array are not returned as a row in the result set unless ALL instances of the array are NULL. If all instances of the array are NULL, then eXadas returns a single row for the non-array information in the record and the array data items, each set to NULL.

### Multiple Record Arrays in a Single Database Record

If the record layout for a database contains multiple record arrays, these arrays can all be mapped into a single logical table providing they are not defined as COBOL OCCURS DEPENDING ON clauses. However, before mapping multiple arrays in a single logical table definition, it is important to understand the implication of multiple arrays on the result set.

Returning to the example in the previous topic, assume that the employee record layout includes a second OCCURS clause as follows:

```

01  EMPLOYEE-RECORD.
    . . .
    05  DEPENDENTS-ARRAY OCCURS 20 TIMES.
    . . .
    05  HOBBIES-ARRAY OCCURS 5 TIMES PIC X(10).

```

If you use the DataMapper COBOL copybook Import facility to create a table definition for this record layout, it will, by default, map both the DEPENDENTS-ARRAY and HOBBIES-ARRAY into the same table.

Generally, this will not be the desired mapping as now the result set for a single record for the database will be the Cartesian product of the dependents array and the hobbies array, which is 100 rows for each database record.

In this case, it makes more sense to create multiple tables from the single record layout. Each table would include either the DEPENDENTS-ARRAY, or the HOBBIES-ARRAY, or possibly neither array if only employee information is desired.

## Record Typing

Many COBOL record layouts contain layout information to map different record types within a single physical file. In these cases, converting these layouts to table definitions is not as simple as importing the COBOL copybook into the DataMapper. Whenever record types are defined in a copybook, create a separate table for each value the record type field can contain and import only the COBOL fields associated with a specified value into each table.

Consider the following COBOL record layout:

```

01  DATABASE-RECORD.
    05  RECORD-KEY      PIC X(8) .
    05  RECORD-TYPE    PIC X.
        88  EMPLOYEE-RECORD  VALUE '1' .
        88  DEPENDENT-RECORD VALUE '2' .
        88  HOBBY-RECORD    VALUE '3' .
    05  EMPLOYEE-DATA.
        10  EMP-LAST-NAME    PIC X(20) .
        10  EMP-FIRST-NAME  PIC X(20) .
        10  EMP-SSN         PIC 9(9) .
    05  DEPENDENT-DATA REDEFINES EMPLOYEE-DATA.
        10  DEP-SSN         PIC 9(9) .
        10  DEP-NAME        PIC X(20) .
        10  DEP-DOB         PIC 9(6) .
        10  DEP-GENDER      PIC X.
    05  HOBBY-DATA REDEFINES EMPLOYEE-DATA.
        10  HOBBY-NAME      PIC X(10) .

```

The most logical mapping for this record layout is to create a logical table for each of the three discrete record types in the file (EMPLOYEE, DEPENDENT, and HOBBY). In each case, the DataMapper can assist in the mapping by importing specific group level items EMPLOYEE-DATA, DEPENDENT-DATA, and HOBBY-DATA, rather than the complete copybook. For more information on

importing selected group items from a COBOL copybook, see the *eXadas DataMapper Guide* or its associated Help File.

**WARNING:** You must ensure that the WHERE clause only selects the proper record type for each table mapped. The following examples show different ways to retrieve Employee data:

```
SELECT * FROM CAC.EMPLOYEE WHERE RECORD_TYPE='1';
```

Alternately, a view could be defined for this table by using the following statement in the Meta Data Utility:

```
CREATE VIEW CAC.EMPLOYEE_INFO AS SELECT * FROM CAC.EMPLOYEE  
WHERE RECORD_TYPE = '1';
```

The following query could then be used to retrieve the data:

```
SELECT * FROM CAC.EMPLOYEE_INFO;
```

## IMS Segment Mapping Considerations

Each logical table that maps an IMS database represents a path of one or more segments in the database. The path is defined by a leaf segment and all segments up the parent chain to the root segment of the hierarchy. The selected leaf segment can be any segment in the DBD.

Columns are mapped from segments in the selected hierarchy. They can be mapped from any or all segments in the selected path. When converting the IMS path to SQL rows, eXadas combines all occurrences of child segments with their respective parent occurrences as a separate row in the result set. For example, in a two-level hierarchy where a second level child is selected as the leaf segment for the table, assuming the first root in the database has 3 child occurrences, three distinct rows are returned with the root segment information duplicated with each occurrence of the child information returned. If a root segment exists with no child occurrences, a single row is returned containing the root segment information and null columns for those columns that map child segment data.

When mapping tables using IMS secondary indexes, IMS inverts the hierarchy if the index is not on the root segment. The index segment becomes the implied root of the new IMS hierarchy. In this case, all access to the defined table must be through the selected secondary index. This requirement ensures a consistent hierarchical view of the IMS database. The hierarchical view of the database is determined at Meta Data Utility runtime based on the index defined for the logical table. If no index is defined, access is assumed to be through the physical hierarchical view as defined in the IMS DBD. For more information on IMS secondary indexes and database inversion behavior with secondary indexes, see the appropriate IBM IMS manuals.

**NOTE:** See [Chapter 9, “Optimization,”](#) for additional information on how mapping non-relational data affects query optimization.



# Optimization

## Introduction to Optimization

Like any database system, eXadas performance varies based on the type of data and how the data is accessed. The single most important task with regard to performance is ensuring that the underlying data has been mapped correctly. This includes assuring that any indexes, keys, and/or units of data that are defined to the underlying database or file system are defined to the server when mapping the data. All other optimization built into the server is based on the assumption that these tasks have been performed to their fullest extent.

Based upon the above requirements, query optimization varies based upon how the eXadas Enterprise Servers, Servers, and Client applications are tuned. Most important is how the Query Processor SIEs are tuned. See [Chapter 2, “Deploying Applications,”](#) for more information on tuning server and application components including the Query Processor service.

This chapter covers some of the table definition and query writing techniques that can help ensure the best performance from eXadas. The following topics are discussed:

- [“Query Optimization,”](#) on page 98,
- [“JOIN Optimization,”](#) on page 99,

- [“Query Processor Optimization,”](#) on page 101,
- [“IMS Data Access Optimization,”](#) on page 103,
- [“VSAM Data Access Optimization,”](#) on page 112,
- [“VSAM Query Processor Optimizations,”](#) on page 115, and
- [“Server Execution,”](#) on page 115.

## Query Optimization

The performance of SQL queries can vary based on how the queries themselves are written. This is true of most relational databases and eXadas as well. The eXadas Query Processor optimizes queries on underlying data. However, this optimization is limited to the information it has about the underlying databases and the organization of the databases themselves.

eXadas query optimization is based on the extent to which the filtering required to obtain the final result set will be performed by the underlying database/file system. For example, take a three segment IMS HIDAM DBD that contains 10,000 instances of the lowest level segment (also called the **leaf segment**). If the Query Processor can build an SSA containing a search argument for every segment, then only a single access is required to retrieve the final result set, without the requirement of further filtering by the Data Savant or the Query Processor. If no SSA can be built, then 10,000 IMS GET commands will be issued; then that intermediate result set must be filtered by the Data Savant and/or the Query Processor to obtain that single row result set. This full retrieval of the mapped segments (or an entire VSAM file) is referred to as a **full table scan**. The IMS retrieval of the single row for the SSA example will be even faster if the SSA contains primary or secondary index fields. The scenario is the same for VSAM access, involving primary and alternate indexes as opposed to SSAs.

**NOTE:** Information regarding the filtering utilized to obtain the final result set is written to the log if the trace level field of the Query Processor SERVICE INFO ENTRY (for IMS, it is the IMS SERVICE INFO ENTRY) is set to 2. For more information about logging, see [Chapter 12, “Server Logging.”](#)

The remainder of this section provides tips for writing effective queries and how to define logical tables to eXadas to obtain optimum query performance.

## Using Keys

Whenever possible, keys should be passed in queries to the database(s) to be accessed. Keys are identified to eXadas by the inclusion of index definitions on logical tables. Defining index information often allows front-end tools that access

eXadas to create optimized keyed access queries based on the SYSIBM.SYSINDEXES and SYSIBM.SYSKEYS information stored in the eXadas Meta Data Catalogs. For more information on how to create an Index see [Chapter 8, “Mapping Data.”](#)

**NOTE:** Index definitions to eXadas do not actually index the underlying data, they merely describe existing underlying physical index(s) on the data.

When queries are performed without qualifying WHERE clause information on those indexes, nor any qualifying information on non-indexed columns, eXadas must scan the entire logical table in order to process the query. To perform this scan, eXadas must read the entire portion of the database/file that the involved table(s) define. The scanning process can result in poor performance on large databases. This is particularly true in cases where an SQL join is performed and an inner table must be scanned multiple times. See [“JOIN Optimization,” on page 99](#), for tips on how to optimize queries that contain joins.

**NOTE:** Because eXadas adds a new method of accessing existing data, it may be necessary to add additional indexes to your underlying data to meet the needs of certain queries. Review the types of queries you expect to be issued to determine whether new indexes can improve performance without impacting any other application’s use of the data.

## JOIN Optimization

This section describes the methods used to optimize SQL JOINS. For this discussion, JOIN optimization refers only to JOINS that are passed to the client without previous decomposition into single table queries at the server level.

JOIN processing utilizes the nested loop access method for all queries containing joined columns. Nested loop processing for a two table JOIN involves reading an outer table and, for each row selected in that table, reading the inner table to JOIN with that row, based on the JOIN column(s). This processing involves creating a row in the result table with the requested columns of both inner and outer tables. This processing applies to every row of the inner table where the value of its JOIN column(s) satisfy the specified relational conditions with the correlating JOIN column(s) of the selected row of the outer table. The next row of the outer table is selected and this process is repeated until there are no more rows to be selected from the outer table.

For example, the query:

```
SELECT A.COL_A1, B.COL_B1 FROM TABLE1 A, TABLE2 B
```

will by default read all rows in TABLE1, and for each row in TABLE1, read all rows in TABLE2 to join every row in TABLE1 with every row in TABLE2. If

TABLE1 has 100 rows and TABLE2 has 100 rows, the result set contains 10,000 rows for this query.

The number of rows is calculated as:

```
(number of rows in TABLE1) x (number of rows in TABLE2)
(100) x (100) = 10,000
```

The actual number of database reads is 10,100 (100 for the outer table and 10,000 (100 x 100) for the inner table).

While this sort of Cartesian join is not commonly performed in SQL queries, it does show what can happen in worst case scenarios for JOIN queries.

For example, if we change the previous query to:

```
SELECT A.COL_A1, B.COL_B1 FROM TABLE1 A, TABLE2 B
WHERE A.COL_A1 = B.COL_B1
```

the new query may only return a handful of rows. However, because there are no other qualifications for TABLE1 nor TABLE2, nested loop JOIN processing may still require reading every row from TABLE1 and matching it with every row of TABLE2 (10,100 reads). This is especially true if the JOIN column in the inner table (COL\_B1 in TABLE2) is not indexed.

If COL\_B1 in TABLE2 is a uniquely indexed column, the number of reads necessary is reduced in the previous query from 10,100 to 200 by reading each row in the outer table (TABLE1) once and doing a single indexed read to the inner table (TABLE2).

However, even with a unique index on COL\_B1 in TABLE2, if the outer and inner tables are switched as follows:

```
SELECT B.COL_B1, A.COL_A1 FROM TABLE2 B, TABLE1 A
WHERE B.COL_B1 = A.COL_A1
```

and the column COL\_A1 in TABLE1 is not indexed, then 10,100 rows are read. This is because the inner table (which is now TABLE1) must be scanned for each row in the outer table TABLE2. Again, this assumes COL\_A1 is not an indexed column in TABLE1 whereas COL\_B1 in TABLE2 is indexed.

When eXadas receives a JOIN table, ordering optimization is based on information in the WHERE clause. This optimization is attempted in two phases.

In the first phase of optimization, the optimizer checks to see if the default outer table in the JOIN has any WHERE clause information that is not part of a JOIN condition with a column in another table. If there is no non-JOIN WHERE information, the optimizer will check the remaining tables referenced in the query and use the first table with non-JOIN WHERE information as the outer table in the JOIN.

If Phase 1 of the optimization reorders the default outer-inner table processing of the JOIN and there are more than two tables referenced in the JOIN, Phase 2 of the optimization is run. In Phase 2, the optimizer attempts to order all of the inner

---

tables of the JOIN such that each inner table processed contains a column joined with a table that precedes it in the outer to inner order. This second phase helps ensure that filtering of rows from outer to inner table processing is maximized such that the number of access reads is kept to a minimum.

## Query Processor Optimization

The Query Processor executes SQL queries. The Meta Data Catalogs are accessed during Query Processing. During execution, the Query Processor calls database specific components, referred as Data Savants. A Data Savant is responsible for accessing the physical database/file associated with a logical table referenced in the query. The Data Savant uses the information in the WHERE clause to attempt to access the database/file in an optimum fashion.

The data returned from the Data Savant is then inspected to determine whether it meets the WHERE qualification, if any was supplied. The data that remains is then staged. Once all the data has been read, and staged, any post-processing that is required is then performed and those results are staged. Once the Query Processor has completed processing, the staged result set is returned to the client application.

The Data Savants automatically optimize Query Processor performance based on index information. Even if the Data Savant cannot perform optimized access to the database/file, the Data Savant still attempts to filter the number of records returned to the Query Processor, based on analyzing the WHERE clause against the data returned from the database/file system. If the Data Savant can successfully filter the records that will be returned, it informs the Query Processor. The corresponding instructions within the Query Processor that would have filtered the returned data are then skipped.

The Query Processor supports several different techniques for optimizing execution and for returning the result set data to the client application. Whether these optimization strategies work depends on the query that is issued and the size of the expected result set. Therefore, eXadas has made activation of these different optimization strategies optional by using configuration parameters to activate them. The different optimization strategies available are:

- Immediate return of data (PDQ),
- Static catalog Access, and
- Result set staging.

## Immediate Return of Data

The Immediate Return of Data option attempts to eliminate the intermediate result set staging process in the Query Processor. When active, a row of the result set is returned to the client application as soon as it has been returned from a Data Savant. Immediate Return of Data is activated using the PDQ configuration parameter. For syntax information see [“PDQ,” on page 396](#).

When active, the query is inspected to determine whether staging is required. If staging is not required, immediate return of data processing is invoked. Otherwise, normal Query Processor execution is performed. The inclusion of an ORDER BY clause requires the rows to be staged, thus preventing the immediate return of data.

Using Immediate Return of Data dramatically improves query performance for medium to large result sets. For small result sets (less than a single fetch buffer) activation of immediate return of data processing does not significantly improve query performance.

## Static Catalogs

Identifying the Meta Data Catalogs accessed by the Query Processor as being static optimizes the Query Processor compilation process. Usually, the Query Processor opens and closes the Meta Data Catalogs for each logical table referenced in a query during the Query Processor compilation process. Additionally, ENQUEUE READ locks must be established on the Meta Data Catalogs to ensure that a Meta Data Utility does not update the catalogs while they are being accessed by a Query Processor.

When you indicate that the catalogs are static, the Meta Data Catalog files are opened once when a user issues their first query and then are closed when the user disconnects from the Query Processor. Identification that the catalogs are static is performed using the STATIC CATALOGS configuration parameter. For syntax information see [Appendix A, “Configuration Parameters.”](#)

Identifying that the catalogs are static dramatically increases query performance when client applications issue several queries that return small result sets.

**NOTE:** Once catalogs are defined as static, eXadas does not allow updates from any source while the Server is active.

You can further optimize by creating linear versions of the Meta Data Catalog files. Linear catalogs are created and accessed in memory rather than via disk. eXadas provides a sample JCL member, CACLCAT, in the SCACSAMP dataset to create linear catalogs.

---

## Result Set Staging

If you do not activate immediate return of data processing, or a query cannot have immediate return of data processing applied to it, a query's result set is staged before it is returned to the client application. Most of the columns in a result set row are written to a B-tree file.

eXadas includes a set of configuration parameters that allow you to trade off between using virtual storage and DASD resources to stage data. Configuration parameters are supplied to control the number of B-tree buffer caches that are used, and the physical type, size, and placement of the temporary DASD storage that is used after all memory cache is exhausted. See [Appendix A, "Configuration Parameters,"](#) for additional information on the following parameters used to process result set staging.

- BTREE BUFFERS,
- FETCH BUFFER SIZE
- LD TEMP SPACE.

## IMS Data Access Optimization

This section describes how to optimize access to IMS data. The following topics are discussed in this section:

- ["General Guidelines," on page 103](#), discusses how to get the best performance using IMS indexes, keys and search fields.
- ["PCB Selection Options," on page 108](#), discusses the options that you have in the methods that eXadas uses to select a PCB to access a logical table's IMS data.
- ["PSB Scheduling," on page 109](#).

### General Guidelines

This section describes some general guidelines for optimizing access to your IMS databases. This optimization relies on the keyed access techniques that are available within IMS.

#### IMS Native Access

For optimum performance, the columns referenced in a WHERE clause should supply a key value or multiple key values. The same holds true in JOIN situations,

if the inner tables in a JOIN have key information so that a qualified SSA call can be used to optimize access. A qualified SSA is only generated (for either condition) when index information is available in the eXadas catalogs for the logical table(s) referenced in an SQL statement.

eXadas includes several options for optimizing access to the underlying database. It relies on the IMS index and SSA features to optimize access to IMS data whenever possible. Guidelines for using primary indexes, secondary indexes, and search fields are discussed in the next section.

## Using Primary Indexes

Defining columns that map to the primary index field in an IMS database will optimize queries that contain WHERE and JOIN qualification on those columns. eXadas creates qualified SSAs when a query contains the key values in the WHERE clause. This is also true in a JOIN situation when the tables from the outer loops supply a unique key for the inner loop sub-queries.

**WARNING:** HDAM databases do not have a primary index. They use a key-hashing technique to gain fast access to data. Logical tables mapped to an HDAM database might not necessarily be retrieved in ascending key sequence because the key sequence is a function of the HDAM randomizer, which is an IMS user exit. It is possible to order keys in an HDAM database either by specifying an ORDER BY clause in the SELECT statement or by specifying the name of a column that maps an XDFLD whose source is the HDAM primary key.

## Using Secondary Indexes

eXadas automatically uses secondary indexes when they are defined for a table with the USE INDEX statement. The query optimizer automatically optimizes using secondary indexes based on WHERE information supplied in each query. In cases where both a primary sequence field and secondary index are provided to access IMS data, the primary sequence field is given precedence.

**WARNING:** If an IMS secondary index is defined using a USE INDEX statement, the PSB used to access the target table must contain one or more PCBs with the correct processing sequence (PROCSEQ). If the PSB does not contain one or more proper PCBs, a runtime error will occur whenever the Query Processor selects the mapped index for keyed processing.

## Defining IMS Indexes With the USE INDEX Statement

You can use the USE INDEX statement to define the primary sequence field on a HIDAM database or XDFLDs for either HIDAM or HDAM databases. This statement may not be used to define the sequence field on an HDAM database. There are specific limitations on keyed access in HDAM that could mislead optimization algorithms if the primary sequence field was defined as an index to eXadas.

**NOTE:** HDAM keyed access will still be used whenever possible even though it cannot be defined with the USE INDEX statement.



When defining an IMS index, the columns specified in the USE INDEX statement must match either the sequence field for a HIDAM root segment or the SRCH fields defined on an XDFLD statement in the DBD. The Meta Data Utility validates all USE INDEX statements against the DBD by matching up column offset and length information against the offset and length information defined for sequence and SRCH fields within the DBD. This matching is only performed against the root segment for the defined table, as specified by the INDEXROOT in the actual USETABLE Statement (defaults to the physical ROOT if no INDEXROOT is specified).

The column(s) specified in a USE INDEX statement may sub-define sequence or SRCH fields in the DBD, if desired. For example, a primary HIDAM sequence field defined as 8 bytes may be mapped as two 4-character columns.

**NOTE:** The order of the columns in a USE INDEX definition is significant to the matching process. Columns that match sequence or SRCH fields but are in the incorrect order will be flagged as an error in the Meta Data Utility.

## Using Search Fields

eXadas will also use an IMS search field to optimize access to your IMS data. If you define a search field as a column in the Meta Data Grammar and then reference that column in a WHERE clause, eXadas generates an SSA that contains the WHERE qualification for the search field.

eXadas does not distinguish between a search field associated with the primary key for the root segment of an IMS database, a search field associated with a key for a subordinate segment, or a search field that is defined for a non-key field. Therefore, it is highly recommended that the WHERE clause in a query contain a reference to the column(s) associated with the root segment primary index, or a secondary index that maps to the root segment of the database (when using a secondary index, the DBD hierarchy may have been inverted). For optimum performance, include WHERE qualification on subordinate segment key fields or normal search fields in the root or subordinate segments. This allows eXadas to build a single SSA that causes IMS to return only the segments that match the qualification in the WHERE clause, and thereby eliminate or minimize any Query Processor and Data Savant filtering required.

**NOTE:** For more information, see [“Advanced Mapping Considerations,”](#) on page 89.

## Partial Key Support

eXadas supports mapping multiple columns against an IMS field (this can be the primary key, a secondary index (XDFLD), or a normal search field). When multiple columns are mapped to a single IMS field and a WHERE clause references a sub-set of those column(s) mapped to that IMS field, then this is referred to as a **partial key situation**.

eXadas attempts to optimize access using the partial key information supplied in a WHERE clause. This is accomplished by generating a key range based on the parts of the key supplied in the WHERE clause. The key range will specify the lowest and highest key values based on the values given for the columns specified

in the WHERE clause. eXadas can only perform this optimization and generate SSA(s) when the partial keys are supplied in the sequence that they map to the IMS field, starting from the beginning of the field.

For example, if you have an IMS field (named FIELD1) that is 10 bytes long and you map three columns to the IMS field in the following order:

- COLUMN1 = bytes 1-3,
- COLUMN2 = bytes 4-5, and
- COLUMN3 = bytes 6-10.

and you issue the following query:

```
WHERE COLUMN1 = 'abc' and COLUMN2 = '00'
```

then eXadas generates the following SSA:

```
FIELD1 >= abc00(low values) & FIELD1 =< abc00(high values).
```

This causes IMS to only return the segments that match the WHERE qualification. However, if you only specify a WHERE clause that references COLUMN2 and/or COLUMN3, eXadas cannot generate an SSA and instead must retrieve all the mapped segments and must perform all filtering logic within the Query Processor and Data Savant.

## Path Calls

eXadas uses path calls to access databases whenever there are WHERE qualification criteria for segments at levels other than the root of the hierarchy. The number of DLI calls required in these cases can be significantly reduced by using path calls. In order to take advantage of this feature, PSBs used to access IMS databases must contain PCBs defined with a PROCOPT value including P.

For example:

```
PCB DBDNAME=dbdname , PROCOPT=GOTP
```

If you have questions regarding whether your specific PSBs support path calls, review the PSBs in question with your IMS database administrator.

## HDAM/HIDAM Access Considerations

eXadas IMS data access optimization primarily focuses on the HIDAM and HDAM types of DBDs. The previous topics in this section detailed how access to IMS data is optimized by passing SSAs to IMS whenever possible. This topic discusses the differences in optimization when accessing HIDAM versus HDAM databases and the reasons behind the HDAM limitations.

In the case of HIDAM databases, SSAs are built for WHERE clauses containing columns that map to FIELDS and:

- reads on fields containing EQ, LT, LE, GT, GE operators with any combination of AND and OR conditions against any combination of mapped segments in the database.

Queries are optimized against HDAM databases for:

- simple keyed reads with the EQ operator and no AND or OR conditions for the key field.
- AND and OR conditions for other fields in any mapped segment as outlined for HIDAM databases.

**NOTE:** The WHERE clause cannot result in the request for a range of HDAM keys for optimization to take place. SSA support for HDAM databases is more restrictive due to the restrictions within IMS itself.

Full table scans may occur for many HDAM queries, as the default HDAM RANDOMIZER may not store keys in sequenced order. For example, a segment with a key field of 4 does not necessarily come sequentially after a segment with a key field of 3. So, while IMS allows OR conditions on an HDAM key qualification in the SSA, there are no guarantees that the correct result set will be returned from IMS. Rows that satisfy the WHERE clause might not be returned by IMS. This is a direct and documented limitation of IMS HDAM processing.

Therefore, the only way for eXadas to ensure that the correct result set is returned is:

- not to pass qualification information, thus forcing a full table scan.
- to access the database through a secondary index on the primary root key. The secondary index provides direct access to the correct result set, and is subject to the same optimization rules as a HIDAM secondary index.

For further information on HDAM processing, see the *IMS/ESA Application Programming: Database Manager Guide*.

## DEDB Considerations

IMS Fastpath DEDB databases can be accessed as either BMPs or by using the DRA interface provided with eXadas.

### ***High-Speed Sequential Processing***

When accessing DEDBs as a BMP service, high-speed sequential processing (HSSP) must be disabled in order to successfully process databases. PCBs are defined as HSSP when the PROCOPT keyword parameter includes the value H. To disable HSSP in a BMP, include a DFSCCTL JCL statement in the Server start-up JCL with a SETO control statement.

An example of the SETO statement is:

```
//DFSCCTL DD *
```

```
SETO DB=IVPDB3 , PCB=HSSP , NOPROCH
```

Failure to disable PROCOPT=H PCBs results in an FY status call when accessing the DEDB and an unexpected DLI status error messages.

See the *IBM IMS/ESA Installation Volume 2: System Definition and Tailoring* for more information about the SETO control statement.

### **Fast Path Buffers**

When running as either a BMP or using DRA Access, fast path buffers must be available to process DEDB databases. If IMS runs out of buffers during a query, an FR status code is returned while processing a DLI call. The eXadas client returns an unexpected status code error message.

To set the buffers in a BMP environment, the NBA and OBA keyword parameters may be set in the Server JCL for passing to IMS at BMP start-up time. For a description of these parameters, see the description of the IMSBATCH procedure in the *IBM IMS/ESA Installation Volume 2: System Definition and Tailoring* guide.

To set the buffers in the DRA environment, the CNBA and FPBxx parameters in the DFSPRP macro must be defined when generating the DRA start-up table used by the Server. For more information on the DRA startup table, see [“Setting Up the DRA for Use by eXadas,” on page 37](#).

## **PCB Selection Options**

This section describes the options available to determine how the query processor selects a PCB to access the segments mapped by a logical table or column. The query processor can use either of the following methods to select a PCB:

- PCB selection by verification or
- PCB selection by name.

Selection by name is the faster PCB selection option; however, it requires PSB/PCB definition work by the IMS DBA and careful coordination of its specification as well as the related table definitions within eXadas. An overview of the processing performed for each option is described in the sections that follow.

### **PCB Selection by Verification**

PCB selection by verification involves the Query Processor issuing DL/I calls to verify that the PCB selected can successfully access the database path that the logical table will access. PCB verification also requires the proper PROCSEQ to be specified if a column(s) that maps to an XDFLD is specified in a WHERE clause. This is the default method that the Query Processor uses to select a PCB for processing.

## PCB Selection by Name

PCB selection by name, unlike PCB selection by verification, does not issue DL/I calls to verify the database path but selects the PCB by a partial PCB name associated with a logical table or index. The PCB name selection option is faster but takes more administrative work to set up. PCB name selection uses the AIBTDL/I interface to locate an appropriate PCB.

Activating the PCB selection by name option requires you to:

- Specify a PCBPREFIX parameter in your Meta Data Grammar. See [Chapter 8, “Mapping Data,”](#) for additional information. The following Meta Data Grammar elements support the PCB prefix:
  - Table source definitions and
  - Index source definitions.
- Create PCB names with the appropriate PCB prefix and sequence numbers.

**NOTE:** If an invalid PCB prefix name is specified, queries against the associated local table or column will fail with a negative SQLCODE returned to users and applications. A negative SQLCODE also is returned if all PCBs for a given prefix are in use.

The PCB prefix is a one-to-seven character alphanumeric string that conforms to IMS PCBNAME naming conventions. The specified value does not have to be unique. In fact, the same PCB prefix name should be used for all logical tables that access the same logical or physical DBD using the same primary or secondary index.

The PCB prefix is used to generate a unique PCB name by appending a sequence number to the prefix. The sequence number begins with zero.

For example, if you specify a PCB prefix of EMPL, the PCB names will be:

- EMPL0,
- EMPL1,
- EMPL2, and so on for the number of concurrent accesses that are required to the logical tables or columns associated with the EMPL PCB prefix.

**WARNING:** While PCB selection by name is faster than selection by verification, if the PCB selected does not have the correct IMS path access for the table (SENSEGs), an error is returned when attempting to query the table. Care must be used to ensure the named PCB has correct access for the defined table(s).

## PSB Scheduling

The PCBs whose selection was discussed in the previous topic are contained within PSBs. A **PSB** is the unit of access that a given program uses to interface with IMS. This interfacing process is called PSB scheduling. The processes

involved in scheduling a PSB for a Server, and how you specify which PSB a Server will use, differ according to which eXadas IMS interface is utilized.

The ASMTDLI interface allows only a single PSB to be scheduled per server instance, which is specified within the Server startup JCL. This is the interface that is used to allow a Server to access IMS in much the same manner as a CICS region accesses IMS.

For more information regarding other aspects of IMS interfaces eXadas offers, see [Chapter 3, “Server Setup for IMS Access.”](#)

## Using the ASMTDL/I Interface

The ASMTDL/I interface allows the Server instance to access IMS as a BMP, DBB, or DL/I region. To use this interface, an IMS region controller is loaded into the Server address space by activating the IMS BMP/DBB/DL/I Initialization Service Info Entry. The type of IMS region controller: BMP, DBB, or DL/I, that is activated is determined by parameter specification within the Server startup JCL, the same way a non-eXadas IMS region controller determines this information. See [Chapter 3, “Server Setup for IMS Access,”](#) for more information.

Access to IMS data using an ASMTDL/I interface is limited to a single PSB for an instance of a Server. This is because the IMS region controller only allows a single PSB to be scheduled within itself (or by itself), and has a limit of one instance of an IMS region controller per OS/390 address space.

If you:

- require access to multiple DBDs,
- plan on performing JOINS of IMS data, and/or
- plan on supporting multiple users,

then the PSB scheduled in this environment must contain enough PCBs to support all concurrent user access to the IMS data. Since very large PSBs require significant IMS resources, CrossAccess strongly recommends using the DRA interface to access IMS data, when possible. If this is not possible at your site, then it is strongly recommended that you configure Servers using the ASMTDL/I interface as single-user servers. This requires a smaller number of PCBs to be defined within the PSB, which results in a smaller PSB that requires significantly less resources for the Server instance. See [Chapter 3, “Server Setup for IMS Access,”](#) or [Chapter 17, “Enterprise Server,”](#) for more information about configuring Data and Enterprise Servers as single or multi-user servers.

---

## Using the DRA Interface

Data access to IMS using the DRA interface is the preferred IMS data access approach. DRA is the most scalable solution in that it allows the scheduling of multiple IMS PSBs from a single Server and can support multiple user connections at a time.

The ability to schedule multiple PSB(s) also allows the PSB(s) to be small in size and thus reduces the IMS resources required for a query against IMS data. This topic discusses the PSB scheduling options that you have when using the DRA interface.

The Meta Data Grammar for an IMS logical table allows for the association of two PSB(s) with each logical table. This is done on the SCHEDULEPSB parameter. The first PSB that you specify is referred to as the **standard PSB**, the second PSB is the **JOIN PSB**.

When a query is received that does not contain a JOIN between two or more IMS logical tables, the standard PSB is scheduled to service the query. This PSB only needs to contain a single PCB that has sensitivity to the segments that make up the IMS path that the IMS logical table maps.

When a query is received that contains a join between two or more IMS logical tables and a JOIN PSB is specified for the first table in the JOIN, eXadas schedules the JOIN PSB. The assumption is that the JOIN PSB will contain multiple PCB(s), at least one for each IMS logical table referenced in the JOIN.

When eXadas attempts to schedule a PSB for a second, third, or additional time to access the IMS logical table referenced in the JOIN, the PSB(s) that are already scheduled (to service the query) are inspected to see if they contain a PCB that can be used to service the query. If an available PCB is found, then eXadas does not schedule another PSB and instead uses the PCB that it has located to issue IMS calls for access to the referenced table. If a PCB cannot be located, and a JOIN PSB is specified for this table as well, eXadas schedules that JOIN PSB. If no JOIN PSB is specified, eXadas schedules the standard PSB associated with that logical table.

The DRA interface also allows a default PSB to be specified in the SERVICE INFO ENTRY parameter used to initialize the DRA interface. If an IMS logical table does not have a Standard PSB defined, then eXadas schedules the default PSB and inspects it to determine whether there is a PCB that can be used to service the query. If a default PSB is used, then it should contain PCB(s) for all IMS databases/segment paths/secondary indexes that need to be accessed. This is very similar to the type of PSB that is required if the ASMTDL/I interface is used to access your IMS data.

CrossAccess strongly recommends that you create standard PSB(s) for all IMS logical tables if you are going to use the DRA interface and you will be joining IMS data together with composite JOIN PSB(s) that can be used to eliminate the need to schedule multiple PSB(s) in JOIN situations.

**NOTE:** It is not necessary to create individual PSB(s) for each logical table. You can share the same standard or JOIN PSB among multiple logical tables. Additionally, if you are accessing multiple paths in the same IMS database using multiple logical tables, you may want to create a single PSB that contains one or more PCB(s) that maps all paths in the DBD to be accessed. You can then associate this PSB with all of the logical tables that are mapped to that database. This same recommendation also applies to all of the logical tables that are mapped to a DBD that will use the same secondary index.

## VSAM Data Access Optimization

This section describes how to optimize access to VSAM data. The following topics are discussed in this section:

- [“General Guidelines,” on page 112](#), provides tips for using primary and alternate indexes as well as partial keys to optimize VSAM access.
- [“VSAM Query Processor Optimizations,” on page 115](#), discusses the VSAM AMPARMS parameter that can be used to optimize VSAM access.
- [“VSAM Service,” on page 115](#), discusses how the VSAM service allows multiple concurrent users to share files that are already opened.

The optimization techniques discussed in this section also apply to IAM files. IAM (Innovation Access Method) is supplied by Innovation Data Processing and is a reliable, high-performance disk file manager that can be used in place of VSAM KSDS and ESDS data sets. Innovation Data Processing also provides an optional Alternate Index feature that is supported by eXadas. An exception is any reference to VSAM RRDS support, which currently is not supported by IAM.

### General Guidelines

This section describes general guidelines about optimizing access to your VSAM data. These techniques only work for VSAM KSDS data sets. ESDS and RRDS data sets require the entire contents of the files to be read to process a query, as there is no direct access to the data. The exception to this rule is accessing an ESDS through an alternate index. This type of access mimics accessing a KSDS using an alternate index. Any discussion in this section regarding the use of an alternate index refers to ESDS or KSDS alternate indexes.

The best way to optimize access is to utilize keyed access techniques available in VSAM. eXadas supports VSAM keyed access using either a primary index or an alternate index. Additionally, eXadas supports access using either of these index types when only part of the key is supplied. These different options are discussed in the sections that follow.



---

## Primary and Alternate Indexes

There are two types of VSAM indexes: **primary** and **alternate**. The query processor supports both primary and alternate indexes when processing SQL requests. Although these indexes are usually transparent to the end user, queries that exploit primary and/or alternate indexes can enjoy substantial improvements in performance when retrieving data from large VSAM files particularly during JOIN processing.

For a query to qualify for the optimizations described in [“Query Optimization,” on page 98](#), the following requirements must be met:

- a column(s) that maps to a primary or secondary index must be referenced in a WHERE clause or as a JOIN column, and
- the index that the columns map must be defined in the Meta Data Catalog.

The remainder of this section discusses how a VSAM index is defined to the Meta Data Catalog.

The **primary index** is determined automatically by the Meta Data Utility during the USE TABLE processing and stored in the eXadas system catalog for use by the Query Processor. No explicit definition using Meta Data Grammar is required.

**Alternate indexes** are defined by supplying an appropriate USE INDEX statement to the Meta Data Utility. Index definitions created by eXadas are stored in the SYSIBM.SYSINDEXES and SYSIBM.SYSKEYS tables.

**NOTE:** Index definitions to eXadas do not actually create an index of the underlying data. They merely describe an existing physical index to the data.

**WARNING:** If an index is defined on a field that is not an actual VSAM index, the query may behave unpredictably.

As part of the USE INDEX statement, either the data set name or the DD name of a DD pointing to the VSAM Alternate Index PATH data set must be specified. This information is stored in the eXadas system catalog for use by the Query Processor when accessing that alternate index.

### **Primary Indexes**

The utilization of primary indexes is determined by the query processor at execution time. This is based on the existence of index information that defines the columns making up the key and by supplying key values in the SQL WHERE clause. The query processor automatically performs keyed reads of the data set, if possible.

## Alternate Indexes

Alternate indexes can also be used by the Query Processor. An alternate index is used to satisfy an SQL query if the following conditions are met:

- A column in the WHERE clause maps to an alternate index field.
- The VSAM alternate index path data set or DD name that was supplied in the INDEX definition is accessible by the Server.

## Partial Key Support

eXadas supports mapping multiple columns against both primary and alternate indexes. When multiple columns are mapped to an index, and a WHERE clause references a sub-set of the column(s) mapped to the index, this is referred to as a **partial key situation**.

eXadas attempts to optimize VSAM access using the partial key information supplied in the WHERE clause. This is accomplished by generating a key range based on the parts of the key supplied in the WHERE clause. The key range will specify the lowest and highest key values based on the values provided for those columns specified in the WHERE clause. eXadas can only perform this optimization when the columns that make up the partial key are supplied in the sequence they are mapped to the index, starting from the beginning of the index.

For example, if you have a primary index that is 10 bytes long and you map three columns to the index in the following order:

- COLUMN1 = bytes 1-3,
- COLUMN2 = bytes 4-5, and
- COLUMN3 = bytes 6-10.

and you issue the following query:

```
WHERE COLUMN1 = 'abc' and COLUMN2 = '00'
```

then eXadas generates the following key: 'abc00x', padded to the right with low values (x'00') for the length of the key. This key is used for the initial read of the VSAM file. Processing of the VSAM file then continues sequentially with the key of each returned record being compared to a high key value that eXadas generated. In this example, the high key value would be 'abc00x' padded to the right with high values (x'FF') for the length of the key. The records are read until the returned record's key exceeds this generated value or the end of the file is reached.

**NOTE:** If you specify a WHERE clause that only references COLUMN2 and/or COLUMN3 without referencing COLUMN1, eXadas cannot generate a key range and instead must process the entire VSAM file sequentially and perform any record filtering within the Data Savant and/or QP.

---

## VSAM Query Processor Optimizations

The VSAM AMPARMS parameter is a Query Processor related configuration parameter that can be used to optimize access to your VSAM files. VSAM AMPARMS allows you to specify the number of in-core buffers used to cache VSAM index and data components and the overall amount of memory that VSAM can use for buffer caching. For more information about the syntax of this parameter see [Appendix A, “Configuration Parameters.”](#)

The VSAM AMPARMS parameter is used to specify VSAM buffering information for VSAM files. If a large number of records will be read, increasing the size of the VSAM data buffer pool can substantially improve query performance since more of the VSAM data can be cached in core.

## VSAM Service

The VSAM service allows multiple concurrent users to share files that are already opened via either local VSAM or VSAM through CICS. This reduces overhead as open and closes are done less often. This is particularly useful in join situations where a VSAM file is joined to itself.

To enable the VSAM service, uncomment the Service Info Entry in the Server configuration file. See [“SERVICE INFO ENTRY,” on page 398](#), for more information on the Service Info Entry.

## Server Execution

Executing the server at a proper dispatching priority can have a dramatic affect on query performance. Additionally, a Work Load Manager (WLM) sample system exit is provides that allows you to place the individual queries that your users issue into WLM goal mode. Each of these topics are discussed in the sections that follows.

## Dispatching Priority

The dispatching priority that a server executes in and the paging class that the server is assigned can dramatically improve performance. eXadas is very much like a transaction processing system, such as DB2, CICS, or IMS.

The lower the dispatching priority that the server runs in the better it will execute. CrossAccess recommends that the Server dispatching priority be set greater than the operating system and the TCP/IP sub-systems that are used by eXadas to communicate with remote clients. The dispatching priority should be set at or above the dispatching priorities of the CICS, IMS, and/or DB2 sub-systems that you have running at your OS/390 site. Consult your data center personnel to select an appropriate dispatching priority that does not adversely affect the other applications running at your OS/390 site.

## WLM Support

eXadas includes a sample Workload Manager (WLM) system exit that allows the individual queries that your users issue to be placed in WLM goal mode. Placing queries in goal mode allows WLM to control the amount of resources that are available for the query to use. eXadas WLM support uses enclave TCBs, which are available in OS/390 version 2, release 3, or higher.

The WLM exit allows you to place the users of a query processor into different service classes. Each service class can have different rules for the amount of resources that are assigned (by OS/390) to execute the query. This also allows OS/390 to apply period switching rules for each query that is executed. Typically, when running in goal mode using period switching, the longer a query runs the less resources it gets. This allows you to set up a service class for small queries that need to execute quickly (and will therefore be given more resources). For longer running queries you can assign these queries to a different DATASOURCE name, with a different service class that allows OS/390 to reduce the amount of resources these queries get the longer they run, without impacting the small high performance queries.

For more information on WLM support see [Chapter 15, “System Exits.”](#)

# 10

## Server Operations

### Introduction to Server Operations

This chapter provides a brief overview of running servers in a test and production environment. The following topics are covered:

- [“Starting Servers,” on page 118,](#)
- [“Monitoring and Controlling Servers,” on page 118,](#)
- [“Starting and Stopping Individual Services,” on page 124,](#) and
- [“Stopping the Server,” on page 125.](#)

For a list of MTO commands, see [Appendix C, “MTO Command Reference.”](#) For additional information on the Enterprise Server, see [Chapter 17, “Enterprise Server.”](#)

OS/390 Servers can be run as either an OS/390 batch job or as a started task. Like any batch job or started task, the user ID of the server itself is based on the submitter of the batch job or the task name of a started task. While authorization to access databases and files is based on the user ID of the client task, the server must have authority to access all STEPLIB files as well as the server configuration data set. In addition, if you want to make configuration changes while the server is running and then write the changes back to the configuration data set, the server must also have update authority for the configuration data set.

# Starting Servers

If you are not using an Enterprise Server, then each of your production servers must be pre-started before your client applications can be used. If you are using an Enterprise Server, then only the Enterprise Server must be pre-started. The Enterprise Server automatically starts Servers based on the DSH SERVICE INFO ENTRY parameter definitions in the Enterprise Server's Master Configuration member. The Enterprise Server can communicate with servers on other machines as well. For information on Enterprise Server Operations or configuration settings, see [Chapter 17, "Enterprise Server."](#)

Once a server is started, it is ready to accept client connections and requests for data.

# Monitoring and Controlling Servers

The servers and the eXadas Enterprise Server are designed to run continuously. In the current version, eXadas supplies an OS/390 MTO (Master Terminal Operator) interface that can be used to monitor and control Server/Enterprise Server operations.

Using the MTO interface you can issue commands to:

- display active services in a server,
- display users connected to a server,
- display configurations,
- modify configuration parameters,
- display memory utilization,
- **START** and **STOP** individual services, and
- **STOP** the Server.

The basic format of an OS/390 MTO command is:

```
F servername , command , [ subcommand1 , subcommand2 , . . . ]
```

When executing these commands under IBM's SDSF, you must precede the command with the slash (/) character, as shown in the following example:

```
/F CACDS , DISPLAY , SERVICES
```

For more information regarding specific MTO commands, see [Appendix C, "MTO Command Reference."](#)

## Displaying Active Services in a Server

At start-up, the server processes the Master configuration file and starts services defined with SERVICE INFO ENTRY parameter values. The number of services started depends on the minimum tasks entry (field 4) in each SERVICE INFO ENTRY.

**NOTE:** This field can be set to 0 when you do not want the service started automatically at Server initialization. For more information on the SERVICE INFO ENTRY parameter settings, see [Appendix A, “Configuration Parameters.”](#)

To display the active services, issue the **DISPLAY,SERVICES** command from the OS/390 Master Console or SDSF. An example of the output from the **DISPLAY,SERVICES** command is shown in the following example and a sample of the available services for the **DISPLAY** command are described in [Table 8, “Available Services With the DISPLAY Command.”](#)

```
F servername , DISPLAY , SERVICES
```

**Table 8: Available Services With the DISPLAY Command**

Service	Type	TASKID	TASKNAME	Status	User
LOGGER	CACLOG	9392624	CACLOG	READY	
IMSDRA	CACDRA	9272720	CACDRA	READY	
CACSAMP	CACQP	9270608	CACQP	READY	
CACSAMP	CACQP	9269664	CACQP	OPEN CURSOR	CACUSER5
TCPIP	CACINIT	9214680	CACINIT	READY	
DSN	CACCAF	9213312	CACCAF	READY	
DSN	CACCAF	9170568	CACCAF	READY	
DCOMIS	CACDCI	9275696	CACDCI	READY	

**NOTE:** The active service display only shows services that are currently running in the server. To view all services defined to a server, you must display the master configuration with the command **DISPLAY,CONFIG=MASTER**. For more information on displaying configurations, see [“Displaying Configurations,” on page 120.](#)

## Displaying Users Connected to a Server

Once a server is started, it is ready to accept user connects. These connections may come from local OS/390 clients; ODBC or JDBC clients; AIX clients; or any

other eXadas-supported platform. To determine if any connections exist, you can issue the **DISPLAY,USERS** command.

An example of the **DISPLAY,USERS** command follows. The sample output is described in [Table 9, “DISPLAY,USERS Command Output.”](#)

```
F servername , DISPLAY , USERS
```

**Table 9: DISPLAY,USERS Command Output**

USER	SESSIONID	HOST NAME	PROCESSID	THREADID	SERVICE	TASKID
CACUSER5	92115632	P390	74	9397120	CACSAMP	9269664
CACUSER1	91659664	P390	42	9397112	CACSAMP	9270608
CACUSER3	91319456	P390	80	9396976	CACSAMP	9269664
Total Number of USERS: 3						

The **SERVICE** field in the **DISPLAY,USERS** command identifies the query Processor task that receives and responds to all of the user’s SQL requests. This information is valuable in determining the number of users that are currently sharing any particular instance of a Query Processor service.

While each user is assigned a particular service for processing SQL requests, other services may also be required for the user to successfully receive SQL results.

These services include:

- Connection Handlers (service type CACINIT): manages the session with the user’s client process.
- Database Interface Services: databases, such as IMS and DB2 may require additional services to manage specific data requests on behalf of a Query Processor service.

For more information on Connection Handlers and Database Interface services, see the **SERVICE INFO ENTRY** parameter description in [Appendix A, “Configuration Parameters.”](#)

## Displaying Configurations

Configuration information is loaded from members of the **VHSCONF** data set at Server initialization time. To display active configurations while a Server is running, issue the **DISPLAY,CONFIGS** command.



The following is an example of the command with five active configurations:

```
F servername , DISPLAY , CONFIGS
```

Active Configurations:

- MASTER: always active while a Server is running.
- CACQPCF: active when a Query Processor is running and its service information field contains a configuration override name of CACQPCF (as shown in the DISPLAY,CONF=MASTER example).
- CACUSCF1: a user currently connected to the Server.
- CACUSCF3: a user currently connected to the Server.
- CACUSCF5: a user currently connected to the Server.

These configurations remain active until the corresponding users disconnect from the Server.

The initial member identified in the VHSCONF DD is known as the master configuration member. Minimally, there is a master configuration active in all running Servers.

To display the master configuration, issue the following MTO command:

```
F servername , DISPLAY , CONFIG=MASTER
```

The output from this command follows:

```
*(1)MESSAGE POOL SIZE = 7000000
(2)TASK PARAMETERS = NULL
*(3)USER CONFIG = 0
*(4)STATIC CATALOGS = 1
*(5)NL = US ENGLISH
*(6)NL CAT = DD:CACCAT
*(7)BTREE BUFFERS = 4
*(8)LD TEMP SPACE = ALCUNIT=CYL , SPACE=20 , UNIT=SYSDA , EXTEND=5
(9)SAF EXIT = NULL
(10)SMF EXIT = NULL
*(11)MAX ROWS EXAMINED = 0
*(12)MAX ROWS RETURNED = 500
(14)MAX ROWS EXCEEDED ACTION = ABORT
(15)JOIN MAX TABLES ANALYZED = 4
(16)CPU GOVERNOR = NULL
(17)LOCALE = NULL
(18)WLM UOW = NULL
(19)VSAM AMPARMS = NULL
*(101)SERVICE INFO ENTRY = CACCNTL CNTL 0 1 1 100 4 5M 5M
NO_DATA
*(102)SERVICE INFO ENTRY = CACLOG LOGGER 1 1 1 100 4 5M 5M 4K
*(103)SERVICE INFO ENTRY = CACIMSIF IMS 2 0 1 50 4 5M 5M
NO_DATA
*(105)SERVICE INFO ENTRY = CACQP CACSAMP 2 2 4 50 4 5M 5M
CACQPCF
*(106)SERVICE INFO ENTRY = CACINIT XMNT 2 0 1 50 4 5M 5M
XM1/CACQA/CACQA
*(107)SERVICE INFO ENTRY = CACINIT TCPIP 2 1 10 50 4 5M 5M
TCP/199.242.140.11/7067
*(108)SERVICE INFO ENTRY = CACCAF DSN 2 2 5 1 4 5M 5M CACPLAN
```

```
*(109)SERVICE INFO ENTRY = CACDCI DCOM 2 1 1 50 4 5M 5M 4
```

Lines in the output display prefixed by an asterisk (\*) denote configuration values that were either read in at initialization time or updated by an MTO operator **SET** command. All other configuration items are set to their default value as defined in [Appendix A, “Configuration Parameters.”](#)

Displaying the master configuration is the only way to view all the **SERVICE INFO ENTRY**s defined to a running server. While many services can be viewed with a **DISPLAY,SERVICES** command, services that are not running cannot be viewed.

In addition to the master configuration, there are also service level configurations and user level configurations. Both service level and user level configurations are used to override specific configuration values in the **MASTER** configuration at runtime. Service level configurations are defined in the service data of a Query Processor service and are activated each time an instance of that service is started. User level configurations are activated each time a user connects to a Query Processor whose configuration **USER CONFIG** value is set to 1.

Unlike the master configuration, service and user-level configurations do not have **SERVICE INFO ENTRY** values. These configurations are strictly used for overriding the non **SERVICE INFO ENTRY** and **USER CONFIG** descriptions as described in [Appendix A, “Configuration Parameters.”](#)

## Modifying Configuration Parameters

Any active configuration can be dynamically modified with the MTO **SET** command. The format of the **SET** command is as follows:

```
F
servername , SET , NAME=configname , ORD=ordinalnumber , VALUE='value'
```

**WARNING:** Dynamic modification of production servers is not recommended unless the full impact of configuration changes is known.

Configurations are generally static, and modification at runtime is not necessary. To set a configuration value to its system default value, you can issue the **SET** command and pass a **VALUE=NULL** as the configuration parameter's value. The **SERVICE INFO ENTRY** is updated in memory by specifying its associated ordinal value in the **SET** command.

For example, to replace the **SERVICE INFO ENTRY** for the Query Processor, issue the following command:

```
F servername , SET , NAME=MASTER , ORD=105 , VALUE='CACQP CACSAMP /
2 2 10 100 4 5M 0M CACQPCF'
```

New service info entries can also be created by specifying an ordinal VALUE greater than the largest existing ordinal. For example, to create a new Query Processor for a location named MYNEWLOC, issue the command:

```
F servername, SET, NAME=MASTER, ORD=999, VALUE='CACQP MYNEWLOC /
2 2 10 100 4 5M 0M NO_DATA'
```

Changes made to a Master or Query Processor configuration can be written back out to the configuration data set with the MTO **FLUSH** command. If changes were made to the master configuration and you want to have those changes written back to the associated VHSCONF member, issue the command:

```
F servername, FLUSH, NAME=MASTER
```

This command writes out each configuration item in the display that contains an asterisk (\*), which denotes a modification.

**WARNING:** Configurations written with the FLUSH command will not contain any comment lines, even if the original configuration contained comments. For this reason, we recommend backing up configuration data sets prior to issuing the FLUSH command.

In order for the FLUSH command to work successfully, the user ID associated with the Server task must have update authority for the configuration data set.

## Displaying Memory Utilization

The server runs with a fixed pool of memory that must be shared by all services and users. The size of the Server Memory Pool is defined by the MESSAGE POOL SIZE parameter in the MASTER configuration file.

To display the current memory utilization from the message pool, issue the command:

```
F servername, DISPLAY, MEMORY
```

The output of the command is as follows:

```
TOTAL MEMORY 2048K, USED 229K (11%), MAX USED 958K (46%)
```

The memory display includes:

- the total size of the pool,
- the amount of memory used by services and users,
- the percentage of the pool used,
- the maximum amount of memory used since the Server was activated, and
- the percentage of the maximum used in relation to the total storage pool.

While memory requirements vary based on the number of user connections and size of SQL queries, this command can be useful in estimating the maximum number of users allowable for a given Server.

**NOTE:** The MESSAGE POOL SIZE for a Server must be at least two megabytes less than the actual region size for the server as memory must also be available for load modules, C runtime stacks, and OS/390 control blocks.

## Starting and Stopping Individual Services

Using the MTO interface, you can issue commands to start and stop individual services defined in the MASTER configuration.

**NOTE:** Additional information on the available MTO commands can be found in [Appendix C, “MTO Command Reference.”](#)

To start a new instance of a service, issue the command:

```
F servername, START, SERVICE= servicename
```

For example, to start a new instance of the service CACSAMP as shown in [“Displaying Configurations,” on page 120](#), issue the following command:

```
F servername, START, SERVICE=CACSAMP
```

When starting a service, the region controller first checks the number of active instances of the service and the maximum task count for the service before attempting to start a new instance. If the service has not reached its defined maximum task count, the Region Controller starts a new instance of the service.

Stopping a service can be performed either by its service name or its task ID. If a service is stopped by its service name, all active instances of the service are stopped. The formats of the **STOP** command for individual services are as follows:

```
F servername, STOP, SERVICE= servicename  
F servername, STOP, TASKID= taskid
```

**WARNING:** The STOP command cancels any user activity in an active service and disconnects all active users from the stopped service.

Uses for the **START** and **STOP** commands include:

- starting additional instances of a Query Processor;
- stopping idle instances of a Query Processor;

- stopping a Query Processor that is processing an unusually large query;
- switching from IMS Batch access to IMS DRA access by stopping the Batch service instance and starting the IMS DRA instance; and
- starting new connection handlers, such as Cross Memory and TCP/IP.

## Stopping the Server

To stop the Server, issue the following command:

```
F servername, STOP, ALL
```

**WARNING:** The STOP command cancels any user activity in the Server and disconnects all active users.



# 11

## Views

### Introduction to Views

Database tables define the structure and organization of the data it contains. Using SQL, you can look at the stored data in other ways by defining alternative views of the data. A **view** is an SQL query that is stored in the database and assigned a name, similar to a table name. The results of the stored query are then visible through the view, and SQL lets you access these query results as if the results were a real table in the database.

Views allow you to:

- tailor the appearance of a database so that different users see it from different perspectives;
- restrict access to data, allowing different users to see only certain rows or certain columns of a table; and
- simplify database access by presenting the structure of the stored data in the way that is most natural for each user.

This chapter describes how to use views to simplify processing and how to use them for record and column filtering. It includes the following topics:

- [“What is a View?”](#) on page 128,
- [“How the Query Processor Handles Views,”](#) on page 129,
- [“How to Create a View,”](#) on page 130,
- [“Advantages and Disadvantages of Views,”](#) on page 130,
- [“Using Views for Record and Column Filtering,”](#) on page 132,
- [“Joined Views,”](#) on page 134, and
- [“Dropping Views,”](#) on page 135.

## What is a View?

A view is a virtual table in the database whose contents are defined by a query. To the database user, the view appears to be a real table, with named columns and rows of data. Unlike a real table, a view does not exist in the database as a stored set of data values. Instead, the rows and columns of data visible through the view are the results of an SQL query defining the view. This view looks like a table because the SQL query gives it a table name and stores the definition of the view in the database.

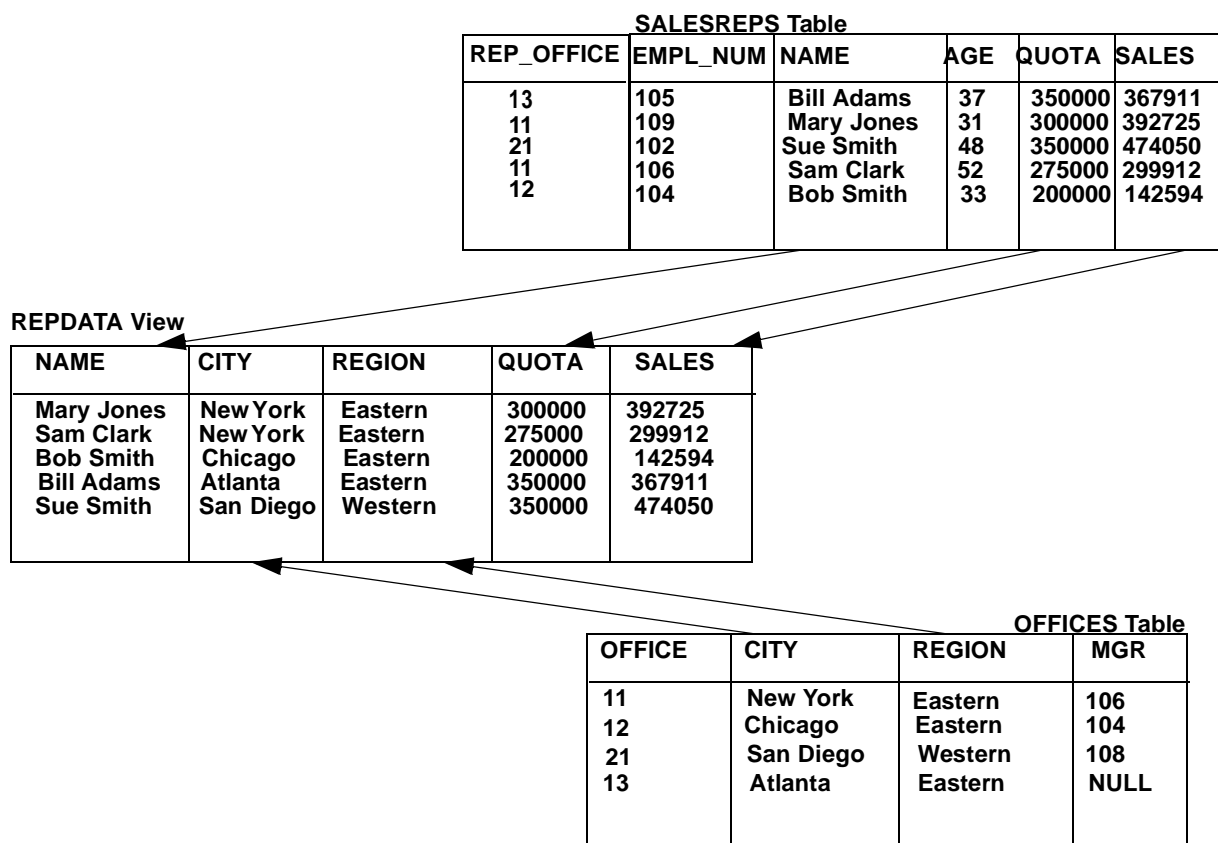
The following two-table query produces the view shown in [Figure 16: “View With Two Source Tables,”](#) on page 129. The view is given the name REPDATA.

```
SELECT NAME, CITY, REGION, QUOTA, SALESREPS.SALES
       FROM SALESREPS, OFFICES
       WHERE REP_OFFICE = OFFICE
```

The data in the view comes from the SALESREPS and OFFICES tables. These tables are called source tables for this view because they are the source of the data in the view. This view contains one row of information per salesperson, extended with the name of the city and region of the salesperson. The view appears as a table and its contents look just like the query results that would be obtained if the query were actually run.



Figure 16: View With Two Source Tables



Once a view is defined, you can use it in a SELECT statement, just like a real table.

## How the Query Processor Handles Views

When the Query Processor encounters a reference to a view in an SQL statement, it finds the definition of the view stored in the database. Then the Query Processor translates the request that references the view into an equivalent request against the source tables of the view and then carries out that request. The Query Processor maintains the illusion of the view while maintaining the integrity of the source tables.

## Advantages and Disadvantages of Views

Views provide a variety of benefits and can be useful for many types of databases. In a personal computer database, views are usually a convenience, defined to simplify requests to databases. In production database installation, views can play an important role in defining the structure of the database for users or groups of users and can enforce the database security.

Views provide the following benefits:

- Built-in security by giving each user permission to access the database only through a small set of views containing the specific data the user or group of users is authorized to see, restricting user access to other data.
- Simplicity for queries because a view can draw data from several tables and present it as a single table, simplifying the information and turning multi-table queries into single-table queries for a view.
- Simplicity in structure because views give users a specific view of the database structure, presenting the database as a set of virtual tables specific to particular users or groups of users.
- Stabilization of information because views present a consistent, unchanged image of the database structure, even if underlying source tables are changed.

Although there are many advantages to views, the main disadvantage to using views, rather than real tables is performance degradation. Since views only create the appearance of a table, not a real table, the Query Processor must still translate queries against the view into queries against the underlying source tables. If the view is defined by a complex, multi-table query, then even simple queries against the view become complicated joins that may take a long time to complete.

## How to Create a View

eXadas supports the creation and management of DB2-compatible views using either the eXadas Meta Data Utility or any eXadas client connected to the Server. Although any client can create a view with standard SQL processing, the Meta Data Utility is a more controlled mechanism for creating and managing views. CrossAccess recommends that you use the Meta Data Utility to create views.

This section describes how to create a simple view, how to use views for record filtering, and a brief description about using views for security.

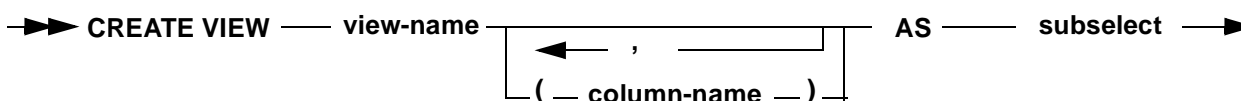
The CREATE VIEW statement can optionally assign a name to each column in the newly created view. If a list of column names is specified, it must have the same number of items as the number of columns produced by the query. Only column names are specified, the data type, length, and other characteristics of the columns are derived from the definition of the columns in the source tables. If the

list of column names is not included in the CREATE VIEW statement, each column in the view will have the name of the corresponding column in the query. The list of column names must be specified if the query includes calculated columns or if it produces two columns with identical names.

The CREATE VIEW statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

The CREATE VIEW syntax diagram and the descriptions follow.

**Figure 17: CREATE VIEW Statement Syntax Diagram**



**Table 10: CREATE VIEW Syntax Statement Descriptions**

Statement	Description
view-name	<p>Assigns a name to the view. The name cannot identify a table, view, alias, or synonym that exists at the current server. The name can be a two-part name. The authorization name that qualifies the name is the view's owner.</p> <p>For information about security for views, see <a href="#">“Using Views for Security,”</a> on page 134, or <a href="#">“Table and View Privileges ,”</a> on page 78.</p>
column-name	<p>Names the columns in the view. If you specify a list of column names, it must consist of as many names as there are columns in the result table of the subselect. Each name must be unique and unqualified. If you do not specify a list of column names, the columns of the view inherit the names of the columns of the result table of the subselect.</p> <p>You must specify a list of column names if the result table of the subselect has duplicate column names or an unnamed column (a column derived from a constant, function, or expression that was not given a name by the AS clause).</p>
AS subselect	<p>Defines the view. At any time, the view consists of the rows that would result if the subselect were executed. Subselect cannot refer to host variables or include parameter markers (question marks).</p> <p>A query containing either a UNION or an ORDER BY clause is not a valid subselect.</p>

## Using Views for Record and Column Filtering

Filtering allows you to define one or more views on a single table mapping to select specific records based on the type of column. This is useful in cases where multiple record types are kept in a single database. Filtering at the column level allows you to create views that contain only a subset of the complete table mapping.

This section describes how views can be used to filter records and columns so that views contain only the information you want presented to a particular user or group of users.

The following types of views are discussed:

- horizontal views, which give access to selected rows of a table;
- vertical views, which give access to particular columns within a particular table;
- row/column subset views, which give access to particular rows and columns in a particular table; and
- grouped views, which (using the GROUP BY clause) group related rows of data and produce one row of query results for each group.

### Horizontal Views

One common use of views is to restrict a user's access to only selected rows in a particular table. For example, in the sample database, you may want to let a sales manager see only the SALESREPS rows for salespeople in that manager's region. To do this, using our SALESREPS Table from [Figure 16: "View With Two Source Tables," on page 129](#), you would create two views, one for the Eastern sales reps (EASTREPS) and one for the Western sales reps (WESTREPS), using the following CREATE VIEW statements:

```
CREATE VIEW EASTREPS AS
    SELECT *
        FROM SALESREPS
        WHERE REP_OFFICE IN (11, 12, 13);
CREATE VIEW WESTREPS AS
    SELECT *
        FROM SALESREPS
        WHERE REP_OFFICE IN (21, 22)
```

This type of view is called a horizontal view because it slices the source table horizontally when it creates the view. All of the columns from the source table are present in the view, but only a subset of the rows are included in the view. These types of views are appropriate only when the source table contains data that relates to various organizations or users, for example the managers of the Eastern and Western sales regions.

Horizontal views can also be used to filter relational records when a database or file contains multiple record types.

## Vertical Views

Another use of views is to restrict user access to particular columns in a table. For example, the order processing department may require access to the employee number, name, and office assignment for each salesperson in order to process orders correctly. The order processing staff does not need to see the salesperson's year-to-date sales or the quota. This selective view of our SALESREPS Table from [Figure 16: "View With Two Source Tables," on page 129](#), is created using the following CREATE VIEW statement:

```
CREATE VIEW REPINFO AS
    SELECT EMPL_NUM, NAME, REP_OFFICE
    FROM SALESREPS
```

This type of view is called a vertical view because it includes all of the rows in a particular table, but limits the columns to only a subset, pertinent to a particular user or group of users. The view is derived from a single source table. The select list in the view definition determines the columns of the source table that will be visible in the view. In a vertical view, every row of the source table is represented in the view and the view definition does not include a WHERE clause.

## Row/Column Subset Views

When you define a view, SQL does not restrict you to purely horizontal or vertical views of a table. It is very common to define a view that slices the table both horizontally and vertically. For example, to define a view that contains the customer number, company name, and credit limit of all customers assigned to Bill Adams (employee number 105), issue the following CREATE VIEW statement:

```
CREATE VIEW BILLCUST AS
    SELECT CUST_NUM, COMPANY, CREDIT_LIMIT
    FROM CUSTOMERS
    WHERE CUST_REP = 105
```

The data visible through this view is called a row/column subset of the CUSTOMERS table. Only the columns explicitly named in the select list of the view and the rows that meet the search condition are visible through the view.

## Grouped Views

A query specified in a view definition may include a GROUP BY clause. This type of view is called a grouped view because the data visible in this view is the result of a grouped query. Grouped views perform the same function as grouped queries: they group related rows of data and produce one row of query results for each group, summarizing the data of that group. A grouped view turns grouped query results into a virtual table, allowing you to perform additional queries.

An example of a grouped view that contains summary order data for each salesperson is:

```
CREATE VIEW ORD_BY_REP (WHO, HOW_MANY, TOTAL, LOW, HIGH,
```

```
AVERAGE) AS
  SELECT REP, COUNT(*), SUM(AMOUNT), MIN(AMOUNT),
  MAX(AMOUNT),
  AVG(AMOUNT)
  FROM ORDERS
  GROUP BY REP
```

The definition of a grouped view, as this example shows, always includes a column name list. The list assigns names to columns in the grouped view that are derived from column functions, such as SUM() and MIN(). It can also specify a modified name for a grouping column. In this example, the REP column of the ORDERS table becomes the WHO column in the ORD\_BY\_REP view. Once the grouped view is defined, it can be used to simplify queries.

Unlike horizontal or vertical view, the rows in a grouped view do not have a one-to-one correspondence with the rows in the source table. A grouped view is not a filter on the source table that screens out certain rows and columns. It is a summary of the source tables and requires a substantial amount of processing to maintain the illusion of a virtual table for grouped views.

Grouped views can be used in queries but cannot be updated and therefore function as read-only views. They are subject to the SQL restrictions on nested column functions.

## Using Views for Security

Views can also be used to restrict user access to particular tables by allowing viewers to see only particular portions of a table. This is done during view creation when owners are set up for that view. Security can also be set up by granting or revoking user privileges to particular views of data. For more information about using the GRANT and REVOKE statements to restrict user access to views, see [Chapter 7, “SQL Security.”](#)

## Joined Views

One of the most frequent reasons for using views is to simplify multi-table queries. By specifying a two-table or a three-table query in the view definition, you can create a **joined view**. Joined views draw their data from two or three different tables and present the query results as a single virtual table. Once you've defined the view, you can use a simple, single-table query against the view for requests that would otherwise require a two-or-more-table join.

For example, if a user often runs queries against a particular table, such as the ORDERS table in the sample database, but does not want to work with employee

numbers, he would want a view of the `ORDERS` table that has names instead of numbers.

An example of that view is:

```
CREATE VIEW ORDER_INFO (ORDER_NUM, COMPANY, REP_NAME,
AMOUNT) AS
  SELECT ORDER_NUM, COMPANY, NAME, AMOUNT
     FROM ORDERS, CUSTOMERS, SALESREPS
  WHERE CUST = CUST_NUM
     AND REP = EMPL_NUM
```

This view is defined by a three-table join. As with a grouped view, processing required to create this virtual table is substantial. Each row of the view is created from a combination of one row from the `ORDERS` table, one row from the `CUSTOMERS` table, and one row from the `SALESREPS` table.

Although this view has a complex definition, it can be very valuable. For example, the following query against this view can be created:

```
SELECT REP_NAME, COMPANY SUM(AMOUNT)
   FROM ORDER_INFO
  GROUP BY REP_NAME, COMPANY
```

that generates a report of orders, grouped by salesperson:

REP_NAME	COMPANY	SUM(AMOUNT)
Bill Adams	ACME Mfg.	\$35,582.00
Bob Burns	JCP Inc.	\$24,343.00
Dan Jones	First Corp.	\$75,000.00

This query is now a single-table `SELECT` statement, which is far simpler than the original three-table query. Also, the view makes it easier to see what's going on in the query. The QP, however, still must work harder to generate the query results for this single-table query against the view as it would to generate query results for the same three-table query. However, for the actual user, it is much easier to write and understand a single-table query that references the view.

## Dropping Views

To drop a view, you must use the `DROP VIEW` statement. It allows detailed control over what happens when a user attempts to drop a view when the definition of another view depends on it. For example, if two views on the `SALESREPS` table were created by these two `CREATE VIEW` statements:

```
CREATE VIEW EASTREPS AS
  SELECT *
     FROM SALESREPS
  WHERE REP_OFFICE IN (11, 12, 13)
```

```
CREATE VIEW NYREPS AS
  SELECT *
  FROM EASTREPS
  WHERE REP_OFFICE = 11
```

the following **DROP VIEW** statement removes both views as well as any views that depend on their definition from the database:

```
DROP VIEW EASTREPS
```

**NOTE:** eXadas does not directly support the **CASCADE** and **RESTRICT** options in the **DROP VIEW** syntax; however, the **CASCADE** option is implied in the **DROP VIEW** syntax. So whenever the **DROP VIEW** is used, it will delete dependent views along with those specified in the **DROP VIEW**.



# 12

## Server Logging

### Introduction to Server Logging

The server includes a logging facility to assist in problem determination. The types and amount of information written to the log is controlled by a set of user-configurable parameters. In addition, a log print utility is supplied with the product to format and print records written to the log data set.

The log data set is a binary sequential file allocated to the DDNAME CACLOG with attributes of RECFM=FBS, LRECL=1, BLKSIZE=4080. Generally, this data set can be a temporary file allocated to the server, which is then automatically deleted at server termination. However, for problem determination purposes, you may be asked to allocate this file to a permanent data set by CrossAccess Technical Support.

This chapter discusses the following topics:

- [“Controlling Logged Information,”](#) on page 138,
- [“The Log Print Utility,”](#) on page 139, and
- [“Log Print Filtering,”](#) on page 140.

# Controlling Logged Information

The amount of information written to the eXadas log file is controlled by the Tracing Level field (field 7) in the SERVICE INFO ENTRY parameter. The tracing level on services other than the logger controls the type of information that is sent to the log.

The valid trace levels for use with the logger include:

- 4: send only warning and error messages to the logger.
- 20: prevent logging of any information by the service.

Trace levels set less than 4 will send informational trace information to the logger for problem determination purposes.

**WARNING:** Do not change Trace Level settings for any service unless requested to do so by CrossAccess Technical Support.

The logger task receives trace and error messages from all running services and either writes them directly to the log file or caches them to a fixed-length memory buffer. When the memory buffer is full, the logger automatically re-uses the buffer by overwriting the oldest messages in the buffer with any new messages logged.

By default, the memory buffer used for cached messages is 16 kilobytes. This value can be configured on the SERVICE INFO ENTRY parameter of the logger task itself by passing a numeric kilobyte value in the Service Information field (field 10).

The following example shows a logger with a memory buffer size of 256 kilobytes. The acceptable range for a memory buffer specification is 4 KB - 1000 KB. Any other value is ignored and the default buffer size is used.

```
SERVICE INFO ENTRY = CACLOG LOG 1 1 1 100 4 5M 0M 256K
* 256K memory cache buffer ^
```

The trace level value (field 7) on the SERVICE INFO ENTRY parameter for the logger task determines whether messages are written or cached. When the logger receives messages, it first checks to see if the logger is currently writing messages to file. If so, the logger continues writing all messages to the log file. Otherwise, it compares the severity of the received message to its configured trace level and only writes the message if the severity on the message is greater than or equal to its own trace level.

When the logger switches from caching messages to writing messages, it first writes out all cached messages in its memory buffer prior to writing the message that triggered the write. From that point on, all messages logged are immediately written to the log file regardless of their severity.

For example, given the following configuration entries:

```
* eXadas logger service
SERVICE INFO ENTRY = CACLOG LOG 1 1 1 100 4 5M 0M NO_DATA
*                               ^ Logger Trace
*                               Level
* eXadas Query Processor service
SERVICE INFO ENTRY = CACQP CACSAMP 2 1 4 50 2 5M 5M CACQPCF
*                               ^ QP Trace Level
```

the Query Processor will send tracing and diagnostic messages to the logger based on a trace level of 2. The logger itself will take these messages and cache them into its memory buffer until it receives a message with a severity of 4 or higher. In this case, if no warning level 4 or higher severity messages are sent to the logger while the server is active, the logger will never send messages to the log file.

Generally, when a CrossAccess Technical Support representative requests modification of a trace level for a non-logger service, you will also be asked to set the logger task trace level to the same value. This ensures that all messages issued by the service are immediately written to log.

## The Log Print Utility

The log print utility is a separate program supplied with eXadas that allows you to format and print out messages written to the log. In addition, this utility can print a summary report of the messages in the log as well as filter which messages in the log are printed. See the CACLGFLT member in the SCACSAMP library shipped with your eXadas software for a sample log and its output.

The log summary output lists the information found in the logged messages. Included in the list is the start and stop time of messages in the log, the minimum and maximum return codes for the logged messages, which filters were used, and the tasks and nodes logged.

The format of the log summary can be used as filtering input to the future printing of the log. For example, to print just the messages associated with task 9283960, you can pass a SYSIN control card containing: `Tasks=(9283960)` to the log print utility and only messages issued by that specific task are printed. For more information on filtering log print output, see [“Log Print Filtering,” on page 140](#).

# Log Print Filtering

The logger records all messages it receives during server execution. In many cases, these messages include many tasks, nodes, and filtered components that may make problem determination difficult. Therefore, the log print utility accepts SYSIN control cards that contain filtering information to select specific messages to print. These specific control cards can be used to print messages for a specific timeframe, a specific task, a range of return codes, or any combination of the elements listed in the log summary report.

To simplify filtering, the format of the SYSIN filtering is exactly the same as the format of the summary report. Therefore, you can print a summary report or the log file and edit it to remove or modify the elements in the report that you want to print.

**NOTE:** Blank lines and lines whose first character contains an asterisk (\*) are ignored by the log filtering logic.

Using the example summary report from the previous section, you can pass the SYSIN filtering information (shown in the example that follows) to print a report of all messages issued by the Query Processor between 09:08 and 09:16 that had a return code of 8:

```
StartTime='2000/08/05 09:08:04:0000'
StopTime= '2000/08/05 09:16:18:0000'
MinRC=1
MaxRC=12
Filters=(SAVANT)
ExFilters=(MESSAGE)
Tasks=(9194800,9219720)
Nodes=(81)
SPCRC=(ffffffff)
```

Under normal circumstances, you will not see any logged messages in server output, nor will it be necessary to print logs or modify the SERVICE INFO ENTRY parameter trace levels. This information is provided solely to assist you in problem determination while working with CrossAccess Technical Support.

**NOTE:** The Log Print Utility requires a configuration member. In normal circumstances, you would use the same configuration member that the server uses. This is also true when performing an off-line print from a permanent log file. The following configuration parameters are used:

- MESSAGE POOL SIZE
- NL
- NL CAT

For more information on these parameters, see [Appendix A, “Configuration Parameters.”](#)

# 13

## Utilities

### Introduction to Utilities

This chapter describes the utilities delivered with eXadas. These utilities include:

- [“Meta Data Utility,” on page 142,](#)
- [“DB2 Grammar,” on page 184,](#)
- [“CICS VSAM Grammar,” on page 190,](#)
- [“ADABAS USE Statement Generator,” on page 192.](#)

The remainder of this section describes each of these utilities. For each utility, an overview of the function the utility performs and a description of how to run the utility are described. A description of the control card input syntax and the control card parameters used to control the utilities execution are also included.

# Meta Data Utility

The Meta Data Utility is used to complete the process of mapping data into a logical table that eXadas can access. The Meta Data Utility accepts Meta Data Grammar input, which contains the mapping information. The Meta Data Utility reads the Meta Data Grammar and populates Meta Data Catalogs for use by the Query Processor. The Meta Data Catalogs are stored in two files referenced by the CACCAT and CACINDX DD statements in both the Meta Data Utility and the Server JCL streams. The Meta Data Utility supports incremental updates to the Meta Data Catalogs and can be run while Servers are active.

**WARNING:** The Meta Data Utility will fail if a Server is currently using the catalog with STATIC CATALOG parameter set.

**NOTE:** Meta Data Catalogs are stored in standard OS/390 (spanned) Sequential files. Meta Data Catalogs cannot be stored in a partitioned data set.

Additional Meta Data Catalogs can also be accessed in a single server by defining them per data source. See [Appendix D, “Sample SERVICE INFO ENTRY Definitions,”](#) for more information.

The Meta Data Catalogs contain eXadas versions of the following standard DB2 system tables:

- SYSIBM.SYSTABLES
- SYSIBM.SYSCOLUMNS
- SYSIBM.SYSINDEXES
- SYSIBM.SYSKEYS
- SYSIBM.SYSTABAUTH
- SYSIBM.SYSVIEWS
- SYSIBM.SYSVIEWDEP
- SYSIBM.SYSROUTINES
- SYSIBM.SYSROUTINEAUTH
- SYSIBM.SYSPARMS
- SYSIBM.SYSUSERAUTH
- SYSIBM.SYSDBAUTH

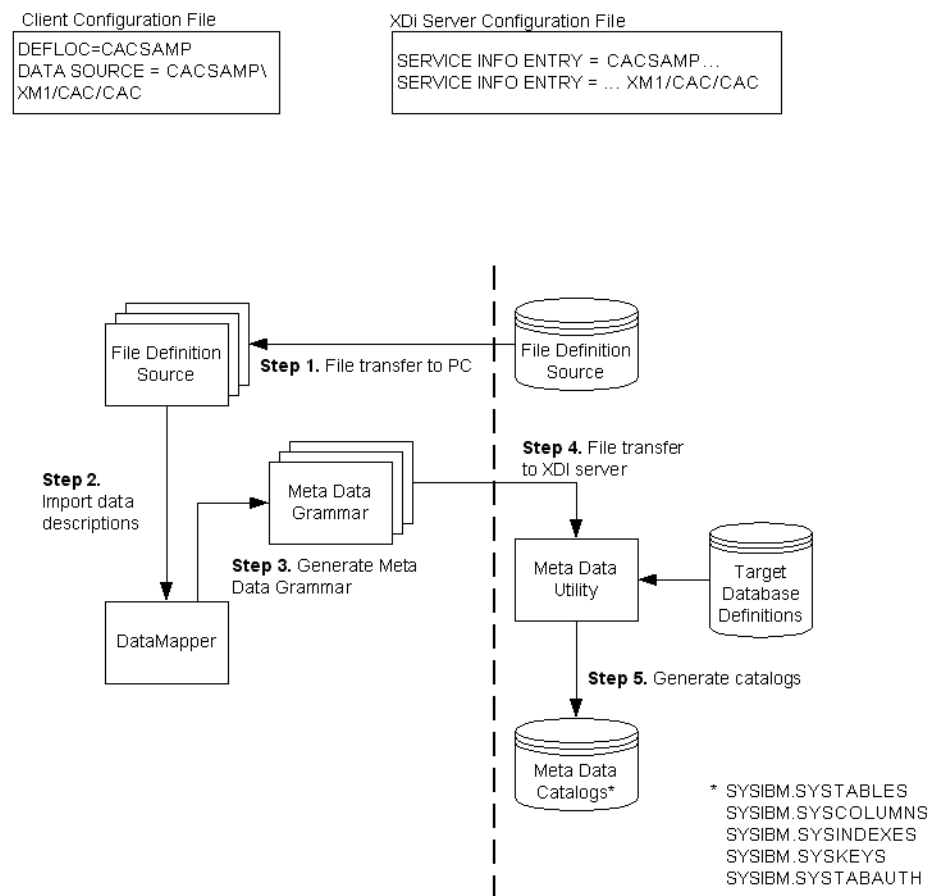
The eXadas Meta Data Catalogs are based on IBM DB2 version 2.2. The Meta Data Catalogs contain the standard column definitions found in the DB2 version 2.2 system catalogs. The Meta Data Catalogs also contain additional information that eXadas uses to optimize access to the physical database files. Your applications can use standard queries to access the information stored in the DB2 columns.

Meta Data Grammar for input into the Meta Data Utility is created using the DataMapper. The DataMapper is a Windows-based application that takes in information about your existing databases and generates Meta Data Grammar.

**NOTE:** Meta Data Grammar can also be manually keyed into a text editor, but this method is not recommended as it is both difficult and error-prone.

Figure 18: “Data Administration Workflow,” on page 143, shows the general procedural steps required for creating logical tables from data information. File Definition Source in this diagram includes COBOL copybooks, IMS DBD source definitions, and IDMS DD dictionary definitions. File transfers between the server and a PC running the DataMapper can be performed using FTP support built into the DataMapper.

**Figure 18: Data Administration Workflow**



The remainder of this section provides step-by-step instructions about running the Meta Data Utility and the resulting Meta Data Grammar.

## Meta Data Grammar

Meta Data Grammar is supplied in a standard 80-byte fixed-format text file. The text file contains free form statements that define the contents of a logical table or an index. Additional statements can be supplied to delete an existing logical table or an index definition. These definitions use a “keyword value” syntax and can span multiple lines. Meta Data Grammar can contain comments. When including comments in Meta Data Grammar, use the following format:

```
/* comment */
```

Anything between the delimiters (`/* */`) is treated as a comment. The comment may span more than one line, but must not begin in column one.

**NOTE:** If you are editing a Meta Data Grammar file manually using an editor like ISPF, make sure that NUM OFF is set in the edit profile when editing the Meta Data Grammar.

The following statements can be specified in the Meta Data Grammar. They include:

- **USE TABLE:** Adds a logical table to the Meta Data Catalogs. This causes the SYSIBM.SYSTABLES and SYSIBM.SYSCOLUMNS tables in the Meta Data Catalogs to be populated with information about the logical table.
- **USE [UNIQUE] INDEX:** Adds an index to the Meta Data Catalogs. This causes the SYSIBM.SYSINDEXES and SYSIBM.SYSKEYS tables in the Meta Data Catalogs to be populated with index information for a logical table. The logical table must be defined in the Meta Data Catalogs using a USE TABLE statement.
- **DROP TABLE:** Removes an existing logical table from the Meta Data Catalogs. This causes all of the information in the SYSIBM.SYSTABLES, SYSIBM.SYSCOLUMNS, SYSIBM.SYSINDEXES, and SYSIBM.SYSKEYS tables to be removed for the logical table specified in the DROP TABLE statement.
- **DROP INDEX:** Removes an existing index from the Meta Data Catalogs. This causes all information in the SYSIBM.SYSINDEXES and SYSIBM.SYSKEYS tables to be removed from the Meta Data Catalogs for the specified index.
- **GRANT:** Assigns one or more privileges to specific users. This causes one or more of the following tables to be populated: SYSIBM.SYSDBAUTH, SYSIBM.SYSROUTINEAUTH, SYSIBM.SYSTABAUTH, SYSIBM.SYSUSERAUTH. See [Chapter 7, “SQL Security,”](#) for more information.
- **REVOKE:** Removes one or more privileges from specific users. This causes information to be removed from one or more of the following tables: SYSIBM.SYSDBAUTH, SYSIBM.SYSROUTINEAUTH, SYSIBM.SYSTABAUTH, SYSIBM.SYSUSERAUTH. See [Chapter 7, “SQL Security,”](#) for more information.



- **CREATE VIEW:** Adds a virtual table whose contents are defined by a query to the Meta Data Catalog. This causes the SYSIBM.SYSVIEWDEP and SYSIBM.SYSVIEWS tables to be populated with information about the view. See [Chapter 11, “Views,”](#) for more information.
- **DROP VIEW:** Removes a table view from the Meta Data catalog. This causes information to be removed from the SYSIBM.SYSVIEWDEP and SYSIBM.SYSVIEWS tables. See [Chapter 11, “Views,”](#) for more information.
- **CREATE PROCEDURE:** Defines a stored procedure in the Meta Data Catalog. This causes information in the SYSIBM.SYSROUTINES table to be populated with information about the stored procedure. See [“Defining Stored Procedures,” on page 274,](#) for more information.
- **DROP PROCEDURE:** Removes a stored procedure from the Meta Data catalog. This causes information to be removed from the SYSIBM.SYSROUTINES table. See [“Defining Stored Procedures,” on page 274,](#) for more information.

The syntax for each of these statements is described in the sections that follow or the referenced chapter.

**NOTE:** All Meta Data statements must be terminated by a semi-colon (;).

## Running the Meta Data Utility

### To run the Meta Data Utility to complete the mapping process:

1. Edit the Meta Data Utility JCL.

Sample Meta Data Utility JCL is found in SCACSAMP member CACMETAU, shipped with your eXadas software.

2. Basic customization.

Supply a valid job card and modify the CAC high-level qualifier to reference the high-level qualifier for the eXadas data sets. At this point, you may want to save the member since the eXadas high-level qualifier is not likely to change after each execution of the Meta Data Utility.

3. IMS mapping customization.

If you are planning to map IMS data then you must modify the IMS high-level qualifier to reference the IMS high-level qualifier where the DBDs referenced in the Meta Data Grammar are located. You must also uncomment the DBDLIB DD statement. Save the member since the IMS high-level qualifier is not likely to change from run-to-run of the Meta Data Utility.

4. Sequential/VSAM mapping customization.

When mapping Sequential and/or VSAM data the Meta Data Grammar can reference the Sequential/VSAM file to be accessed by the logical table either by its data set name or by using a DD name. When referenced by a data set name, the Meta Data Utility dynamically allocates the data set name

referenced to check for its existence and to gather physical attributes about the file.

For Sequential and non-CICS VSAM data, if you are referencing a file by DD name, then you need to add DD statement(s) with the same DD name(s) in the Meta Data Utility JCL. You should specify a DISP=SHR for these data sets. The Meta Data Utility will open these files during processing in order to obtain physical information about the file(s).

**NOTE:** To validate VSAM mapping parameters at METAU execution time, the METAU must connect to CICS and query information about the VSAM file(s) CICS controls. This is accomplished using parameters defined in the CONNECT TO CICS statement. See [“CICS VSAM Grammar,” on page 190](#), for additional information about the CONNECT TO CICS statement.

#### 5. CA-DATACOM/DB mapping customization.

If you are planning to map CA-DATACOM/DB data, you must gather information about the structure of the CA-DATACOM/DB table that you plan to map. You must also modify the Meta Data Utility JCL to include CA-DATACOM/DB specific DD statements and control information. These requirements are as follows:

The CA-DATACOM/DB Source Language Generation (SLG) Facility allows you to generate field definitions for CA-DATACOM/DB tables as high-level language statements suitable for COBOL copybooks. To use this Source Language Generation Facility, you place the appropriate transactions in the CA-DATACOM/DB DDUTILITY job stream. See the topic “Generating Source Language Statements” in the *CA-DATACOM/DB Datadictionary Batch Guide* for detail information about executing this utility program. After the source language generation is completed, use the embedded DataMapper FTP facility to download the COBOL copybook for import into the DataMapper.

The Meta Data Utility JCL must be modified to include three CA-DATACOM/DB libraries into the STEPLIB concatenation. These libraries contain load modules, used to access the CA-DATACOM/DB Datadictionary Service Facility, and control information like load module DDSRTL (the CA-DATACOM/DB System Resource Table). The libraries you must include are CAI.CAILIB, CAIDATACOM.CHLQ.CUSLIB and CAI.DATACOM.THLQ.CAILIB.

The Meta Data Utility JCL must be modified to include a DD statement named DDIDENT that specifies a sequential file or member of a PDS that contains CA-DATACOM/DB Datadictionary Service Facility access information. The access information required is an ENTITY-OCCURRENCE-NAME (user name) and a QUALIFIER (password). The ENTITY-OCCURRENCE-NAME parameter is specified by entering 'USER = ' followed by a user name. The name must be the same as a CA-DATACOM/DB PRODUCTION status PERSON occurrence. If you leave this field blank, CA-DATACOM/DB Datadictionary Service Facility assumes there is no authorization. The QUALIFIER parameter is specified by entering

'PASSWORD =' followed by a user password if one has been assigned. As with any file containing sensitive information, this access information file should be secured according to site specifications.

These access information parameters are entered free-form in a standard text file. The parameters may be entered on the same line separated by a comma or each parameter may be entered on a separate line. If a single parameter is entered more than once, the last occurrence of that parameter will be used when accessing the CA-DATACOM/DB Datadictionary Service Facility. The number of leading and trailing blanks or the number of spaces before and after the equal sign is inconsequential.

A User Requirements Table (URT) must be provided on every request for service sent to CA-DATACOM/DB. Every service request is validated against an open User Requirements Table. This technique provides security (by restricting access) and efficient allocation of CA-DATACOM/DB resources. When you define your User Requirements Tables, consider the security implications. You must decide whether you want to have one User Requirements Table per CA-DATACOM/DB table that you map into the eXadas catalog or have only a few User Requirements Tables for all CA-DATACOM/DB tables that you map into the eXadas catalog. There are obvious security implications when you do the latter.

The URT specifies what CA-DATACOM/DB resources are required and is generated by a macro assembly. The macros used to generate a URT are as follows:

- **DBURSTR:** This User Requirements Table macro defines global program parameters. It must be the first macro in the assembly source and can be defined only once per User Requirements Table assembly. The macro does not generate any actual assembly output. Its parameter data is passed to the following macros.
- **DBURTBL:** This User Requirements Table macro identifies specific program parameters for a single CA-DATACOM/DB table. A separate DBURTBL macro must be defined for each CA-DATACOM/DB table that is accessed by the Server. The name must be specified with a TBLNAM=name parameter, where name is the 3-character CA-DATACOM/DB table name defined in CA-DATACOM/DB. The database ID number is specified in the DBID= parameter. All other DBURTBL macro parameters are allowed to generate their default value. Like DBURSTR, this macro does not generate any actual assembly output either. Its parameter data is passed to the DBUREND macro.
- **DBUREND:** This User Requirements Table macro validates all previous input parameters and generates the assembly output. It must be the last macro in the assembly source and can be defined only once per User Requirements Table assembly.

For more detail information regarding these macros and their associated parameters, see the *CA-DATACOM/DB Database and System Administrator Guide*. To see a sample User Requirements Table, see the CACDCURT member in the SCACSAMP library.

Once the User Requirements Table macro entries are coded, the URT must be assembled and link-edited into an accessible load library. To see sample JCL for assembling and link-editing the URT, see the CACDCURA member in the SCACSAMP library. The load module name specified in the link-edit step must be specified in the Meta Data Utility grammar when the CA-DATACOM/DB table is mapped into the eXadas catalog. Using this name, the URT will be dynamically loaded by the eXadas server at execution-time. During processing, the Server then opens the URT prior to requesting CA-DATACOM/DB service and closes the URT when processing is completed.

**NOTE:** Whenever a new table is to be mapped into the eXadas catalog, an appropriate URT must be created or updated, assembled, link-edited, and the load module name must be specified in the Meta Data Utility grammar for the new table. Failure to include the CA-DATACOM/DB tablename and database ID in the specified URT will result in an access failure while trying to process the new table.

6. ADABAS mapping customization.

Customize the SCACSAMP member CACADADD. Specify the mode, SVC number, database ID, and device type for your Adabas database.

7. IDMS mapping customization.

Use the sample JCL CACIDPCH (in the SCACSAMP data set) to generate IDMS schema and subschema reports. In the JCL CACMETAU, modify the DD card DDLPUNCH to point to the members generated.

8. Determine whether you need to create the Meta Data Catalogs.

Additional catalogs can be created, if required, by using the JCL in the member CACCATLG in the SCACSAMP data set.

9. Identify the Meta Data Grammar to be used as input to the Meta Data Utility.

Specify the name of the Meta Data Grammar file to use as input.

When mapping your own data modify the MEMBER parameter to specify the name of the PDS member that contains the Meta Data Grammar to map. You may also need to change the SYSIN data set name to reference the name of the data set where the Meta Data Grammar is located.

10. Add a user ID and password to the job card if necessary.

**WARNING:** Once the catalogs are initialized, security has been set up with System Administration authority (SYSADM) granted to the user ID who installed eXadas and ran the Meta Data Utility. That user ID is the only user who can access the system or the catalogs. To turn off security, the new System Administrator must either grant SYSADM authorization to PUBLIC, allowing all users access and thus negating security, or grant table access authority to individual user IDs. For additional information about eXadas security, see the [Chapter 7, “SQL Security.”](#)

11. Grant catalog access rights.

Once the catalogs are loaded with the new tables, you will need to grant the appropriate access rights. In the SCACSAMP data set, there is a member called CACGRANT. This member contains JCL to load the catalogs with the appropriate access rights to the tables.

This job will read in its input from the CACGRIVP member. This member contains the GRANTs required to access the samples tables. If you are bringing your own tables on line with eXadas, you must add the appropriate GRANTs for the new tables. See [Chapter 7, “SQL Security,”](#) for more information.

To customize the CACGRANT JCL to run in your environment, perform the following steps:

- a. Ensure the symbolic GRAMMAR = is pointing to the appropriate member containing the desired security commands.
- b. Ensure that the CACCAT and CACINDX DDs refer to the catalogs created using the CACCATLG JCL.
- c. Review CACGRIVP and uncomment the appropriate GRANT for your database.
- d. Submit.

Once this job completes, the catalogs have been loaded with the desired security.

12. If you plan to use tools that require access to the catalog information, you must run CACGRANT using CACGRSYS as the input.

13. Review the output.

Once the Meta Data Utility has completed executing review the output listing. The Meta Data Utility should complete with a COND CODE of zero or 4. A COND CODE of 4 indicates that an attempt to DROP a logical table was made, but the table does not exist. This is merely a warning message that you may receive the first time you map a logical table.

A condition code higher than 4 indicates that the Meta Data Utility encountered an error. Review the listing to determine what the error is.

**NOTE:** Before a server can access the physical databases/files you mapped, it must have authority to read these database/files. Contact your security administrator to inform them of the user ID(s) that the Server(s) will be using, so the security administrator can grant the proper authority.

This completes the basic mechanics for running the Meta Data Utility. For verification purposes CrossAccess recommends you map the Meta Data Grammar for the sample IMS, Sequential, and VSAM logical tables supplied with the system. Instructions for each of these samples follows.

## USE TABLE Statement Syntax

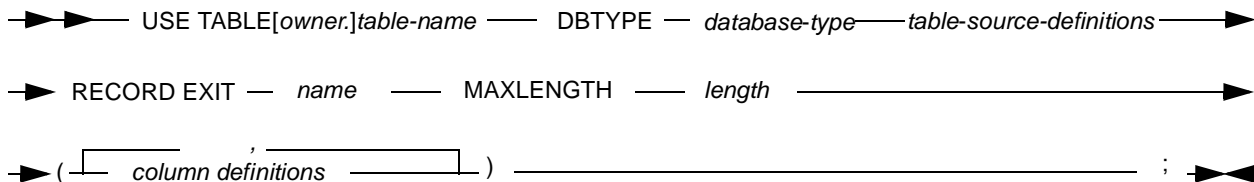
The USE TABLE statement consists of three basic components:

- Information that identifies the logical table name and a table source definition that associates the logical table to a physical database or file. This information is required.
- Column definitions. The column definition identifies the DB2 attributes associated with a column as well as database/file specific information. One or more column definitions must be supplied for a logical table.
- Recurring column information. If the columns defined for a logical table occur multiple times in the mapped physical database/file, tell eXadas to treat this recurring information as separate rows in the result sets returned from a query that references a recurring column(s). Specification of recurring column information is optional.

There is a different format for the table source definition for each source database type, and differing formats for the column definitions. The column definitions for a single USE TABLE statement must be separated by commas with a single pair of parentheses enclosing the entire set of definitions. All strings that contain embedded blanks or USE TABLE keywords must be enclosed in quotes. Quotes must be double (“ ”). All statements must include a terminating semi-colon (;).

The USE TABLE statement syntax is shown in [Figure 19: “USE TABLE Statement Syntax,”](#) and described in [Table 11, “USE Table Parameters and Descriptions.”](#)

**Figure 19: USE TABLE Statement Syntax**



**Table 11: USE Table Parameters and Descriptions**

Parameters	Descriptions
USE TABLE	Keywords that identify the statement. All subsequent parameters describe the logical table identified by <i>table-name</i> until the next USE TABLE, USE INDEX, DROP TABLE, or DROP INDEX keywords are encountered, or EOF on SYSIN is encountered.
owner	SQL authorization ID of the owner. If owner is not specified, the user ID used to run the Meta Data Utility is assigned as the owner. Maximum length is 8 characters.

**Table 11: USE Table Parameters and Descriptions**

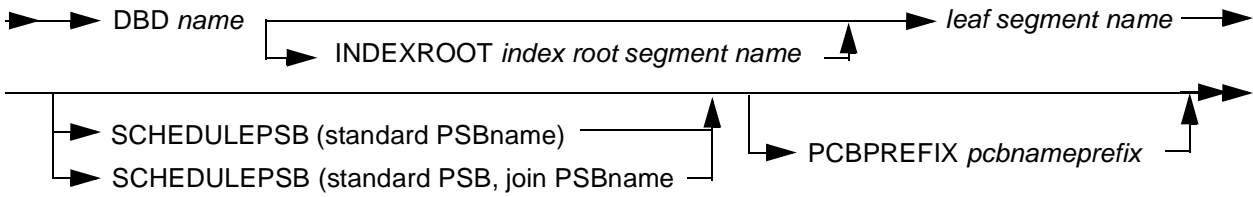
Parameters	Descriptions
table-name	Represents the name for the table you are creating. The maximum length is 18 characters. The combination of owner and table-name must be unique within a Meta Data Catalog. If owner is specified, the syntax is: <i>owner.table-name</i> .
DBTYPE	Keyword for the clause that identifies the database type.
database-type	Identifies the source database/file type. Valid values are IMS, VSAM, SEQUENTIAL, IDMS, ADABAS, and DATACOM.
table-source-definitions	Represents the set of parameters that defines the table source for the specified database/file type. See <a href="#">“Table Source Definitions,”</a> on page 151, for additional information.
RECORD EXIT	An optional keyword to indicate a record exit for this table definition.
name	The name of the record exit module.
MAXLENGTH	A required keyword if RECORD EXIT is specified.
length	The maximum length of the updated record.
column-definitions	Represents the set of parameters that define the column source for the specified table. See <a href="#">“eXadas Column Definition Syntax,”</a> on page 162, for additional information.

### Table Source Definitions

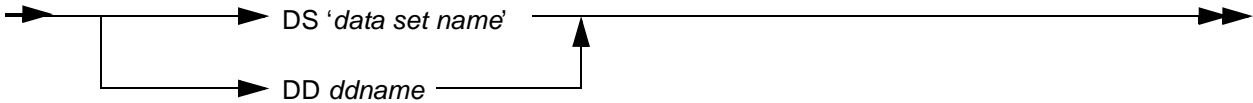
The following diagram describes the IMS, Sequential, VSAM, CICS VSAM, IDMS, ADABAS, and Datacom table source definitions syntax.

Figure 20: Table Source Definitions

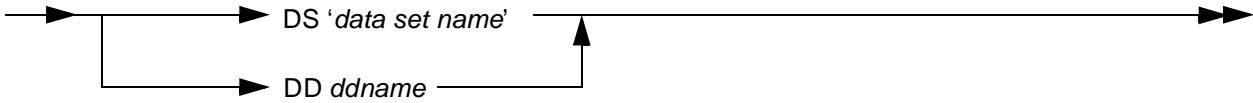
**IMS**



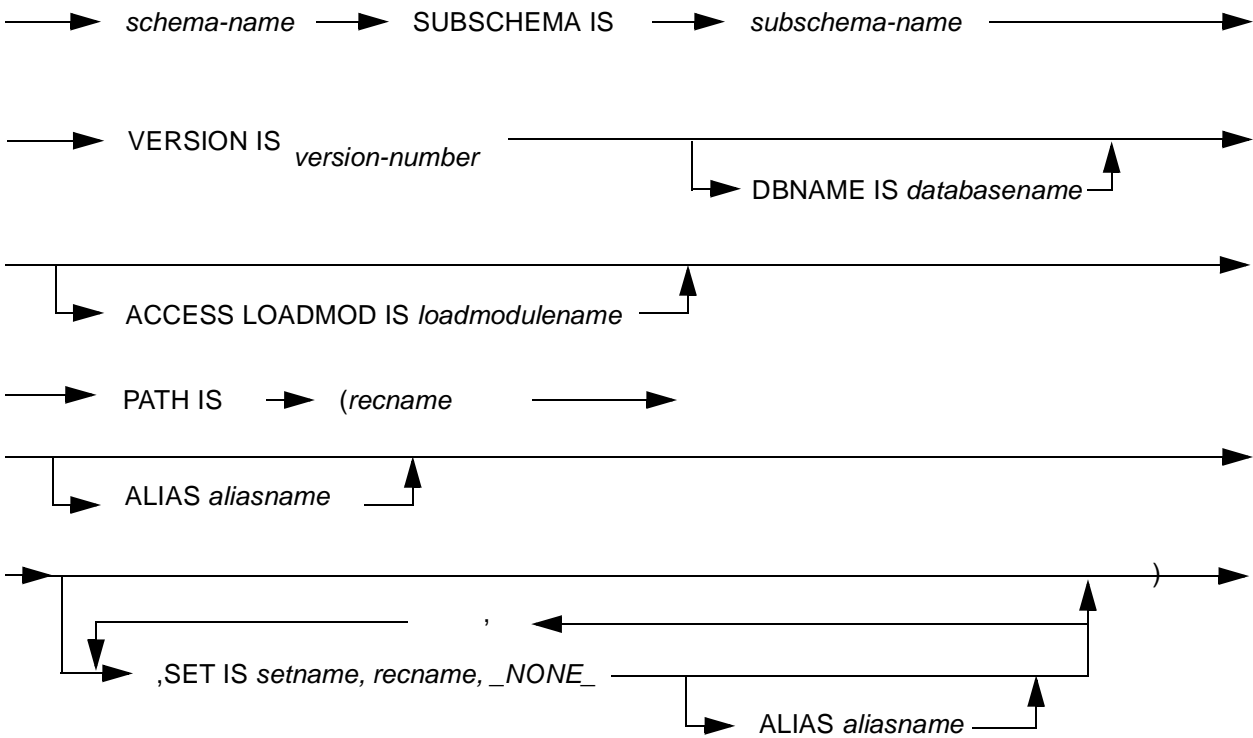
**Sequential**



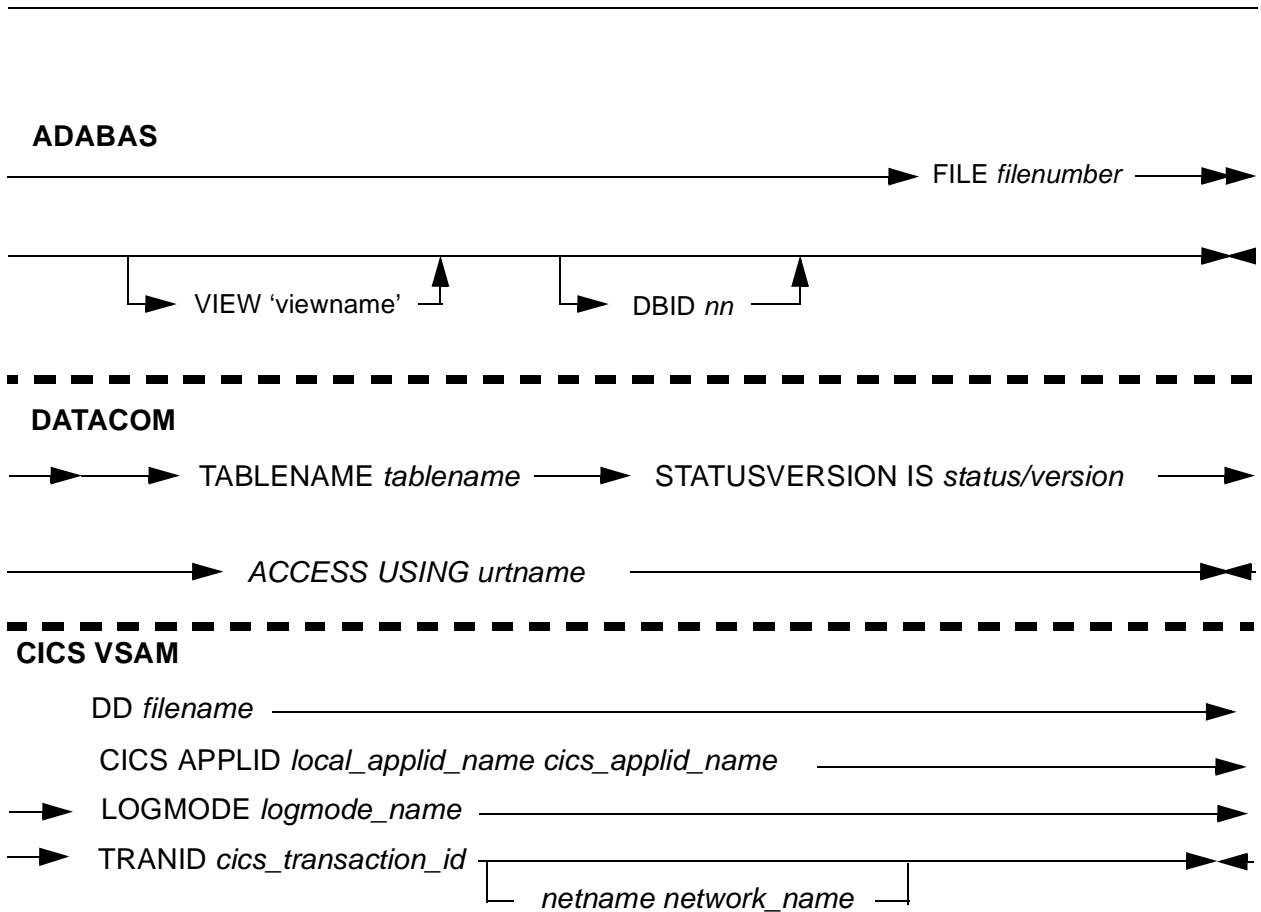
**VSAM**



**IDMS**







### IMS Table Source Definitions

The IMS table source definitions are part of the USE TABLE syntax. The following table describes the parameters and their descriptions.

**Table 12: IMS Table Source Definitions**

Parameter	Description
DBD name	Identifies the DBD name associated with the target IMS database. The DBD name must exist in the DBDLIB(s) referenced in the DBDLIB DD statement when the Meta Data Utility is executed.
INDEXROOT	An optional keyword that allows the specification of the root segment for mapping that utilizes an INDEX causing the IMS database hierarchy to be inverted.

**Table 12: IMS Table Source Definitions**

Parameter	Description
index root segment name	<p>The root of the IMS database hierarchy when accessing the table on a secondary index segment that is not the root of the database as defined in the DBD.</p> <p><b>NOTE:</b> All access to the database for the table must go through a secondary index on the specified segment name.</p>
leaf segment name	Represents the name of the segment farthest from the root segment on a single path for this table definition.
SCHEDULEPSB	An optional keyword that allows the specification of one or more PSB names to be used when the DRA interface is used to access IMS data. See <a href="#">“PSB Scheduling,” on page 109</a> , for more information.
standard PSBname	Specifies the name of the PSB that the DRA interface schedules to service a query that references the logical table name. If SCHEDULEPSB is specified, standard PSBname is required.
join PSBname	An optional parameter that specifies the name of a PSB that the DRA interface schedules to service a query when the table is referenced as part of a JOIN. In JOIN queries, this PSB will be scheduled in place of the standard PSB.
PCBPREFIX	An optional keyword that allows the specification of a PCB prefix to be used for PCB selection by name. See <a href="#">“PCB Selection Options,” on page 108</a> , for additional information.
pcbnameprefix	<p>A one to seven character name that specifies the prefix of a PCB name. Unique PCB names are generated by appending a sequence number to the prefix.</p> <p>If PCBPREFIX is specified, pcbnameprefix is required. The pcbnameprefix must be enclosed in double quotes if it is a numeric value or if the value begins with a number, for example 1 or 1A. It is optional to include double quotes for other string values, such as AB or A1.</p>

## Sequential Table Source Definitions

This section describes the Sequential table source definition syntax.

**Table 13: Sequential Table Source Definitions**

Parameter	Description
DS 'data set name'   DD ddname	<p>Identifies a Sequential file. You can specify either a <code>data set name</code> or a <code>ddname</code>. If you use a <code>data set name</code>, it must be enclosed in quotes, fully-qualified, and 1-44 alphanumeric characters. You can only specify Sequential data sets that already exist on the system as the Meta Data Utility can only access the file catalog information about existing Sequential data sets.</p> <p>A <code>ddname</code> must be 1-8 alphanumeric characters. If <code>ddname</code> was specified, a corresponding <code>DD</code> statement must be included in both the Meta Data Utility JCL and the Server in order for the file to be accessed successfully.</p> <p><b>NOTE:</b> You can also specify the name of a PDS member or reference a generation data set. For a PDS member the syntax is <code>DS 'data set name(member)'</code>. To reference a generation data set, the <code>DS</code> format must be used. A specification of <code>DS 'data set name(0)'</code> causes the current version to be accessed. A specification of <code>DS 'data set name(-n)'</code>, where <i>n</i> is a number specifying the relative number from the current version to be accessed. If no version number is specified, all versions of the generation data set are accessed.</p>

## VSAM Table Source Definition

This section describes the VSAM table source definition syntax.

**Table 14: VSAM Table Source Definitions**

Parameter	Description
DS 'data set name'   DD ddname	<p>Identifies a VSAM cluster. You can specify either a <code>data set name</code> or a <code>ddname</code>. If you use a <code>data-set-name</code>, it must be enclosed in quotes, fully-qualified, and 1-44 alphanumeric characters. You can only specify VSAM data sets that already exist on the system as the Meta Data Utility can only access the file catalog information about existing VSAM data sets.</p> <p>A <code>ddname</code> must be 1-8 alphanumeric characters. If <code>ddname</code> was specified, a corresponding <code>DD</code> statement must be included in both the Meta Data Utility JCL and the Server in order for the file to be accessed successfully.</p>

## IDMS Table Source Definitions

This section describes the IDMS table source definition syntax.

**Table 15: IDMS Table Source Definitions**

Parameter	Description
DBNAME IS <i>databasename</i>	<p>Defines a 1 to 8 character IDMS database name containing the subschema defined for the table. This parameter is optional. If <code>DBNAME</code> is omitted, the <code>DBNAME</code> will be determined by the default database name for the Server runtime environment. For more information on specifying default <code>DBNAME</code>s to IDMS, see the IDMS documentation for the <code>SYSCTL</code> and <code>SYSIDMS</code> JCL declarations and the generation of <code>IDMSOPTI</code> modules.</p>
schema-name	The IDMS schema.
SUBSCHEMA IS	A required keyword separating the schema-name and subschema name fields.
subschemaname	The IDMS subschema name from which data elements are extracted for mapping to subsequently defined columns.

**Table 15: IDMS Table Source Definitions**

Parameter	Description
VERSION IS	A required keyword separating the subschema name and version number fields.
version number	The IDMS schema version number that qualifies the schema name.
ACCESS LOADMOD IS <i>loadmodulename</i>	Defines a 1 to 8 character IDMS batch access load module to be used to communicate with IDMS. If not specified, the default load module name IDMS is used. This parameter can be used to specify a specially built IDMS access module that contains an IDMS OPTI module. The ACCESS LOADMOD specification may be used to specify an access module that points to a different central version of IDMS than the default for the Server environment. For more information on building IDMSOPTI modules, see the IDMS documentation.
PATH IS	A required keyword clause that identifies the start of the PATH clause.
ALIAS <i>aliasname</i>	An optional keyword by which the preceding record name may also be referenced. If the record name appears two or more times in a path, then all occurrences of the record name except one must have an alias-name. Each alias-name must be unique.
SET IS	An optional keyword clause identifying the existence of the setname and rename that follow it.
setname	An IDMS set name identifying a link or navigational path SET between the record name clause that proceeds the SET IS keyword phrase and the record name that follows it.

**Table 15: IDMS Table Source Definitions**

Parameter	Description
recname	An IDMS record name. The elements of this record name can be found by navigating through the preceding record name and set name.
_NONE_	<p>No set relationships exists between the two records mapped in the IDMS path. The use of _NONE_ is not recommended for tables that will be queried, as it produces a Cartesian product of all the records for each record type in the result set.</p> <p>For more information regarding the _NONE_ setting, see <a href="#">“SQL INSERT Considerations,” on page 363.</a></p>

### ***ADABAS Table Source Definitions***

This section describes the ADABAS table source definition syntax.

**Table 16: ADABAS Table Source Definitions**

Parameter	Description
filenumber	A required parameter that specifies the ADABAS file number for the table being defined.
viewname	An optional parameter that specifies the predict view name enclosed in single quotes ( ' '). This name is passed to the security exit for authorization if the SAF Exit is active.
DBID nn	<p>An optional parameter specifying the ADABAS database ID number. The default is 1.</p> <p>If the DBID information is in the Predict Dictionary, the ADABAS USE Statement Generator program automatically generates the DBID option in the Meta Data Grammar. The DBID specified on the ADARUN card for the ADABAS USE Statement Generator must be the Database ID of the Predict Dictionary file.</p>

## **Datacom Table Source Definitions**

This section describes the Datacom table source definition syntax.

**Table 17: Datacom Table Source Definitions**

<b>Parameter</b>	<b>Description</b>
TABLENAME	A required keyword that allows specification of a CA-DATACOM/DB table ENTITY-OCCURRENCE-NAME (table name).
tablename	Specifies the CA-DATACOM/DB ENTITY-OCCURRENCE-NAME of the table to which this Meta Data mapping applies.
STATUSVERSION IS	Required keywords that allow specification of the CA-DATACOM/DB status and version of the named table.
status/version	Specifies the status and version of the named table that is to be accessed from the CA-DATACOM/DB Datadictionary Service Facility. Only PROD, TEST, and HIST are allowed as valid status values. Non-current versions for these statuses are specified using version numbers <i>nnn</i> , <i>Tnn</i> , or <i>Hnn</i> . See the <b>CA-DATACOM/DB DSF Programmer Guide</b> for an explanation of "Specifying Status/Version."

**Table 17: Datacom Table Source Definitions**

Parameter	Description
ACCESS USING	Required keywords that allow specification of a CA-DATACOM/DB User Requirements Table.
urtname	<p>Specifies the name of a User Requirements Table that has been assembled and link-edited into an accessible load library. The User Requirements Table identifies which CA-DATACOM/DB tables can be accessed by CA-DATACOM/DB commands and must contain the three-character CA-DATACOM/DB table name and the CA-DATACOM/DB database ID for the table identified in the 'tablename' parameter, described above.</p> <p><b>NOTE:</b> Whenever a new table is to be mapped into the eXadas catalog, an appropriate URT must be created or updated, assembled, link-edited, and the load module name must be specified in the Meta Data Utility grammar for the new table. Failure to include the CA-DATACOM/DB tablename and database ID in the specified URT will result in an access failure while trying to process the new table.</p>

### ***CICS VSAM Table Source Definitions***

This section describes the CICS VSAM table source definition syntax.

**Table 18: CICS VSAM Table Source Definitions**

Parameter	Description
DD filename	The filename option identifies the CICS File Control Table (FCT) name for the VSAM file.
CICS APPLID <i>local_applid_name</i> <i>cicsapplid_name</i>	Identifies the CICS APPC application ID (LUNAME) for the eXadas Server and the APPC application ID (LUNAME) of the CICS address space that owns the VSAM file.
LOGMODE <i>logmode_name</i>	A 1 to 8 character name that identifies the logmode for the APPC connection to CICS.



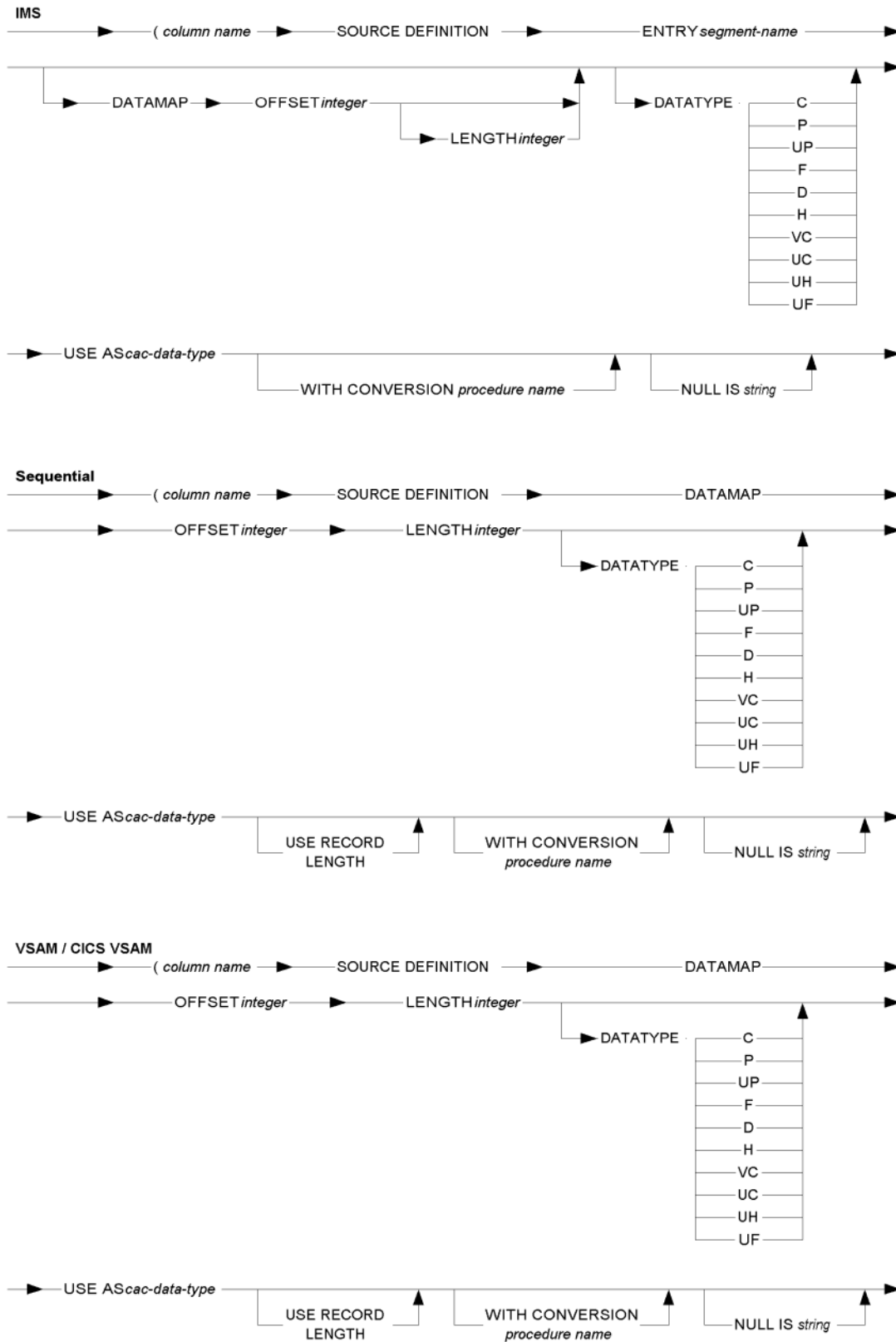
**Table 18: CICS VSAM Table Source Definitions**

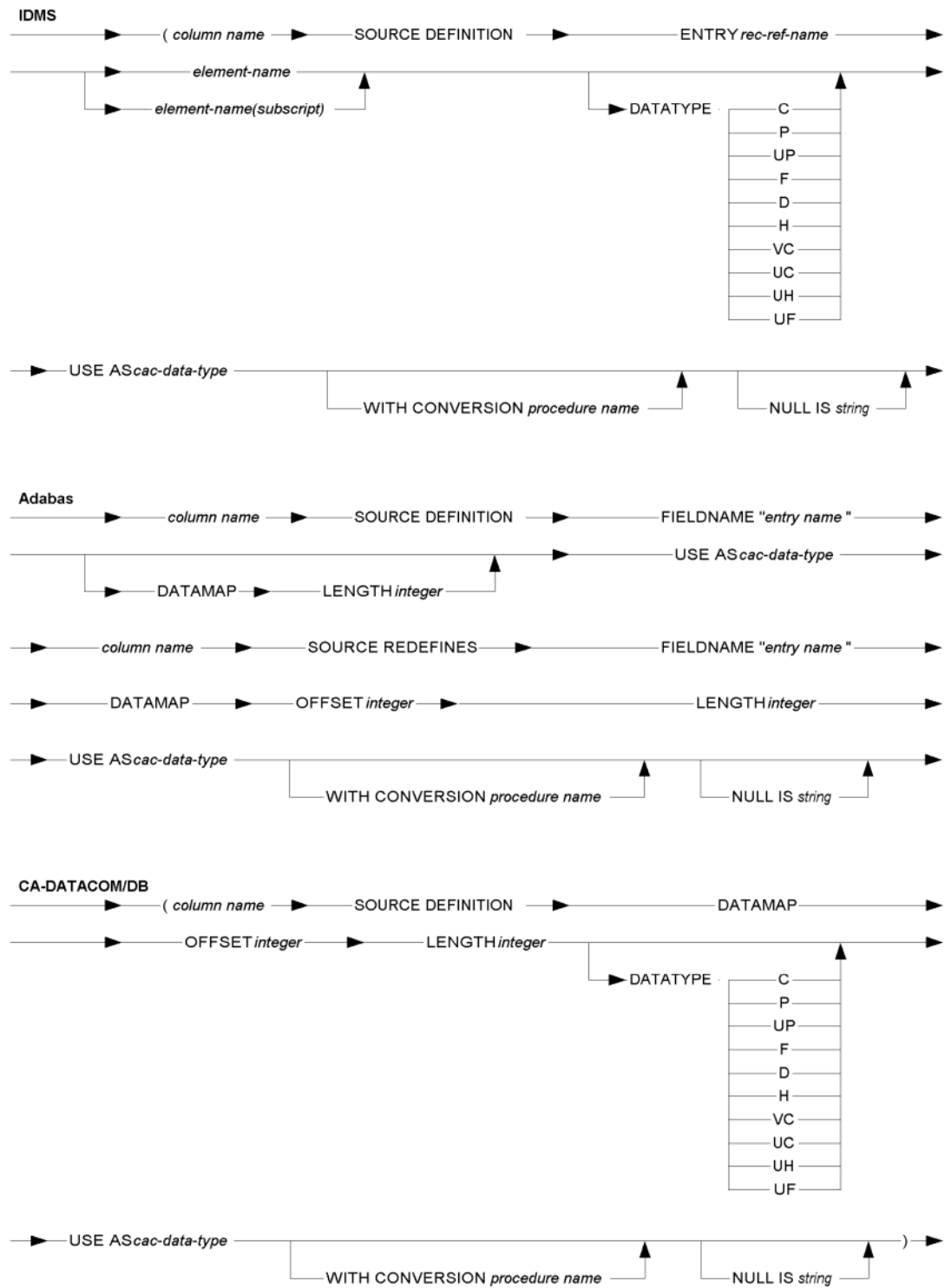
<b>Parameter</b>	<b>Description</b>
<i>TRANID cics_transaction_id</i>	A 1 to 4 character name that identifies the CICS transaction ID of the eXadas CICS VSAM Data Savant.
<i>netname network_name</i>	A 1 to 8 character name that identifies the name of the network where the CICS address space resides. This parameter is optional and only necessary when the CICS address space resides on a network other than the network where the eXadas Server resides.

### Column Definitions

This section describes the column definition syntax for mapping IMS, Sequential, VSAM, IDMS, ADABAS, and Datacom data.

Figure 21: eXadas Column Definition Syntax





This section describes the column definitions and their meanings.

**Table 19: Column Definitions**

Parameter	Definition
column-name	Represents the name of the column described by the statement. The column-name must conform to DB2 column naming conventions. The maximum length for a column-name is 30 characters. The column-name must be unique within the logical table.
SOURCE DEFINITION	Specifies the beginning of the column definition. All subsequent parameters in the statement describe the column identified by column-name.
ENTRY segment-name	Specifies the IMS segment name from which the column resides.
FIELDNAME “entry name” ISN	<p>A required parameter that specifies the ADABAS field definition name for the column. The entry name should be enclosed in double quotes (“ ”).</p> <p>If ISN is specified, it must be specified as USE AS INTEGER and when used in a query, it will return the ADABAS ISN of the record.</p>
DATAMAP	<p>Specifies the size and offset of the column specified. The DATAMAP clause must be coded with both of the following parameters:</p> <ul style="list-style-type: none"> <li>• <b>OFFSET</b> <i>integer</i>: <i>integer</i> represents the number of bytes from the start of the segment relative to zero.</li> <li>• <b>LENGTH</b> <i>integer</i>: <i>integer</i> represents the number of bytes in the field for all data types except graphic, where <i>integer</i> represents the number of double-byte characters.</li> </ul>
DATATYPE	<p>Defines the data type of the entry name. For more information on data types, see <a href="#">“eXadas-Supported Data Types,” on page 170</a>. Specify one of the following values:</p> <ul style="list-style-type: none"> <li>• C (character or signed zoned decimal),</li> <li>• P (packed decimal),</li> <li>• UP (unsigned packed decimal),</li> <li>• D (double word),</li> <li>• H (halfword),</li> <li>• F (fullword or FLOAT),</li> <li>• V (variable, including a two-byte length field).</li> <li>• UC (unsigned character),</li> <li>• UH (unsigned halfword), or</li> <li>• UF (unsigned fullword).</li> </ul>

**Table 19: Column Definitions**

Parameter	Definition
rec-ref-name	<p>An IDMS record type name containing the definition of the element-name that follows it. The record type name must have been mentioned in the PATH IS clause. IDMS allows the use of a circular path in which a particular record may appear multiple times. The rec-ref-name may refer to either a record or a reference, such as an alias-name. It is used to specify a unique occurrence of a record.</p>
element-name	<p>The IDMS record element name that was defined in the rec-ref-name record definition. The element-name may define any of the following:</p> <ul style="list-style-type: none"> <li>• an elemental data field,</li> <li>• a repeating (OCCURS) field, or</li> <li>• a group name.</li> </ul> <p>If the element-name is a group name, the SQL usage must be type CHAR.</p>
element-name(subscript)	<p>If the associated element-name is defined as a repeating (OCCURS) field, the subscript field is used to identify the occurrence from which data is extracted for the SQL column map. If the subscript field is part of a higher level repeating group, multiple subscript positions must be provided, each position must be separated from the next with a comma, the left-most position must refer to the highest level OCCURS clause, and the right-most subscript must refer to the associated element name. Up to three levels of subscription may be specified, for example, n1, n2, n3. All subscript values must be integers within the correct range for the corresponding OCCURS clause.</p>

**Table 19: Column Definitions**

Parameter	Definition
USE AS cac-data-type	<p>Indicates the supported data type used for the column. Specify one of the following values:</p> <ul style="list-style-type: none"> <li>• CHAR (integer),</li> <li>• VARCHAR (integer),</li> <li>• LONG VARCHAR,</li> <li>• INTEGER,</li> <li>• SMALLINT,</li> <li>• DECIMAL (precision-integer[,scale-integer]),</li> <li>• FLOAT (integer),</li> <li>• GRAPHIC (integer),</li> <li>• VARGRAPHIC (integer),</li> <li>• LONG VARGRAPHIC,</li> <li>• DATE “date-format” (Adabas), or</li> <li>• TIME “time-format” (Adabas).</li> </ul>
NULL IS string	<p>Optional parameter. Indicates that string is used to define the value that is interpreted as representing NULL in the source database. Specify character data enclosed in quotes or hexadecimal data enclosed in quotes and preceded by an x (for example, x ‘...’).</p> <p>If NULL IS is not specified on a column, the NULL indicator for the column is never set. When specified, the NULL IS value can be a maximum of 32 characters, including string delimiters. <i>string</i> is compared to the actual data field for the length of string, and if identical, the NULL indicator is set.</p> <p>For example, if a 10-byte character field has a NULL IS specification of NULL IS x’0000’, only the first two bytes of the field are used to determine if the field is NULL. The remainder of the field is ignored.</p> <p><b>NOTE:</b> For VARCHAR, LONG VARCHAR, VARGRAPHIC, and LONG VARGRAPHIC columns, the first byte after the length field is the byte compared to the NULL value. NULL IS on a VARCHAR treats VARCHAR as NULL is either the length field is 0 or the data in the VARCHAR matches the NULL IS specification.</p> <p>For UPDATE and INSERT, the NULL IS specification should define the string with the same length as the column definition.</p>

**Table 19: Column Definitions**

Parameter	Definition
WITH CONVERSION <i>procedure name</i>	Optional parameter. The field procedure, <i>procedure name</i> , is called to decode the column value in the result set when the applicable defined column is referenced in a WHERE clause.
USE RECORD LENGTH	An optional clause that, when specified, indicates that column length is obtained from the physical record length. This clause is used only for VARCHAR, LONG VARCHAR, VARGRAPHIC, and LONG VARGRAPHIC columns that begin at offset zero in a variable length record.

### **Record/Field Data Type Conversions**

You can write a field procedure to map IMS or IDMS data record fields to the data types supported by the Meta Data Utility.

The following table identifies the eXadas-supported mapping from native data types to SQL data types.

**Table 20: Data Type Mappings**

Data Type	eXadas Supported Data Types	Description
Character (VSAM, Sequential) C (IMS, CA-DATACOM/DB) A (Alphanumeric - ADABAS) DISPLAY (IDMS)	CHAR( <i>n</i> )  DECIMAL( <i>p</i> [, <i>s</i> ])        GRAPHIC( <i>n</i> )	CHAR data types are fixed-length character strings of length <i>n</i> , where $1 \leq n \leq 254$ .  DECIMAL data types are zoned decimal strings containing numeric data on which arithmetic comparisons or operations are likely to be performed.  <ul style="list-style-type: none"> <li><i>p</i> is the precision decimal, specifying the total number of digits,</li> <li><i>s</i> is the total number of digits to the right of the decimal.</li> </ul> Fixed-length DBCS strings of length <i>n</i> where $1 \leq n \leq 127$ . For GRAPHIC data types, <i>n</i> specifies the number of DBCS characters, not the amount of physical storage occupied by the field.

**Table 20: Data Type Mappings**

Data Type	eXadas Supported Data Types	Description
Double Float (VSAM, Sequential) F (ADABAS) COMP-2 (IDMS) L (CA-DATACOM/DB)	FLOAT( <i>n</i> )	Double-precision floating point; 64-bit; <i>n</i> - integer $22 \leq n \leq 53$ .
Fullword (VSAM, Sequential) I (ADABAS) COMP-4 (IDMS) PIC 9(8) B (4 bytes; CA-DATACOM/DB)	INTEGER	Fullword signed integer; 32-bit; no decimal point allowed.
Single Float (VSAM, Sequential) F (ADABAS) COMP-1 (IDMS) S (CA-DATACOM/DB)	FLOAT( <i>n</i> )	Single-precision floating point; 32-bit; <i>n</i> - integer $1 \leq n \leq 21$ .
H (VSAM, Sequential) I (ADABAS) COMP-4 (IDMS) PIC 9(4) B (2 bytes; CA-DATACOM/DB)	SMALLINT	Halfword signed integer; 16-bit ; no decimal point allowed.  ADABAS field can be 8-bits or 16-bits.
Packed (VSAM, Sequential) P (IMS, ADABAS) COMP-3 (IDMS) D (CA-DATACOM/DB)	DECIMAL( <i>p</i> [, <i>s</i> ])	Packed decimal value where: <ul style="list-style-type: none"> <li>• <i>p</i> is the precision decimal, specifying the total number of digits and</li> <li>• <i>s</i> is the total number of digits to the right of the decimal.</li> </ul>



**Table 20: Data Type Mappings**

Data Type	eXadas Supported Data Types	Description
Unsigned Packed (VSAM, Sequential) P (IMS, ADABAS)	DECIMAL( $up[,s]$ )	Unsigned Packed decimal value where: <ul style="list-style-type: none"> <li>• <math>up</math> is the precision decimal, specifying the total number of digits and</li> <li>• <math>s</math> is the total number of digits to the right of the decimal.</li> </ul>
N (Numeric - ADABAS, CA-DATACOM/DB)	CHAR( $n$ )  DECIMAL( $p[,s]$ )	CHAR data types are fixed-length character strings of length $n$ , where $1 \leq n \leq 29$ .  DECIMAL data types are zoned decimal strings containing numeric data on which arithmetic comparisons or operations are likely to be performed. <ul style="list-style-type: none"> <li>• <math>p</math> is the precision decimal, specifying the total number of digits,</li> <li>• <math>s</math> is the total number of digits to the right of the decimal.</li> </ul>

**Table 20: Data Type Mappings**

Data Type	eXadas Supported Data Types	Description
V (Variable length character or graphic field. Character strings longer than 254 are changed to LONG VARCHAR Data Type. Graphic strings longer than 127 are changed to LONG VARGRAPHIC Data Type).	VARCHAR( <i>n</i> )  LONG VARCHAR  VARGRAPHIC( <i>n</i> )  LONG VARGRAPHIC	Variable length character string, in which <i>n</i> is an integer $1 \leq n \leq 32704$ .  Variable length character string, for which the size is calculated. See “ <a href="#">eXadas-Supported Data Types</a> ,” following this table for information on calculating LONG VARCHAR.  Variable-length DBCS string where $1 \leq n \leq 127$ . The value of <i>n</i> specifies the number of DBCS characters. For example, VARGRAPHIC(10) specifies a column that occupies 20-bytes of storage.  Variable-length DBCS string where $1 \leq n \leq 16352$ . The value of <i>n</i> specifies the number of DBCS characters.

### ***eXadas-Supported Data Types***

The following table contains the eXadas-supported data types and their descriptions. The values in the eXadas-Supported Data Type column are used in the Column Definitions clause of the USE TABLE statements. They are used for the cac-data-type parameter in the USE AS keyword. The default mappings of source data into eXadas-supported data types are described under each individual database type.

**Table 21: eXadas-Supported Data Types**

eXadas-Supported Data Types	Description ( <i>n</i> is always a decimal integer)
INTEGER	Fullword signed hexadecimal, 32-bits, no decimal point. For Adabas, this can also specify a three-byte binary field.
SMALLINT	Halfword signed hexadecimal, 16-bits, no decimal point. For Adabas, this can also specify a one-byte binary field.

**Table 21: eXadas-Supported Data Types**

eXadas-Supported Data Types	Description ( <i>n</i> is always a decimal integer)
DECIMAL( <i>p</i> [, <i>s</i> ])	Packed decimal $1 \leq p \leq 31$ and $0 \leq s < p$ . Where <ul style="list-style-type: none"> <li><i>p</i> is the precision (total number of digits) and</li> <li><i>s</i> is the total number of digits to the right of the decimal point.</li> </ul>
FLOAT ( <i>n</i> )	Floating point $1 \leq n \leq 53$ . If $1 \leq n \leq 21$ , then single-precision. If $22 < n \leq 53$ , then double-precision.
CHAR ( <i>n</i> )	Fixed-length character string of length <i>n</i> where $1 \leq n \leq 254$ .
VARCHAR ( <i>n</i> )*	Variable-length character string where $1 \leq n \leq 254$ .
LONG VARCHAR*	Variable-length character string where the size is calculated internally. See the <i>IBM DB2 SQL Reference Guide</i> for information on calculating Long VARCHAR lengths.
GRAPHIC ( <i>n</i> )*	This is a fixed-length, double-byte character set (DBCS) string where $1 \leq n \leq 127$ . The value of <i>n</i> specifies the number of DBCS characters. For example, GRAPHIC(10) specifies a column that occupies 20 bytes of storage.
VARGRAPHIC ( <i>n</i> )*	Variable-length, DBCS string where the size is calculated internally. The value of <i>n</i> specifies the number of DBCS characters. For example, VARGRAPHIC(10) specifies a column that occupies 20 bytes of storage where $1 \leq n \leq 32704$ .
LONG VARGRAPHIC*	Variable-length, DBCS string where $1 \leq n \leq 16352$ . See the <i>IBM DB2 SQL Reference</i> for information on calculating LONG VARGRAPHIC lengths.
REAL	Floating point $1 \leq n \leq 21$ . Single-precision value.
DOUBLE PRECISION	Floating point $1 \leq n \leq 53$ . Double-precision value.

**Table 21: eXadas-Supported Data Types**

eXadas-Supported Data Types	Description ( <i>n</i> is always a decimal integer)
DATE “date-format”	This data type is supported only for Adabas and represents the Natural date system variable. The date-format indicates the format of the returned date field. Any combination of MM (month), MMM (name of month), DD (day of month), DDD (day of year), YY (year), YYYY (full year) along with other characters and spaces can be used to represent the format of the date.
TIME “time-format”	This data type is supported only for Adabas and represents the Natural time system variable. The time-format indicates the format of the returned time field. Any combination of MM (month), MMM (name of month), DD (day of month), DDD (day of year), YY (year), YYYY (full year), HH (hour), MI (minute), or SS (seconds) along with other characters and spaces can be used to represent the format of the time.

\* Optionally, these CAC\_DATA\_TYPES may include the USE RECORD LENGTH clause, which causes the length of the data to be used to create a single column from the variable data.

**WARNING:** Incorrect specification of signed vs. unsigned packed decimal fields or signed vs. unsigned halfwords and fullwords can result in incorrect result sets or degraded performance in IMS databases when packed decimal COLUMNS are referenced in SQL WHERE clauses.

### ***Packed Decimal Support***

eXadas supports both unsigned and signed packed decimal fields in database records. While all DECIMAL columns returned to eXadas clients are signed, COBOL applications commonly define packed fields (COMP-3 data items) that are always  $\geq 0$  as unsigned.

When mapping internal data types on SQL columns, it is extremely important to denote unsigned packed fields as unsigned by specifying DATATYPE UP. This is particularly important in IMS as incorrect definitions of packed fields can result in incorrect SQL result sets or degraded performance when qualifying queries on packed data items.

### ***Zoned Decimal Support***

Zoned Decimal data types are character data types consisting of only digits and an optional sign. This data type exists because COBOL supports a data element with a numeric picture clause having a USAGE type of DISPLAY. This allows the creation of numeric data, which can be used to perform arithmetic and can also be stored in a readable format.

The following COBOL formats of Zoned Decimal are supported by eXadas.

- UNSIGNED Numeric, specified as: `PIC 999 USAGE IS DISPLAY`.
- SIGNED Numeric, specified as: `PIC S999 USAGE IS DISPLAY`. The sign in this case is kept in the first four bits of the last byte in the field. For example, the value 123 (`x'F1F2F3'`) would be stored as `x'F1F2C3'` for +123 and `x'F1F2D3'` for -123. COBOL also allows the sign to be kept in the first or last byte of the field, or separate from the field as either a leading or trailing (+ or -) character.

The only external change required to support Zoned Decimal is in the Meta Data Grammar, which is mapped in the DataMapper. To define a Zoned Decimal field, change the SQL data type of the field to `DECIMAL(p,s)` instead of the default `CHAR(n)` and add a `DATATYPE C` for signed numbers and `DATATYPE UC` for unsigned numbers. See the *eXadas DataMapper Guide* for more information on Meta Data Grammar.

For example, a 6-byte Zoned Decimal field is defined to the DataMapper by specifying its internal length as 6 and the data type as `character`. However, instead of specifying its SQL data type as `CHAR(6)`, it is specified as `DECIMAL(6)`. This results in the client application seeing the data as SQL decimal and allows arithmetic operations and comparisons to be performed on the field.

DataMapper will also transform COBOL Zoned Decimal fields on an import to be SQL `DECIMAL` data types if either of the following conditions is true:

- The field is declared as having a sign, for example `PIC S999`.
- The field has implied decimal positions (`PIC 999V9`).

## **VARCHAR**

eXadas expects the first two bytes of a field mapped to a column defined as `VARCHAR` to contain a binary length indicator (LL). There are two types of length definitions:

- LL represents the length of the field, excluding the two bytes required for LL.
- LL represents the total length of the field, including the two bytes required for LL.

For eXadas to extract `VARCHAR` data from the target database correctly, the `USE TABLE` statements must account for the different field length definitions.

If the `VARCHAR` data in the target database has an LL field that excludes the two bytes required for LL (definition 1, above), the `USE` statement must specify the `LENGTH` parameter two bytes greater than the specification for the `USE AS VARCHAR` parameter.

For example:

```
USE TABLE CACEMP
DBTYPE DBB CACEMP EMPLOYEE
(
  DEPT SOURCE DEFINITION
```

```

        DATAMAP OFFSET 45 LENGTH 5
        USE AS VARCHAR(3)
    )

```

If the data in column DEPT is “CAC” and the USE statement in Figure 9 is used, eXadas assumes the following is in the target database:

LL	Data
0003	CAC

If the VARCHAR data in the target database has an LL field that includes the two bytes required for LL, the USE statement must specify the LENGTH parameter equal to the “USE AS VARCHAR” specification. For example:

```

USE TABLE CACEMP
DBTYPE DBB CACEMP EMPLOYEE
(
    DEPT SOURCE DEFINITION
        DATAMAP OFFSET 45 LENGTH 5
        USE AS VARCHAR(5)
)

```

If the data in column DEPT is “CAC,” and the USE statement is used, eXadas assumes the following is in the target database:

LL	Data
0005	CAC

The record in the target database is translated by eXadas as follows when it is returned to the eXadas application:

LL	Data
0003	CAC

In the following example, LENGTH and USE AS VARCHAR have the same value.

```

OFFSET 0 LENGTH 30000
USE AS VARCHAR(30000)

```

In the previous example, the LL field contains the length of the data plus the LL field, so two bytes are subtracted when returning data to the application.

In the next example, LENGTH and USE AS VARCHAR differ by two bytes.

```

OFFSET 0 LENGTH 30000
USE AS VARCHAR(29998)

```

In the previous example, the LL field contains the size of the data only, so its value is returned to the application as-is.

### **LONG VARCHAR**

If a VARCHAR definition exceeds 254 bytes, eXadas converts the data type to LONG VARCHAR. With respect to the LL field, LONGVARCHAR is handled like VARCHAR.

The following is an example of a DB2 compatible LONG VARCHAR where the LL field contains the size of the data only, so the value is returned to the application as-is.

```
OFFSET 0 LENGTH 30000
USE AS LONG VARCHAR
```

### **VARGRAPHIC**

eXadas expects the first two bytes of a field mapped to a column defined as VARGRAPHIC to contain a binary length indicator (LL). There are two types of length definitions:

- LL represents the length of the field in bytes, excluding the two bytes required for LL.
- LL represents the total length of the field in bytes, including the two bytes required for LL.

**NOTE:** The LL field is converted from a length in bytes to a length in DBCS.

For eXadas to extract VARGRAPHIC data from the target database correctly, the USE TABLE statements must account for the different field length definitions.

If the VARGRAPHIC data in the target database has an LL field that excludes the two bytes required for LL (definition 1, above), the USE statement must specify the LENGTH parameter one DBCS character greater than the specification for the USE AS VARGRAPHIC parameter.

In the following example, LENGTH and USE AS VARGRAPHIC have the same value.

```
OFFSET 0 LENGTH 15000
USE AS VARGRAPHIC(15000)
```

In the previous example, the LL field contains the length of the data in bytes plus the LL field, so two bytes are subtracted and the result is divided by two when returning data to the application.

In the next example, LENGTH and USE AS VARGRAPHIC differ by one graphic character (two bytes).

```
OFFSET 0 LENGTH 15000
USE AS VARGRAPHIC(14999)
```

In the previous example, the LL Field contains the size of the data only in bytes, so its value is divided by two and returned to the application as-is.

### **LONG VARGRAPHIC**

If a VARGRAPHIC definition exceeds 127 bytes, eXadas converts the data type to LONG VARGRAPHIC. With respect to the LL field, LONG VARGRAPHIC is handled like VARGRAPHIC.

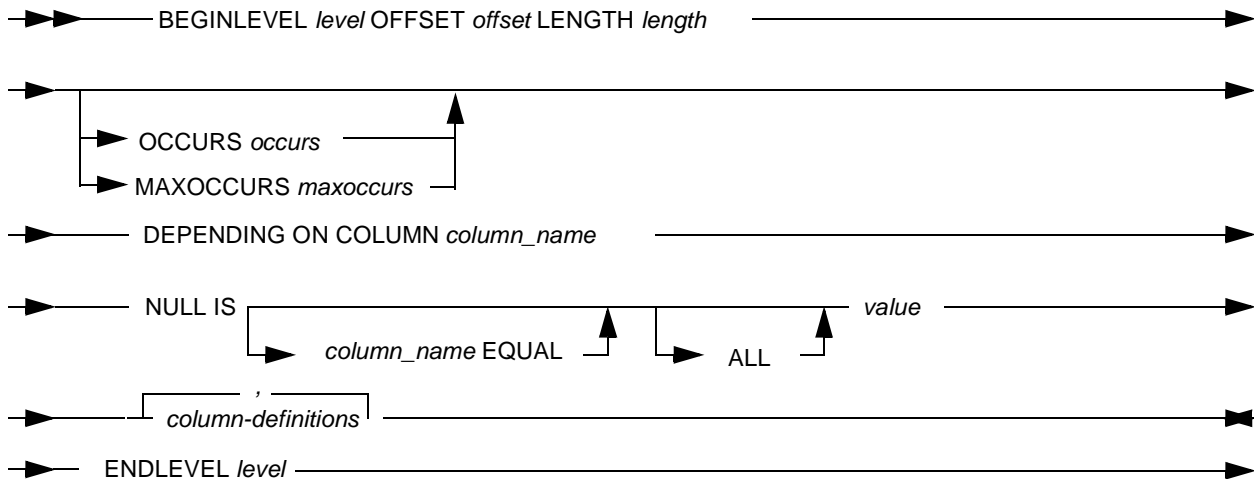
The following is an example of LONG VARGRAPHIC where the LL field contains the size of the data only in bytes. It's value is divided by 2 and returned to the application as-is.

```
OFFSET 0 LENGTH 15000
USE AS LONG VARGRAPHIC
```

## **Record Arrays**

When defining columns that map data that repeats in the physical database/file to be accessed, the columns can be enclosed within ENDLEVEL statements. The BEGINLEVEL/ENDLEVEL statement syntax is shown in the following figure.

**Figure 22: BEGINLEVEL and ENDLEVEL Statement Syntax**





The following table describes the parameters for the BEGINLEVEL statement.

**Table 22: BEGINLEVEL Definitions**

Parameter	Description
BEGINLEVEL	<p>A parameter that identifies that a group of repeating columns is being defined. If the record array repeats a fixed number of times, then multiple BEGINLEVEL statements can be specified.</p> <p>More than one record array of fields can be defined providing that the arrays are not OCCURS DEPENDING ON types.</p>
level	<p>A required parameter that specifies the nesting level. For example, if there is one group of repeating columns in the record, those fields would be defined as level 1 (the non-repeating columns are assumed to be at level 0). For columns that recur a fixed number of times, you can nest BEGINLEVEL statements. For example, if you have a set of columns that repeat 10 times and one of those 10 columns can also repeat 10 times, then you would define a BEGINLEVEL block for the first set of columns and assign them a level of 1. For the column within the level 1 group that repeats 10 times, bracket it with a BEGINLEVEL/ENDLEVEL statement and identify its level number as 2.</p>
OFFSET integer	<p>A required clause where integer is the number of bytes from the start of the recurring data relative to the current offset in the mapping. For example, for a level 1 specification, the offset is relative to the beginning of the record. For a level 2 specification (or higher) it is relative to the beginning of the offset within the “parent” BEGINLEVEL block.</p> <p>All offsets are specified relative to zero.</p> <p>All of the columns that are mapped within a BEGINLEVEL/ENDLEVEL block are mapped at relative offset zero within the block.</p>
LENGTH length	<p>A required clause where length is the total length (number of bytes) of the columns that are defined in the BEGINLEVEL/ENDLEVEL group.</p>
OCCURS integer	<p>An optional clause that specifies the number of times that the group of columns occurs. When specified, all occurrences of the group are considered as valid and no NULL checking is performed. Either an OCCURS or MAXOCCURS parameter must be specified on a BEGINLEVEL statement.</p>
MAXOCCURS maxoccurs	<p>An optional clause that specifies the maximum number of times that the group of columns occurs if none of the occurrences are NULL. Either an OCCURS or MAXOCCURS parameter must be specified on a BEGINLEVEL statement.</p>

**Table 22: BEGINLEVEL Definitions**

Parameter	Description
DEPENDING ON COLUMN column-name	An optional clause that specifies the name of a column that identifies the number of occurrences that exist in an individual record. This specification is used to define a varying number of occurrences situation. The column-name specified must have a numeric data type associated with it and must be defined before the BEGINLEVEL statement in the Meta Data Grammar.
NULL IS	An optional clause that specifies the name of a column within the BEGINLEVEL/ENDLEVEL block that is to be tested against the value to determine if a specific occurrence is NULL.  For DataMapper, NULL IS has a 32-byte limit. However, if only one byte is specified and the data retrieved matches the value indicated for NULL IS, NULL IS is turned on. If the data retrieved does not match the value indicated for NULL IS, NULL IS is never turned on.
column-name	An optional column name used to specify the NULL IS value.

**Table 22: BEGINLEVEL Definitions**

Parameter	Description
EQUAL value	<p>An optional clause that specifies a comparison value that is used to determine if an occurrence of the repeating group is NULL. This value can either be specified as a character string surrounded by quotes ('0000000') or as a hexadecimal string prefixed with X and surrounded by quotes (X'00000000').</p> <p>The value can be up to 32 characters in length. The comparison is made for the length of the value in determining whether an occurrence of the group is NULL. If a column-name specification is supplied, the contents of column-name is compared to determine whether an occurrence is NULL. If a NULL IS specification is not supplied, the comparison is performed against the beginning of the recurring data for each occurrence for the length specified on the LENGTH parameter.</p> <p>The EQUAL value and EQUAL ALL value specifications are mutually-exclusive.</p>
EQUAL ALL value	<p>An optional clause that specifies a comparison value that is used to determine if an occurrence of the repeating group is NULL. Only a single character is specified in value. Value can either be specified as a character string surrounded by quotes ( ' ') or as a hexadecimal string prefixed with X and surrounded by quotes (X'00').</p> <p>If a column-name specification is supplied, the entire content of the column is compared against <i>value</i> to determine whether an occurrence is NULL. If a column-name specification is not supplied, the comparison is performed against the beginning of the recurring data for each occurrence for the length specified on the LENGTH parameter.</p> <p>The EQUAL ALL value and EQUAL value specifications are mutually-exclusive.</p>

The following table describes the parameters for the ENDLEVEL statement.

**Table 23: ENDLEVEL Parameter Definitions**

Parameter	Description
ENDLEVEL	A required parameter that identifies the end of a group of repeating columns.
level	A required parameter that specifies the nesting level that is being ended. The ENDLEVEL and BEGINLEVEL level numbers must be paired together and the level numbers must match.

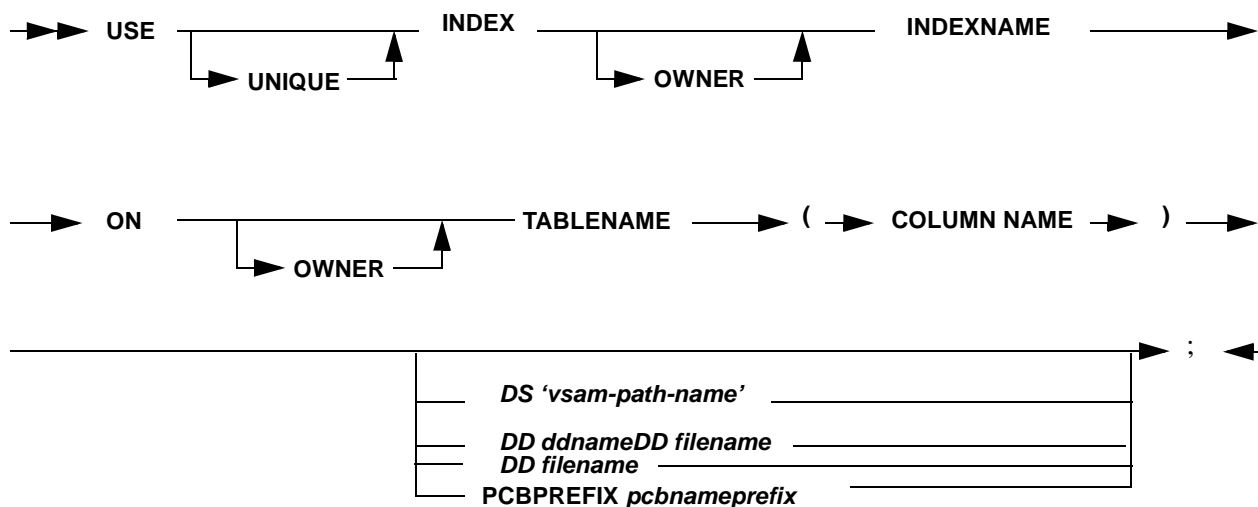
## USE [UNIQUE] INDEX Statement Syntax

The USE INDEX statement is used to define an index on a logical table. Index information can only be defined for an existing logical table definition. The information populated in the SYSIBM.SYSINDEXES and SYSIBM.SYSKEYS tables is used to optimize physical access to the database/file associated with the logical table for which the index is defined.

**NOTE:** eXadas does not actually index data, it only utilizes existing indexes on the data. Logical indexes defined in the DataMapper must match existing physical indexes placed on the underlying database system itself.

The USE INDEX statement syntax is shown in the following diagram.

**Figure 23: USE INDEX Statement Syntax**



The USE INDEX statement may be used to define the primary sequence field on a HIDAM database or XDFLDs for either HIDAM or HDAM databases. When defining an IMS index, the columns specified in the USE INDEX statement must match either the sequence field for a HIDAM root segment or the SRCH fields defined on an XDFLD statement in the DBD. The Meta Data Utility validates all USE INDEX statements against the DBD by matching up column offset and length information against the offset and length information defined for sequence and SRCH fields in the DBD. This matching is only performed against the root segment for the defined table. The root is either specified by the INDEXROOT in the USE TABLE statement or defaulted to the DBD ROOT if no INDEXROOT is specified.

The column(s) specified in a USE INDEX statement may sub-define sequence or SRCH fields in the DBD if desired. For example, a primary HIDAM sequence field defined as 8 bytes may be mapped as two 4-character columns.

**NOTE:** The order of the columns in a USE INDEX definition is significant to the matching process. Columns that match sequence or SRCH fields but are in the incorrect order will be flagged as an error in the Meta Data Utility.

The following table describes the parameters for the USE INDEX statement.

**Table 24: USE INDEX Parameter Descriptions**

Parameter	Description
USE [UNIQUE] INDEX	A required parameter that identifies the start of an index definition. If UNIQUE is specified it indicates that the key(s) the index is defined for are all unique values in the database/file.
Owner	SQL authorization ID of the index owner. If an owner is not specified the user ID that was used to run the Meta Data Utility is assigned as the owner. The maximum length is 8 characters.
index-name	A required parameter that specifies the name of the index. The combination of owner.index-name must be unique within a Meta Data Catalog. If you specify an owner the syntax is owner.index-name.  The maximum length is 18 characters.
ON owner.table-name	Required clause that specifies the name of the logical table definition that the index is being defined for. The table-name must take the form of owner.table-name. The table-name must already exist in the Meta Data Catalog.
column-name	Required parameter that identifies the logical table column(s) that make up the index. If the index is composed of more than one column in the logical table, the column-names must be separated by commas. The column names specified must exist in the logical table specified in the table-name clause.
DS 'vsam-path-name'	VSAM-specific optional clause used to specify the path name of a VSAM alternate index. Use of the VSAM path is determined at SQL Query time and is based upon the indexed columns in the SQL WHERE clause.
DD ddname	Optional clause used to specify the DD name of a VSAM alternate index path data set that should be used to access the logical table. Use of the VSAM path is determined at SQL Query time and is based upon the indexed columns in the SQL WHERE clause. The DD clause is mutually-exclusive with the DS clause.  When the DD clause is specified a DD name referencing the VSAM alternate index path must be supplied in the Meta Data Utility and the Server JCL.
DD filename	The CICS file control name. Use of the VSAM path is determined at SQL Query time and is based upon the indexed columns in the SQL WHERE clause.

**Table 24: USE INDEX Parameter Descriptions**

Parameter	Description
PCBPREFIX	An optional keyword that allows the specification of a PCB prefix to be used for PCB selection-by-name.
pcbnameprefix	A one-to-seven character name that specifies the prefix of a PCB name. Unique PCB names are generated by appending a sequence number to the prefix.  If PCBPREFIX is specified, pcbnameprefix is required. The pcbnameprefix must be enclosed in double quotes (“ ”) if it is a numeric value or if the value begins with a number, for example 1 or 1A. It is optional to include double quotes for other string values, such as AB or A1.

### Defining VSAM Indexes

The USE INDEX statement may be used to define the primary index on a VSAM KSDS or an alternate index on a VSAM KSDS or ESDS. The COLUMNS referenced in the USE INDEX definition must match the index you are mapping, based on offset and length values. The order of the columns must also be contiguous from the first to last character position of the indexed data.

When mapping indexes to VSAM alternate indexes, you must also specify the PATH data set name for the alternate index. Do not specify the alternate index data set name as the PATH data set name is required to correctly utilize the index at runtime. If you use the DD option on a VSAM alternate index definition, you must allocate the PATH data set in the Meta Data Utility and Server JCL for correct index access.

**NOTE:** When defining indexes on the VSAM file mapped to CICS, the DD option must be used and the name provided in the DD option must be the CICS file control table (FCT) definition for the VSAM alternate index.

### Defining IMS Indexes

If you define multiple overlapping COLUMN mappings to a single IMS index, you must also define a separate USE INDEX definition for each mapping to ensure the best index optimization at runtime.

## DROP TABLE Statement Syntax

The DROP TABLE statement removes an existing table from the system catalogs. All information, including indexes and views, for the table, columns, and indexes is removed from the Meta Data Catalog. This statement is required before attempting to replace an existing table with a table having the same name. The syntax follows.

Figure 24: DROP TABLE Statement Syntax

➡➡➡ DROP TABLE [*owner.*]*table-name*; ➡➡➡

Table 25: DROP TABLE Parameters and Descriptions

Parameter	Description
DROP TABLE	Statement that removes an existing table from the Meta Data Catalogs. This statement is required when replacing an existing table with a table of the same name.
owner	SQL authorization ID of the table owner. If an owner is not specified, then the user ID that was used to run the Meta Data Utility is assigned as the owner. The maximum length is 8 characters.
table-name	Name of the table to be dropped. The syntax is owner.table-name.

## DROP INDEX Statement Syntax

The DROP INDEX statement removes an existing index from the Meta Data Catalogs. All information for the index is removed from the SYSIBM.SYSINDEXES and SYSIBM.SYSKEYS tables. This statement is required before attempting to replace an existing index with an index having the same name. The syntax follows.

Figure 25: DROP INDEX Statement Syntax

➡➡➡ DROP INDEX [*owner.*]*index-name*; ➡➡➡

Table 26: DROP INDEX Parameters and Descriptions

Parameters	Description
DROP INDEX	Statement that removes an existing index from the Meta Data Catalogs. This statement is required when replacing an existing index with an index of the same name.
owner	User ID of the table owner. The syntax is owner.index-name.
index-name	Name of the index to be dropped.

# DB2 Grammar

The DB2 grammar is supplied in a standard 80-byte fixed format text file. The text file contains free form statements that identify the logical DB2 table, DB2 view, or DB2 index to be imported into the eXadas Meta Data Catalog. Also possible is the removal of DB2 table, DB2 view, or DB2 index definitions that were imported previously into the eXadas Meta Data Catalog. The grammar uses “keyword value” syntax and can span multiple lines. Comments may also be included within the grammar.

Comments are specified using the following format:

```
/*your comment here*/
```

Anything between the delimiters (`/* */`) is treated as a comment. The comment may span multiple lines.

**NOTE:** If you are editing the grammar file using an editor like ISPF, ensure that NUM OFF is set in the edit profile when editing the grammar.

The following statements can be specified in the DB2 grammar:

- **CONNECT TO DB2:** Identifies the DB2 subsystem that will be queried to obtain DB2 table, DB2 view, and DB2 index definitions for import into the eXadas Meta Data Catalog. The connection to DB2, done as the result of this statement, prevails until explicitly changed by another **CONNECT TO DB2** statement or until the end of the DB2 grammar processing.
- **IMPORT DB2 TABLE/VIEW:** Adds a logical DB2 table or view to the eXadas Meta Data Catalog. The eXadas Meta Data Catalog system tables, **SYSIBM.SYSTABLES**, and **SYSIBM.SYSCOLUMNS** are populated with information, retrieved from the DB2 Meta Data Catalog, describing the logical table or view identified in the statement. The imported object is identified as a table within the eXadas Meta Data Catalog. Connection to DB2 is required for an **IMPORT DB2 TABLE/VIEW** statement to process successfully.
- **IMPORT DB2 INDEX:** Adds a DB2 index to the eXadas Meta Data Catalog. The eXadas Meta Data Catalog system tables, **SYSIBM.SYSINDEXES**, and **SYSIBM.SYSKEYS** will be populated with information, retrieved from the DB2 Catalog, describing the index identified in the statement. The logical table or view definition to which the index applies must be present in the eXadas Meta Data Catalog before an **IMPORT DB2 INDEX** statement can be specified. Connection to DB2 is required for an **IMPORT DB2 INDEX** statement to process successfully.

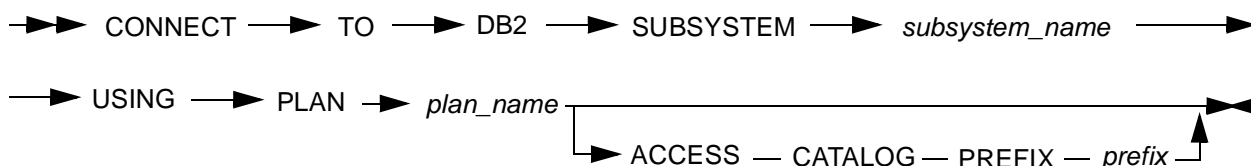
The syntax for each of these statements is described in the sections that follow.



## CONNECT TO DB2 Statement Syntax

The following diagram illustrates the CONNECT TO DB2 statement syntax.

**Figure 26: CONNECT TO DB2 Statement Syntax**



The following table describes the CONNECT TO DB2 statement parameters and descriptions.

**Table 27: CONNECT TO DB2 Statement Parameters and Descriptions**

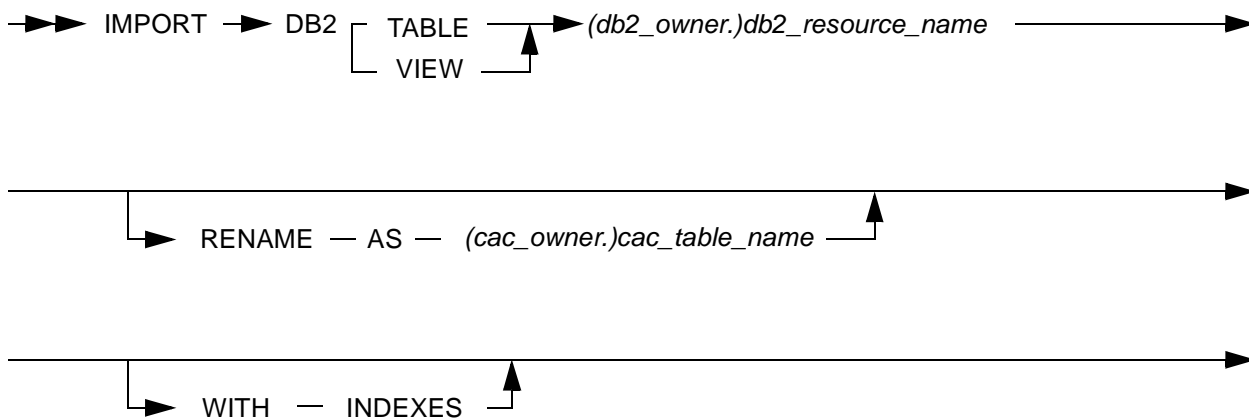
Parameter	Description
CONNECT TO DB2	Keywords that identify the statement. All subsequent parameters describe the DB2 system and catalog that is queried for information during an IMPORT operation. The identified DB2 system and catalog is used until it is explicitly changed by another CONNECT TO DB2 statement or until the Meta Data Utility terminates.
SUBSYSTEM	Required keyword for the clause that identifies the DB2 subsystem.
subsystem_name	Identifies the DB2 subsystem to which the connection will be made. Subsystem name cannot exceed 4 characters in length. The characters permitted in subsystem name vary with the version of DB2. See the DB2 documentation for further details. This parameter is required. No default value is supplied.
USING PLAN	Required keywords for the clause that identifies the DB2 application plan.
plan_name	Plan name cannot exceed 8 characters in length and the first character must be alphabetic. This parameter is required. No default value is supplied.  <b>NOTE:</b> Accessing DB2 data requires binding an application plan for use by the DB2 Call Attach Facility (CAF) service. eXadas includes the DB2 database request module (DBRM) required for creating the necessary plan. You may give the plan whatever name you want based on site-installation standards.  For more information on binding plans, see the appropriate IBM DB2 documentation.

**Table 27: CONNECT TO DB2 Statement Parameters and Descriptions**

Parameter	Description
ACCESS CATALOG PREFIX	Optional keywords for the clause that identifies the prefix of the DB2 catalog. Do not enter this keyword phrase if you wish to accept the default value SYSIBM as the prefix to be used.
prefix	Required parameter that is part of the optional 'ACCESS CATALOG PREFIX' clause. It supplies the prefix to use when accessing the DB2 catalog. If importing a table or view, prefix is used to access the DB2 catalog tables 'prefix.SYSTABLES' and 'prefix.SYSCOLUMNS'. If importing an index, prefix is used to access the tables mentioned above as well as the DB2 catalog tables 'prefix.SYSINDEXES' and 'prefix.SYSKEYS'.

## IMPORT DB2 TABLE Statement Syntax

The following diagram illustrates the IMPORT DB2 TABLE Statement Syntax.

**Figure 27: IMPORT DB2 TABLE Statement Syntax**

The following table describes the IMPORT TO DB2TABLE statement parameters and descriptions.

**Table 28: IMPORT DB2 TABLE Statement Parameters and Descriptions**

Parameter	Description
IMPORT DB2 TABLE IMPORT DB2 VIEW	Keywords that identify the statement. All subsequent parameters describe the logical DB2 resource to be imported and the processing options to be performed when storing that resource in the eXadas Meta Data Catalog. Options allow renaming the resource and the automatic importing of all related DB2 indexes.

**Table 28: IMPORT DB2 TABLE Statement Parameters and Descriptions**

db2_owner	Optional parameter specifying the SQL authorization ID (CREATOR) of the DB2 resource to be imported. If the db2_owner is not specified, the user ID used to execute the Meta Data Utility is assigned as the db2_owner. This parameter value cannot exceed 8 characters in length.
db2_resource_name	Required parameter that identifies the DB2 resource (NAME) to be imported into the eXadas Meta Data Catalog. This parameter value cannot exceed 18 characters in length. The combination of db2_owner and db2_resource_name must identify a resource that currently exists within the DB2 catalog.
RENAME AS	Optional keywords identifying the clause that allows renaming a resource during the import operation. If this clause and its related parameters (cac_owner and cac_table_name) are omitted, the resource will be stored in the eXadas Meta Data Catalog using db2_owner as the CREATOR and db2_resource_name as the NAME. If this clause is specified, its related parameters describe the CREATOR and NAME that will be used when the resource is stored in the eXadas Meta Data Catalog. In either case, the full name (CREATOR.NAME) must be unique within the eXadas Meta Data Catalog.
cac_owner	Optional parameter that is part of the optional 'RENAME AS' clause. It identifies the SQL authorization ID (CREATOR) of the table to be recorded in the eXadas Meta Data Catalog. If the cac_owner is not specified, the user ID used to execute the Meta Data utility is assigned as the cac_owner. This parameter value cannot exceed 8 characters in length.
cac_table_name	Required parameter that is part of the optional 'RENAME AS' clause. It identifies the table (NAME) to be recorded in the eXadas Meta Data Catalog. This parameter value cannot exceed 18 characters in length. The combination of cac_owner and cac_table_name must be unique within the eXadas Meta Data Catalog.
WITH INDEXES	Optional parameter indicating that all DB2 indexes, if any, related to the DB2 resource being imported shall be imported automatically when the DB2 source import has been successfully completed. Automatic import of related indexes does not allow renaming of indexes during the import operation. If the full index name (CREATOR.NAME) in the DB2 catalog does not represent a unique name in the eXadas Meta Data, the automatic import of the like-named index will fail. Every related index in the DB2 catalog will be processed for import. Import failures due to like-name indexes will not affect the processing of other related indexes. To rename an index during import, use the IMPORT DB2 INDEX statement.

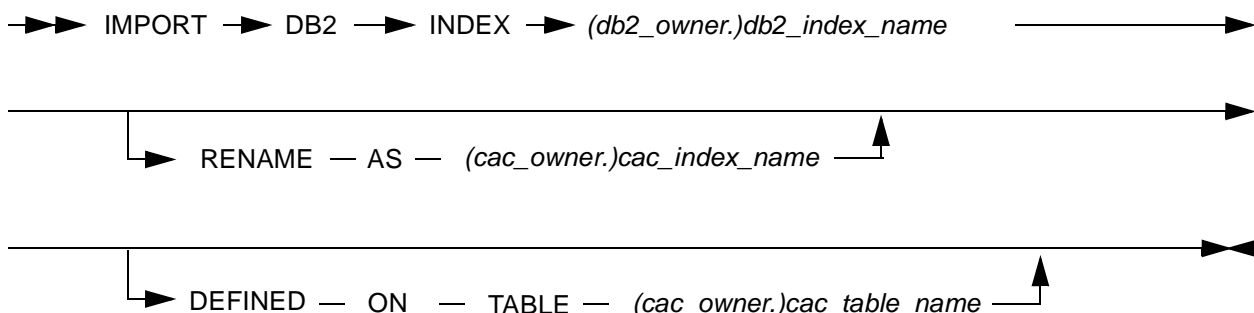
The following processing conditions and considerations should be taken into account when using the IMPORT DB2 TABLE Statement.

- The table name recorded in the eXadas Meta Data Catalog for any imported DB2 object cannot be SYSTABLES, SYSCOLUMNS, SYSINDEXES, SYSKEYS, or SYSTABAUTH.
- If a DB2 VIEW object is identified in an IMPORT DB2 TABLE statement, the import will continue to process. Upon successful completion of that import, a warning message is issued to note that a view was actually imported.
- If a DB2 TABLE object is identified in an IMPORT DB2 VIEW statement, the import will continue to process. Upon successful completion of that import, a warning message is issued to note that a table was actually imported.
- If a like-named table currently exists in the eXadas Meta Data Catalog, that table must be dropped (DROP TABLE) or a new name (RENAME AS) must be specified for the import to be successful.
- If WITH INDEXES and RENAME are both specified, the DB2 object imported is renamed. Any associated indexes that are imported are adjusted to reflect the new (rename) table creator and table name, but the index creator and index name are not modified. If renaming the actual index is required, import the indexes individually using and IMPORT DB2 INDEX statement.
- If WITH INDEXES is specified, each index that is successfully imported into the eXadas Meta Data Catalog is identified with an informational message.

## IMPORT DB2 INDEX Statement Syntax

The following diagram illustrates the IMPORT DB2 INDEX Statement Syntax.

**Figure 28: IMPORT DB2 INDEX Statement Syntax**



The following table describes the IMPORT DB2 INDEX Statement parameters and descriptions.

**Table 29: IMPORT DB2 INDEX Statement Parameters and Descriptions**

Parameter	Description
IMPORT DB2 INDEX	Keywords that identify the statement. All subsequent parameters describe the logical DB2 index to be imported and the processing options to be performed when storing that index in the eXadas Meta Data Catalog. Options allow renaming the index and correlation of index entries to renamed tables.
db2_owner	Optional parameter specifying the SQL authorization ID (CREATOR) of the DB2 index to be imported. If the db2_owner is not specified, the user ID used to execute the Meta Data Utility is assigned as the db2_owner. This parameter value cannot exceed 8 characters in length.
db2_index_name	Required parameter that identifies the DB2 index (NAME) to be imported into the eXadas Meta Data Catalog. This parameter value cannot exceed 18 characters in length. The combination of db2_owner and db2_index_name must identify an index that currently exists within the DB2 catalog.
RENAME AS	Optional keywords identifying the clause that allows renaming a index during the import operation. If this clause and its related parameters (cac_owner and cac_index_name) are omitted, the index will be stored in the eXadas Meta Data Catalog using db2_owner as the CREATOR and db2_index_name as the NAME. If this clause is specified, its related parameters describe the CREATOR and NAME that will be used when the index is stored in the eXadas Meta Data Catalog. In either case, the full name (CREATOR.NAME) must be unique within the eXadas Meta Data Catalog.
cac_owner	Optional parameter that is part of the optional 'RENAME AS' clause. It identifies the SQL authorization ID (CREATOR) of the index to be recorded in the eXadas Meta Data Catalog. If the cac_owner is not specified, the user ID used to execute the Meta Data Utility is assigned as the cac_owner. This parameter value cannot exceed 8 characters in length.
cac_index_name	Required parameter that is part of the optional 'RENAME AS' clause. It identifies the index (NAME) to be recorded in the eXadas Meta Data Catalog. This parameter value cannot exceed 18 characters in length. The combination of cac_owner and cac_index_name must be unique within the eXadas Meta Data Catalog.

**Table 29: IMPORT DB2 INDEX Statement Parameters and Descriptions**

DEFINED ON TABLE	Optional keywords identifying the clause that allows a DB2 index to be correlated with a previously renamed table during the import operation. If this clause and its related parameters ( <code>cac_owner</code> and <code>cac_table_name</code> ) are omitted, the index will be stored in the eXadas Meta Data Catalog using the TBCREATOR and TBNAME found in the DB2 index entry. If this clause is specified, its related parameters describe the TBCREATOR and TBNAME referenced in the index when the index is stored in the eXadas Meta Data Catalog. The full name (TBCREATOR.TBNAME) in the index entry must match an existing table within the eXadas Meta Data Catalog.
<code>cac_owner</code>	Optional parameter that is part of the optional 'DEFINED ON TABLE' clause. It identifies the SQL authorization ID (TBCREATOR) referenced in the index entry to be recorded in the eXadas Meta Data Catalog. If the <code>cac_owner</code> is not specified, the user ID used to execute the Meta Data Utility is assigned as the <code>cac_owner</code> . This parameter value cannot exceed 8 characters in length.
<code>cac_table_name</code>	Required parameter that is part of the optional 'DEFINED ON TABLE' clause. It identifies the TBNAME referenced in the index entry to be recorded in the eXadas Meta Data Catalog. This parameter value cannot exceed 18 characters in length. The combination of <code>cac_owner</code> and <code>cac_table_name</code> in the index entry must match an existing table within the eXadas Meta Data Catalog.

The following processing conditions and considerations should be taken into account when using the IMPORT DB2 INDEX Statement.

- The table associated with the DB2 index being imported is identified by the DEFINED ON TABLE clause, if present, or by information from the DB2 catalog. The table must currently exist in the eXadas Meta Data Catalog and that table must be a DB2 object or the import process will fail.
- If a liked-name index currently exists in the eXadas Meta Data Catalog, that index must be dropped (DROP INDEX) or a new name (RENAME AS) must be specified for the import to be successful.

## CICS VSAM Grammar

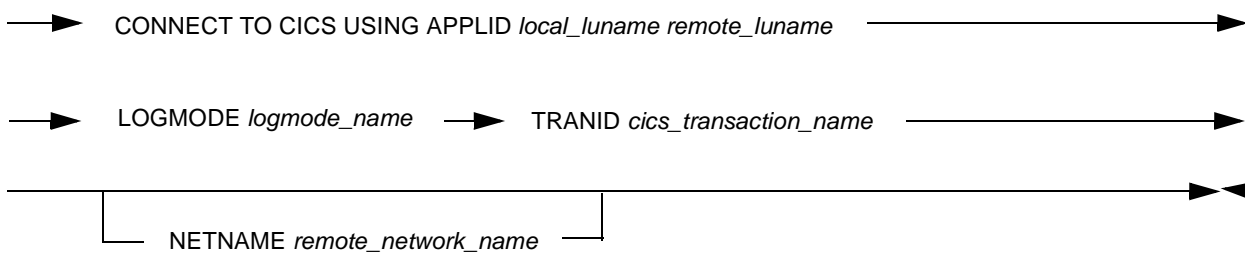
To validate VSAM mapping parameters at Meta Data Utility execution, the Meta Data Utility must connect to CICS and query information about the VSAM file(s) CICS is controlling. This is accomplished with the parameters in the CONNECT TO CICS statement. Communicating with CICS for this purpose is the same as communicating with a Data Savant except that the transaction to execute when

querying file information in CICS is different from the transaction used to access VSAM data.

The separate transaction for querying file information is necessary for security purposes. In general, CICS transactions are secured and the APPC conversation to CICS is authorized with the user ID and password of a user connected to the eXadas Server. The Meta Data Utility transaction, however, is intended to be unsecured as the user password is not available for the user of the Meta Data Utility and is not available for connecting to CICS. The transaction is restricted to retrieving non-sensitive data about such files as LRECL, key offset and lengths, and the like.

CrossAccess has provided the CONNECT TO CICS statement in the Meta Data Grammar to define the APPC connection parameters to allow the Meta Data Utility to validate CICS file information. The following syntax diagram describes this statement.

**Figure 29: CONNECT to CICS Statement Syntax Diagram**



The parameter descriptions are as follows:

**Table 30: CONNECT TO CICS Statement Parameters and Descriptions**

Parameter	Description
local_luname	A 1 to 8 character application ID (LU name) for the Meta Data Utility. This LU name should be different from the LU name used in table mapping to avoid conflict between an active eXadas Server and the Meta Data Utility.
remote_luname	A 1 to 8 character application ID (LU name) for the CICS system containing the VSAM file.
logmode_name	A 1 to 8 character LOGMODE name required for the APPC session.
cics_transaction_name	A 1-4 character transaction name defined in CICS for use by the eXadas Meta Data Utility. This transaction is created during the eXadas installation and verification process.
remote_network_name	An optional 1 to 8 character remote network name that can be used when two LU names exist on different networks.

The CONNECT statement defines the APPC information to be used by the Meta Data Utility for retrieving file information about CICS mapped tables. This file

information is then stored in the eXadas system catalogs and used to validate consistency between file and key information and mapped table and column information.

If you plan to run the Meta Data Utility and map file information while an eXadas server is running, the `local_luname` value in the generated `CONNECT` statement must be changed to ensure that a conflict does not occur between the Meta Data Utility and the Server attempting to concurrently access CICS with the same LU. To change the `local_luname` in the `CONNECT` statement, the generated grammar must be modified after it is created by the DataMapper.

For example, the `CONNECT` statement:

```
CONNECT TO CICS USING
  APPLID "CACAPCDS" "CICS1"
  LOGMODE "MTLU62"
  TRANID "EXV2"
```

could be changed to:

```
CONNECT TO CICS USING
  APPLID "CACAPPCMU" "CICS1"
  LOGMODE "MTLU62"
  TRANID "EXV2"
```

Although the DataMapper generates a separate `CONNECT` for each VSAM table mapped to CICS, a single `CONNECT` can be included in the grammar file and used for multiple VSAM table mappings. Once a `CONNECT` is processed by the Meta Data Utility, its values are retained for the duration of the Meta Data Utility run unless another `CONNECT TO CICS` statement is encountered. Therefore, when multiple `CONNECT`s within a grammar file require the same modification, the first `CONNECT` can be changed and all remaining `CONNECT`s can be removed from the grammar file.

## ADABAS USE Statement Generator

The ADABAS USE Statement Generator (USG) allows you to generate USE TABLE statements for the ADABAS Meta Data Utility based on information in the PREDICT system dictionary for the target ADABAS view. The USE statement generator program for ADABAS is CACADAUG.

The USG creates one or more USE TABLE statements that can be used as input to the Meta Data Utility. You can use output from USG as-is, or you can edit the statements before running the Meta Data Utility. Input for the USG consists of one OPTIONS statement and one or more EXTRACTTABLE statements.

Periodic group (PE) and multiple value (MU) fields are handled as Record Arrays. See the [“Record Arrays,” on page 176](#). The USG generates the statements for the



Record Array(s), but leaves them commented out. Only one Record Array is allowed per table.

By creating additional logical tables using the record arrays (the columns that are commented out) you can create third normal form of the PEs and MUs. For example, by creating another USE TABLE definition with the DESCRIPTIVE column (index) along with a single record array, you can have third normal form representation.

The OPTIONS statement specifies the file number of the system dictionary and the EXTRACT TABLE statements identify the target ADABAS views.

The following table displays the OPTIONS statements syntax with the defaults underlined.

```

OPTIONS  SYSDIC file number DATE 'MM/DD/YY'  TIME 'HH:MI:SS'
                                'date format'    'time format'

GROUP NO                SUPERDESC ALL          VARCHAR LENGTH nn
      ALL                CHAR          NO
      CHAR                NO

LONG VARCHAR LENGTH nn    MAXOCCURS nn          SYNONYM NO
                                                YES
  
```

The parameter descriptions are as follows:

**Table 31: OPTIONS Statements and Descriptions**

OPTIONS Statement	Description
OPTIONS	Specifies the keyword that identify the statement. Only one OPTIONS card can be specified.
SYSDIC <i>file-number</i>	Identifies the system dictionary file number as defined to ADABAS.
DATE <i>date-format</i>	Specifies the default date format to be used when generating Meta Data Grammar for ADABAS date fields. Any combination of MM (months), DD (days), YY or YYYY (year) can be used along with /, -, spaces and other characters. MMM will return the name of the month.
TIME <i>time-format</i>	Specifies the default time format to be used when generating Meta Data Grammar for ADABAS time fields. Any combination of MM (months), DD (days), YY or YYYY (years), HH (hours), MI (minutes), and SS (seconds) can be used along with /, -, spaces and other characters. MMM will return the name of the month.

**Table 31: OPTIONS Statements and Descriptions**

<b>OPTIONS Statement</b>	<b>Description</b>
GROUP	<p>Specifies whether Meta Data Grammar will be generated for group fields.</p> <ul style="list-style-type: none"> <li>• NO specifies do not generate Meta Data Grammar for any group fields.</li> <li>• CHAR specifies to only generate Meta Data Grammar if all the elements of the group are character or zoned decimal.</li> <li>• ALL specifies to generate Meta Data Grammar for all group fields.</li> </ul>
SUPERDESC	<p>Specifies whether Meta Data Grammar will be generated for superdescriptors.</p> <ul style="list-style-type: none"> <li>• CHAR specifies to only generate Meta Data Grammar for superdescriptors where all the elements of the superdescriptor are character or zoned decimal.</li> <li>• NO specifies to never generate Meta Data Grammar for superdescriptor fields.</li> <li>• ALL specifies to always generate Meta Data Grammar for all superdescriptor fields.</li> </ul>
VARCHAR LENGTH	<p>Specifies the default maximum length to use for VARCHAR fields. The default is 20.</p>
LONG VARCHAR LENGTH	<p>Specifies the default maximum length for LONG VARCHAR fields. The default is 20.</p>
MAXOCCURS	<p>Specifies the default maximum number of occurrences to use for Record Arrays. Record Arrays are used to define periodic group and multiple value fields. The default is 5.</p>
SYNONYM	<p>Specifies whether the Predict User Synonym field will be used for the column name.</p> <ul style="list-style-type: none"> <li>• NO specifies to use the Predict Dictionary field name.</li> <li>• YES specifies to use the User Synonym field from the Predict Dictionary when it has been defined.</li> </ul>

**NOTE:** Options can be omitted, but the options entered must be in the order specified previously.

The following table displays the EXTRACT TABLE statements syntax.

```
EXTRACT TABLE table-name FROM VIEW view-name FILE nn DBID nn
```

The parameter descriptions are as follows:

**Table 32: EXTRACT Table Statements Syntax**

<b>EXTRACT Statement</b>	<b>Description</b>
EXTRACT TABLE	specifies the keywords that identify the statement.
<i>table-name</i>	specifies the name of the table for which USE TABLE statements are generated. If you do not specify a table name, the first 18 characters of the view name are used with all dashes ( - - - ) replaced with underscores ( _ _ _ ).
FROM VIEW <i>view-name</i>	identifies the ADABAS view from which USE TABLE statements are generated.
FILE nn	Logical file number where view exists.
DBID nn	Database ID where view exists. It overrides what is in the predict database.



## Open Catalog

### Introduction to Open Catalog

This chapter covers open catalog support, which allows you to obtain access to the internal information stored in the eXadas System Catalog. The topics covered include:

- [“Objects Used to Define and Access a Table,”](#) on page 202,
- [“What are Fragments?,”](#) on page 204,
- [“Record Arrays,”](#) on page 208,
- [“Differences With Meta Data Tables,”](#) on page 210, and
- [“Installing Meta Table in the eXadas System Catalog,”](#) on page 214.

# Open Catalog Overview

Open catalogs provide access to the DB2, eXadas, and DBMS-specific information stored in the eXadas System Catalog. This is done by projecting DBMS-specific meta data tables over the standard DB2 SYSIBM.SYSTABLES, SYSIBM.SYSCOLUMNS, SYSIBM.SYSINDEXES and SYSIBM.SYSKEYS tables.

The SYSIBM.SYSTABLES and SYSIBM.SYSCOLUMNS tables, and optionally the SYSIBM.SYSINDEXES and SYSIBM.SYSKEYS tables, are used to define non-relational databases and files within the eXadas System Catalog. These system tables are also populated when DB2 tables are imported into the eXadas System Catalog.

The DBMS-specific table projections use the creator (owner) name SYSCAC. The meta data table names identify the data source they are mapped over. They have names like SYSIMSTABLES, SYSIDMSTABLES, and so on. For all data sources except sequential files, there is one meta data table defined for each IBM system table. Sequential does not support projecting meta data tables over the SYSIBM.SYSINDEXES and SYSIBM.SYSKEYS tables because these tables are never populated for sequential data sources. [Table 33, “Meta Data Tables,” on page 199](#), identifies all of the meta data tables that are supported.

When you access a meta data table, the Catalog Savant automatically performs filtering based on the DBMS name of the table being accessed. For example, when you issue a query against the SYSCAC.SYSIDMSTABLES table, only information about tables that access IDMS data are returned.

In addition to the “standard” meta data tables, there are three “special” meta data tables that are not DBMS-specific. These tables use the name META as the intermediary qualifier (for example, SYSMETACOLUMNS). The META tables allow common information to be retrieved about any table or column defined in the eXadas System Catalog. You can think of the META tables as providing a “template” that provides basic information about any table defined in the System Catalog.

Two of the META tables are projected over the standard IBM SYSTABLES and SYSCOLUMNS tables. These versions contain both the common DB2 column definitions as well as eXadas-specific columns. The third META data table, SYSCAC.SYSMETAFRAGMENTS, is eXadas-specific, and is described in [“What are Fragments?,” on page 204](#).

The following table identifies the meta data tables that have been created. The tables are listed in alphabetical order. For each table a brief description of the tables contents is provided.

**Table 33: Meta Data Tables**

Table Name	Description
SYSCAC. SYSADABASCOLUMNS (see the table on <a href="#">page 440</a> )	Contains the standard SYSIBM.SYSCOLUMNS columns and Adabas-specific column definitions. There is one row for each column for every Adabas table defined in the eXadas System Catalog.
SYSCAC.SYSADABASINDEXES (see the table on <a href="#">page 445</a> )	Contains the standard SYSIBM.SYSINDEXES columns and Adabas-specific column definitions. There is one row for each Adabas index defined in the eXadas System Catalog.
SYSCAC.SYSADABASKEYS (see the table on <a href="#">page 446</a> )	Contains the standard SYSIBM.SYSKEYS columns and Adabas-specific column definitions. There is one row for each ADABAS index key defined in the eXadas System Catalog..
SYSCAC.SYSADABASTABLES (see the table on <a href="#">page 447</a> )	Contains the standard SYSIBM.SYSTABLES columns and Adabas-specific column definitions. There is one row for each Adabas table defined in the eXadas System Catalog.
SYSCAC. SYSDATACOMCOLUMNS (see the table on <a href="#">page 450</a> )	Contains the standard SYSIBM.SYSCOLUMNS columns and CA-DATACOM/DB-specific column definitions. There is one row for each column for every DATACOM table defined in the eXadas System Catalog.
SYSCAC.SYSDATACOMINDEXES (see the table on <a href="#">page 454</a> )	Contains the standard SYSIBM.SYSINDEXES column definitions. There are no CA-DATACOM/DB-specific column definitions. There is one row for each CA-DATACOM/DB index defined in the eXadas System Catalog.
SYSCAC.SYSDATACOMKEYS (see the table on <a href="#">page 456</a> )	Contains the standard SYSIBM.SYSKEYS column definitions. There are no CA-DATACOM/DB-specific column definitions. There is one row for each column of a CA-DATACOM/DB key.
SYSCAC.SYSDATACOMTABLES (see the table on <a href="#">page 456</a> )	Contains the standard SYSIBM.SYSTABLES columns and CA-DATACOM/DB-specific column definitions. There is one row for each CA-DATACOM/DB table defined in the eXadas System Catalog.
SYSCAC.SYSDB2COLUMNS (see the table on <a href="#">page 460</a> )	Contains the standard SYSIBM.SYSCOLUMNS columns and eXadas-specific column definitions. There is one row for each column for every DB2 table defined in the eXadas System Catalog.

**Table 33: Meta Data Tables**

Table Name	Description
SYSCAC.SYSDB2INDEXES (see the table on <a href="#">page 461</a> )	Contains the standard SYSIBM.SYSINDEXES column definitions. There are no eXadas-specific column definitions. There is one row for each DB2 index defined in the eXadas System Catalog.
SYSCAC.SYSDB2KEYS (see the table on <a href="#">page 463</a> )	Contains the standard SYSIBM.SYSKEYS column definitions. There are no eXadas-specific column definitions. There is one row for each DB2 key defined in the eXadas System Catalog.
SYSCAC.SYSDB2TABLES (see the table on <a href="#">page 463</a> )	Contains the standard SYSIBM.SYSTABLES columns and eXadas-specific column definitions. There is one row for each DB2 table defined in the eXadas System Catalog.
SYSCAC.SYSIDMS_INDEXES (see the table on <a href="#">page 466</a> )	Contains the standard SYSIBM.SYSTABLES columns and IDMS-specific column definitions. There is one row for each IDMS index associated with an IDMS table defined in the eXadas System Catalog.
SYSCAC.SYSIDMSAREAS (see the table on <a href="#">page 470</a> )	Contains the standard SYSIBM.SYSTABLES columns and IDMS-specific column definitions. There is one row for each IDMS area associated with an IDMS table defined in the eXadas System Catalog.
SYSCAC.SYSIDMSCOLINDEXES (see the table on <a href="#">page 474</a> )	Contains the standard SYSIBM.SYSCOLUMNS columns and IDMS-specific column definitions. There is one row for each column that references an IDMS index for every IDMS table defined in the eXadas System Catalog.
SYSCAC.SYSIDMSCOLUMNS (see the table on <a href="#">page 478</a> )	Contains the standard SYSIBM.SYSCOLUMNS columns and IDMS-specific column definitions. There is one row for each column for every IDMS table defined in the eXadas System Catalog.
SYSCAC.SYSIDMSINDEXES (see the table on <a href="#">page 485</a> )	Contains the standard SYSIBM.SYSINDEXES column definitions. There are no IDMS-specific column definitions for this table. There is one row for each IDMS index defined in the eXadas System Catalog.
SYSCAC.SYSIDMSKEYS (see the table on <a href="#">page 487</a> )	Contains the standard SYSIBM.SYSKEYS column definitions. There are no IDMS-specific column definitions. There is one row for each IDMS key defined in the eXadas System Catalog.
SYSCAC.SYSIDMSRECORDS (see the table on <a href="#">page 488</a> )	Contains the standard SYSIBM.SYSTABLES columns and IDMS-specific column definitions. There is one row for each IDMS record associated with an IDMS table defined in the eXadas System Catalog.



**Table 33: Meta Data Tables**

Table Name	Description
SYSCAC.SYSIDMSSETS (see the table on <a href="#">page 492</a> )	Contains the standard SYSIBM.SYSTABLES columns and IDMS-specific column definitions. There is one row for each IDMS set associated with an IDMS table defined in the eXadas System Catalog.
SYSCAC.SYSIDMSTABLES (see the table on <a href="#">page 495</a> )	Contains the standard SYSIBM.SYSTABLES columns and IDMS-specific column definitions. There is one row for each IDMS table defined in the eXadas System Catalog.
SYSCAC.SYSIMSCOLUMNS (see the table on <a href="#">page 512</a> )	Contains the standard SYSIBM.SYSCOLUMNS columns and IMS-specific column definitions. There is one row for each column for every IMS table defined in the eXadas System Catalog.
SYSCAC.SYSIMSINDEXES (see the table on <a href="#">page 516</a> )	Contains the standard SYSIBM.SYSINDEXES columns and IMS-specific column definitions. There is one row for each IMS index defined in the eXadas System Catalog.
SYSCAC.SYSIMSKEYS (see the table on <a href="#">page 518</a> )	Contains the standard SYSIBM.SYSKEYS column definitions. There are no IMS-specific column definitions. There is one row for each IMS primary or secondary index key defined in the eXadas System Catalog.
SYSCAC.SYSIMSSEGMENTS (see the table on <a href="#">page 519</a> )	Contains the standard SYSIBM.SYSTABLES columns and IMS-specific column definitions. There is one row for each IMS segment referenced by an IMS table defined in the eXadas System Catalog.
SYSCAC.SYSIMSTABLES (see the table on <a href="#">page 522</a> )	Contains the standard SYSIBM.SYSTABLES columns and IMS-specific column definitions. There is one row for each IMS table defined in the eXadas System Catalog.
SYSCAC.SYSMETACOLUMNS (see the table on <a href="#">page 534</a> )	Contains the standard SYSIBM.SYSCOLUMNS columns, eXadas-specific columns and descriptive columns about the fragment associated with the column. There is one row for each column defined in the eXadas System Catalog.
SYSCAC.SYSMETAFRAGMENTS (see the table on <a href="#">page 538</a> )	The SYSCAC.SYSMETAFRAGMENTS table is eXadas-specific. There is one row for every fragment definition in the eXadas System Catalog.
SYSCAC.SYSMETATABLES (see the table on <a href="#">page 539</a> )	Contains the standard SYSIBM.SYSTABLES columns and eXadas-specific column definitions that are common to all table definitions. There is one row for each table or view defined in the eXadas System Catalog.

**Table 33: Meta Data Tables**

Table Name	Description
SYSCAC.SYSSEQCOLUMNS (see the table on <a href="#">page 542</a> )	Contains the standard SYSIBM.SYSCOLUMNS columns and sequential file-specific column definitions. There is one row for each column for every sequential file table defined in the eXadas System Catalog.
SYSCAC.SYSSEQTABLES (see the table on <a href="#">page 545</a> )	Contains the standard SYSIBM.SYSTABLES columns and sequential file-specific column definitions. There is one row for each sequential file table defined in the eXadas System Catalog.
SYSCAC.SYSVSAMCOLUMNS (see the table on <a href="#">page 548</a> )	Contains the standard SYSIBM.SYSCOLUMNS columns and VSAM-specific column definitions. There is one row for each column for every VSAM table defined in the eXadas System Catalog.
SYSCAC.SYSVSAMINDEXES (see the table on <a href="#">page 552</a> )	Contains the standard SYSIBM.SYSINDEXES columns and VSAM-specific column definitions. There is one row for each VSAM index defined in the eXadas System Catalog.
SYSCAC.SYSVSAMKEYS (see the table on <a href="#">page 554</a> )	Contains the standard SYSIBM.SYSKEYS column definitions. There are no VSAM-specific column definitions. There is one row for each column of a VSAM key.
SYSCAC.SYSVSAMTABLES (see the table on <a href="#">page 555</a> )	Contains the standard SYSIBM.SYSTABLES columns and VSAM-specific column definitions. There is one row for each VSAM table defined in the eXadas System Catalog.

## Objects Used to Define and Access a Table

Not counting security-related tables, normally four IBM system tables are used to define a user table. From an eXadas standpoint, these four tables are used to describe the contents of, and how to access and update an instance of, an object in a non-relational or DB2 data source. For example, a table defines a VSAM file to be accessed/updated, the path of IMS segments in an IMS DBD to be accessed/updated, the IDMS records in an IDMS database to be accessed/updated, and so on. DB2 table definitions can also be imported into the eXadas System Catalog for referential integrity purposes.

In the following sections, an object-oriented metaphor will be used to describe the contents of the system tables and their related DBMS-specific information. eXadas has a DB2 history and uses IBM DB2 Version 2.2 (with extensions) as its

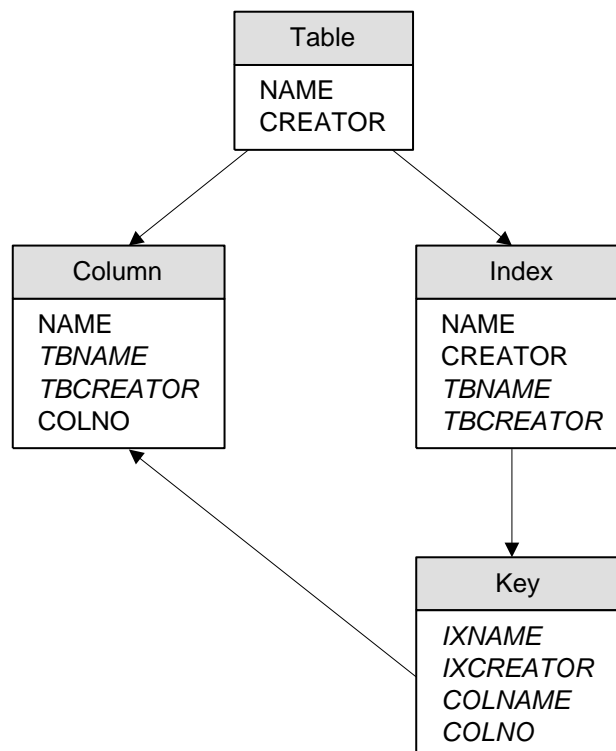
reference model to define how to access and update the target data source. [Figure 30: “DB2 Objects Used to Define a Table”](#) shows the four DB2 objects that are used to define a table.

The Table object identifies a table instance and contains basic information about the table. Associated with the table are one or more Columns that describe data elements that make up the table. A table can have zero, one, or more than one indexes that describe Keys that can be used to directly access the contents of the table. Each Key must reference a Column within the Table.

Each table is uniquely identified by a combination of the NAME and CREATOR columns. The NAME, TBCREATOR, and TBNAME columns uniquely identify each Column within a Table. A Column can also be uniquely identified using the TBCREATOR, TBNAME, and COLNO columns. For meta data tables, a Column can also be uniquely identified using the TBCREATOR, TBNAME, and COLUMN\_SEQUENCE\_NUMBER columns.

The NAME and CREATOR columns uniquely identify an Index. The TBNAME and TBCREATOR columns identify the Table for which the Index is defined. The IXNAME, IXCREATOR, and COLNAME columns uniquely identify keys. The IXNAME and IXCREATOR columns identify the Index that the key(s) are associated with. The COLNAME column identifies the NAME of the Column that the Key is referencing. The COLNO column also identifies the associated Column that the Key is referencing.

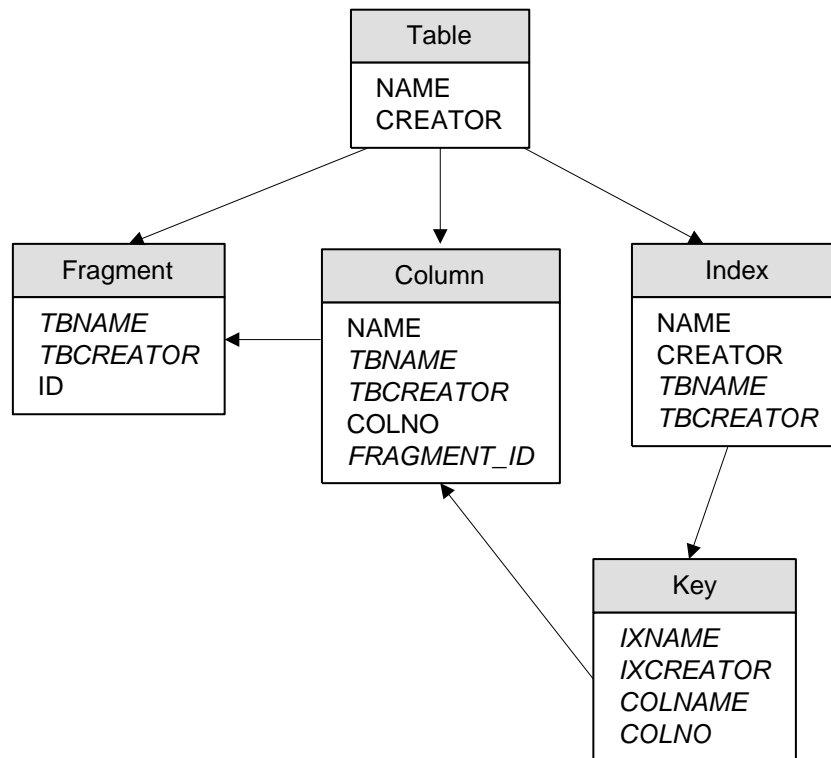
**Figure 30: DB2 Objects Used to Define a Table**



# What are Fragments?

In eXadas, in addition to the standard DB2 objects, an additional Fragment object is used to define a table. The following figure shows the additional Fragment object and its relationship to the other objects.

**Figure 31: Objects Used to Define an eXadas Table**



Each table has at least one fragment associated with it. The only exceptions are the system tables (SYSIBM.SYSTABLES, SYSIBM.SYSCOLUMNS, and so on), which have an implied fragment definition associated with them. This implied fragment is materialized when one of these tables is referenced in a query. DB2 tables do not have fragments associated with them because they use a different technique to transfer data between the DB2 Savant and the Query Processor than the other non-relational Savants use. Non-relational data sources can have more than one fragment definition associated with a table definition.

An IMS table has a fragment definition for each segment referenced by the table. Likewise, an IDMS table has a fragment for each record referenced by the table. Tables that contain record arrays also have a fragment for each record array

---

defined. There are also two special “system” fragments that may be associated with an IMS or IDMS table:

- For IMS, a system fragment can be used to identify an XDFLD value that is extracted from the key feedback area.
- For IDMS, a system fragment is used to refer to a VSAM relative record number.

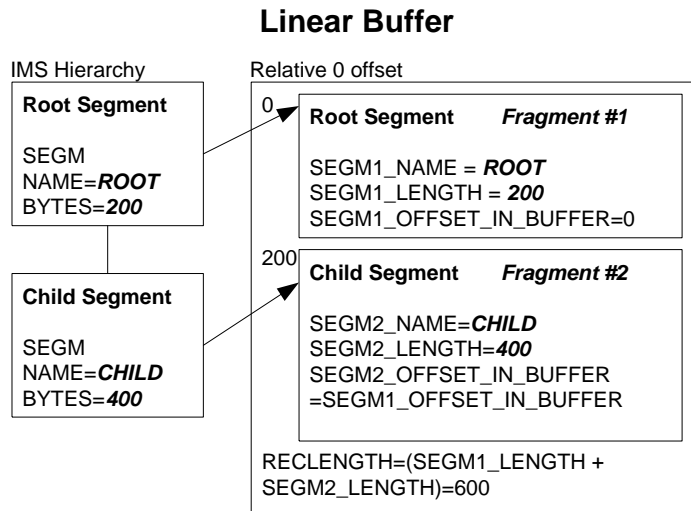
The TBCREATOR, TBNAME, and ID columns uniquely identify a fragment. In addition to the standard DB2 columns, the eXadas version of the Column object contains a FRAGMENT\_ID column that identifies the fragment that is associated with the column. Likewise, the eXadas version of the Table contains one or more columns that identify the fragment that is associated with the table or one of its components. The number and names of these columns are DBMS-specific and can be found in the DBMS-specific table definitions, which start on [page 440](#).

For non-DB2 table definitions, data is transferred between the Savant and the Query Processor using a linear buffer. The fragments are used to identify the starting position and length of an IMS segment, IDMS record, and/or record array within the linear buffer.

## IMS Example

The following figure shows how an IMS database is mapped as fragments and projected into the linear buffer. In this example, a simple two-level fixed length segment database has been defined. The linear buffer that is allocated is based on the RECLENGTH column in the underlying SYSIBM.SYSTABLES definition. In this example, the linear buffer length has been computed based on the maximum sizes of the two segments that are being accessed. In the SYSCAC.SYSIMSTABLES definition for this table, the segments composing the table are named in hierarchic order. The convention is SEG*M**n* where *n* is between 1 and 15.

Figure 32: IMS Linear Buffer Example



Although you can use the `SYSCAC.SYSIMSTABLES` meta data table definition to retrieve information about all of the segments referenced by an IMS table definition, you either have to:

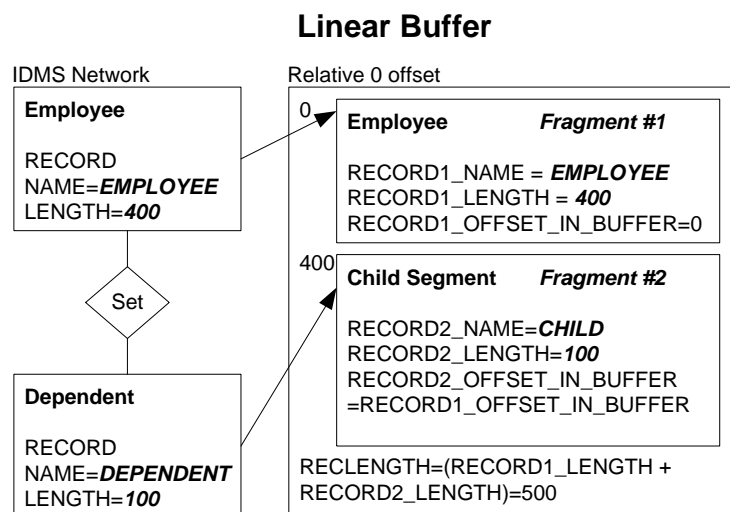
- know how many segments a particular IMS table references, which requires that each query be customized to reference the correct number of segments, or
- use a generic query that retrieves information about all possible segments (maximum number of 15), which means most of the columns retrieved will contain null values.

To make retrieving information about the segments that are referenced by IMS table definitions easier, another `SYSCAC.SYSIMSSEGMENTS` meta data table is available for your use. This meta data table uses a record array to denormalize the segment information and return one row for each segment referenced by an IMS table definition.

## IDMS Example

The following figure shows how an IDMS database is mapped as fragments and projected into the linear buffer. In this example, two records have been mapped as a table. The linear buffer that is allocated is based on the `RECLENGTH` column in the underlying `SYSIBM.SYSTABLES` definition. In this example, the linear buffer length has been computed based on the maximum sizes of the two records that are being accessed. In the `SYSCAC.SYSIDMSTABLES` definition for this table, the records composing the table are named in their physical access order. The convention is `RECORD $n$`  where  $n$  is between 1 and 10.

Figure 33: IDMS Linear Buffer Example



As with IMS, alternate record array IDMS-specific meta data tables are available:

- SYSCAC.SYSIDMSRECORDS (see [page 488](#))
- SYSCAC.SYSIDMSSETS (see [page 492](#))
- SYSCAC.SYSIDMSAREAS (see [page 470](#))
- SYSCAC.SYSIDMS\_INDEXES (see [page 466](#))

IDMS columns also have information about the indexes that are used to access the column. So, in addition to a “standard” SYSCAC.SYSIDMSCOLUMNS meta data table, there is also a SYSCAC.SYSIDMSCOLINDEX meta data table that provides record array access to the index information associated with an IDMS column.

In the case of IMS and IDMS, meta data tables take advantage of mapping non-normalized data into a relational object. From an eXadas perspective, non-normalized data comes in two forms:

- Redefinitions—these are handled by the use of the definition of views, or by using WHERE clause qualification, and
- Arrays—these are handled by the definition of record arrays.

Meta data tables do not contain redefinitions, but there are record arrays in the eXadas meta data table definitions. How record arrays are projected and processed in the linear buffer is discussed next.

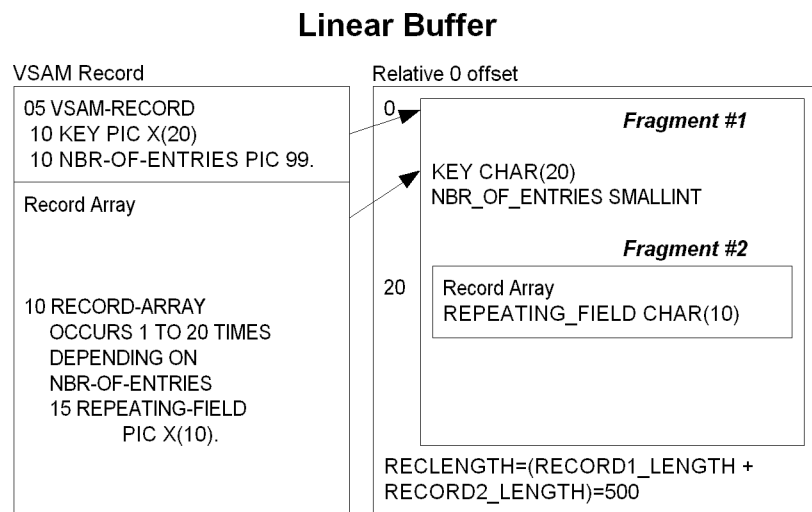
# Record Arrays

A record array defines a column or set of columns that may occur one or more times within a database object. A database object is an IMS segment, an IDMS record, a CA-DATACOM/DB table, and so on. In Meta Data Grammar, you identify the start and ending range of the columns in a record array using the `BEGINLEVEL n` and `ENDLEVEL` statements. eXadas allows you to map multiple fixed number of repetition record arrays in a single database object. The use of this feature is not recommended, and is not used in the meta table definitions, because the result of retrieving database objects with multiple record arrays is the Cartesian product of each instance of a record array occurrence.

However, since it is common for legacy systems to be defined as occurring a variable number of times using a COBOL `OCCURS DEPENDING ON` clause, XDi supports the definition of one variably-occurring record array in a database object. The column(s) that make up this record array must be defined as the last set of column(s) in the table definition.

The following figure shows how a VSAM table containing a record array is projected into the linear buffer, and the fragments that are used to manage access to the table's contents. This projection works the same way for all non-relational database objects.

**Figure 34: Record Array Example**



A record array definition references the starting position of the set of columns that make up the record array within the database object, as well as the length of a record array occurrence. All columns within the record array use a relative zero offset to the start of the record array. In fact, all columns use relative zero offsets to the location of the physical data from the fragment they are associated with. In [Figure 34: “Record Array Example,”](#) the column `REPEATING_FIELD` has an offset of zero, and a maximum of 20 occurrences can be materialized out of a single instance of *Fragment 2*.



When the column `REPEATING_FIELD` is referenced in a query, record array processing is activated. A row is materialized, or possibly materialized, for each occurrence of the record array “object”—the set of column(s) that make up the record array. You can think of the record array as a window that moves down in the database object each time a “record” is retrieved from the database object by the Query Processor.

In the case of an `OCCURS` depending on record array definition, the maximum number of instances that can be materialized is identified by the `MAXIMUM_OCCURENCES` column. The actual number of occurrences that are materialized is determined by the value in the column identified by the name of the column identified by the `CONTROL_COLUMN` and `CONTROL_COLUMN_SEQUENCE_NUMBER` columns. The column identified is referred to as the controlling column, and must be defined before any of the column definitions within the record array.

A record array can physically occur a fixed number of times, but some of the occurrences are considered not to exist based on values of some or all of the of the data in a record array occurrence. The `NULL_LENGTH` and `NULL_VALUE` columns identify the value that constitutes a null value. The `NULLIS_RULE` column identifies how a non-existent record array occurrence is determined.

The following table identifies the `NULLIS_RULE` values that are supported. The table also identifies the text that is contained in the `NULL_FRAGMENT_RULE` column that is contained in each of the meta data tables that are mapped against the `SYSIBM.SYSCOLUMNS` table (such as `SYSIDMSCOLUMNS`). Also included is a description of how a null instance is determined.

**Table 34: NULLIS\_RULE Values**

<b>NULLIS_RULE Value</b>	<b>NULL_FRAGMENT_RULE Value</b>	<b>Description</b>
0	NULL	Either the fragment is not associated with a record array, or the record array occurs the number of times identified by the <code>MAXIMUM_OCCURENCES</code> column.
1	<code>DEPENDING_ON_INDEX_COL</code>	The record array is an <code>OCCURS</code> depending on record array. The number of occurrences depends on the contents of the control column.
2	<code>NULLIS_COLUMN_MATCH</code>	An occurrence is null if the column identified in <code>CONTROL_COLUMN</code> contains the value specified in the <code>NULL_VALUE</code> column.
3	<code>NULLIS_ALL_COLUMN_MATCH</code>	An occurrence is null if the start of the column identified in <code>CONTROL_COLUMN</code> contains the value specified in the <code>NULL_VALUE</code> column.

**Table 34: NULLIS\_RULE Values**

NULLIS_RULE Value	NULL_FRAGMENT_RULE Value	Description
4	NULLIS_MATCH	An occurrence is null if the data contains the value specified in the NULL_VALUE column.
5	NULLIS_MATCH_ALL	An occurrence is null if the beginning of the data contains the value specified in the NULL_VALUE column.

For non-relational databases, fragments are the key to defining the data to be accessed and updated. These fragments are created automatically based on the database to be accessed and the columns defined within a table.

The eXadas IBM definitions of the base system tables themselves (SYSIBM.SYSTABLES, SYSIBM.SYSCOLUMNS, and so on) do not have fragment definitions, though one is synthesized when the table is accessed. Imported DB2 tables also do not have fragments associated with them and do not use linear buffers to transfer data to the query processor.

## Differences With Meta Data Tables

Each meta data table contains column definitions for the columns that are defined in the “base” system table definition. For example, the SYSCAC.SYSIMSTABLES table contains all of the columns defined in the SYSIBM.SYSTABLES table. The SYSCAC.SYSIMSTABLES table has eXadas-specific column definitions that are IMS-specific, as well as some “standard” column definitions that are common to all eXadas tables. The same is true for the other “base” tables—SYSCOLUMNS, SYSINDEXES, and SYSKEYS.

**NOTE:** Although there are DBMS-specific tables projected over all four base tables, for all data sources (except sequential files), many of the SYSINDEXES and SYSKEYS tables only contain DB2 column definitions because there are no eXadas-specific columns required to perform keyed access the table via an index.

Although the meta data tables contain definitions for all of the standard DB2 columns, their meta data definitions are often subtly different than the definitions found in DB2. The meta data tables, themselves, are also subtly different than either a system table or a user table—they have some characteristics of both. These differences are discussed in the following topics:

- [“VARCHAR Columns,”](#)
- [“Use of NULL IS Definitions,” on page 211,](#)
- [“Use of the REMARKS Column,” on page 212,](#)

- [“Predefined Table Names,” on page 213,](#)
- [“Use of Indexes,” on page 213,](#) and
- [“Ability to Delete Meta Data Tables,” on page 214.](#)

## VARCHAR Columns

There are no VARCHAR column definitions in the meta data tables. For example, the NAME column in the SYSIBM.SYSTABLES table is defined as VARCHAR(18), but in the meta data tables the NAME column is defined as CHAR(18). Internally in the eXadas System Catalog, VARCHAR columns are actually stored as CHAR values with an extra two bytes reserved at the end of the column for the column's length. When a query is issued against a SYSIBM system table, the catalog savant converts any VARCHAR column from its CHAR format into a true VARCHAR column in memory.

Although the meta data definition of a DB2 VARCHAR column is defined as a CHAR column, you can still issue a query that perform joins between a DB2 table and their meta data equivalents that references both the DB2 VARCHAR and meta data CHAR version of the column in the WHERE or ON clause. These types of queries return the correct result sets because of the type compatibility between CHAR and VARCHAR columns. However, if you issue a query that performs a LENGTH function against a VARCHAR DB2 column and its meta data equivalent, the results will be different because the trailing spaces for the VARCHAR version of the column have been removed by the catalog savant while in the meta data version of the column the trailing spaces are treated as data.

## Use of NULL IS Definitions

For queries, eXadas uses IBM DB2 Version 4 as its reference model. This means that you can issue queries that use the SQL syntax described in the *IBM DB2 Version 4 SQL Reference Manual* and the result set output is returned as described in the IBM manual. The only exceptions have to do with queries that attempt to perform date, time, or time stamp functions, since these functions are not supported by eXadas. The SYSIBM system table definitions, with the exception of stored procedure tables, use IBM DB2 version 2.2 as their reference model.

Many of the columns defined in the SYSIBM system tables are not used by eXadas, but are nonetheless defined for compatibility purposes. Likewise, these columns are also defined in the meta data tables. However, when the meta data versions of these columns are retrieved, oftentimes the column will be returned as a null value. This is because in the meta data versions these columns have been defined using the equivalent of the NULL IS clause that can be specified for

columns in a user-defined table. Whether a meta data column can be null and what constitutes a null value is documented for each column.

**WARNING:** As discussed in the previous topic, you can issue queries that perform joins against a DB2 table and its meta data equivalent against VARCHAR and CHAR column definitions and these queries work as you would expect. This is not true for meta data columns that support NULL values. If you issue a query that contains a join predicate between a DB2 column and its meta data equivalent that supports NULL values, then generally an empty or “truncated” result set will be returned.

Because these columns are not of interest to eXadas and are populated with default values, it is highly unlikely that you would issue a query that attempted to join based on these columns.

For eXadas-specific columns, the use of NULL IS specifications are used extensively. Many of the eXadas-specific columns are optional. Those that are not can be identified because there is a NOT NULL clause in the Data Type column in the table descriptions.

## Use of the REMARKS Column

The SYSIBM.SYSTABLES and SYSIBM.SYSCOLUMNS tables both contain a REMARKS column. The DB2 definition of these columns is as a VARCHAR(254) column that is populated using the COMMENT ON statement. As covered in [“VARCHAR Columns,” on page 211](#), their meta data equivalents are defined as CHAR(254), and eXadas does not support the COMMENT ON statement.

For the SYSIBM system tables and user-defined tables, the REMARKS column is not populated with any value. This is not true for meta table definitions. For each meta table and each meta data column, the REMARKS column contains a short description of the table or column.

For the REMARKS column in the table definition, the description uses the same style found in the DB2 definitions of the system tables. These descriptions identify how many instances of the table exist within the eXadas System Catalog. The eXadas and DB2 descriptions of a table assume that the reader knows what the purpose of the table is. Usually, these descriptions are not very useful.

The REMARKS columns associated with meta data column definitions, however, do contain meaningful descriptions. For each column, the REMARKS column identifies whether the column is a DB2 column or an eXadas-specific column. Generally, for a DB2 column, the remainder of the description will either contain the DB2 definition of the use of the column (or a terser version since the REMARKS column can only contain 254-characters) or the phrase “Not used by eXadas” which means that the column exists for compatibility purposes with the DB2 Version 2.2 reference model.

A column that is identified as an eXadas-specific column includes a short description of the purpose of the column and whether or not it is a synthesized column. To make accessing the information stored in the eXadas System Catalog easier, the meta data table definitions for non-relational databases contain several synthesized columns that identify information that has been extracted from the fragment definition that is associated with the table or a column. These synthesized columns can be identified by the REMARKS column beginning with the sentence “eXadas column—obtained from fragment definition.”

For IDMS meta data columns, several columns are also synthesized from the IDMS record and index columns associated with the IDMS table definition. These synthesized columns can be identified by the REMARKS column beginning with the sentence “eXadas column—obtained from table definition.”

## Predefined Table Names

Like the SYSIBM system table names, the meta data table names are predefined. The SYSCAC creator name was chosen because it is unlikely that you will be defining user tables that would use SYSCAC as an owner name. The actual table names (such as SYSIDMSTABLES) were selected to identify these as system tables, and to identify the DBMS they are mapped over and the base IBM table that they are referencing. The meta data tables that contain record arrays follow along the same lines, and identify the DBMS-specific object that the record array is mapped against.

Internally, the system tables and the META tables are defined as static structures in the Meta Data Utility. Using these static structures, the Meta Data Utility automatically builds the system table definitions when it detects that the catalogs are empty. As part of open catalog support, the Meta Data Utility has been modified to automatically create the META table definitions when the system tables are created, and to check for the existence of the SYSCAC.SYSMETATABLES table each time it is run. If this table is not found, the META tables are automatically created in the System Catalog.

## Use of Indexes

You cannot define an index for a meta data table. Attempting to will generate a -104 SQL code from the Meta Data Utility, because meta data tables are mapped over the system tables, and you cannot define an index on a system table.

That does not mean that if you issue a query against a meta data table that an index is not used to access the meta data table’s content. When the system tables are initially loaded into the System Catalog, indexes are automatically created for the SYSIBM.SYSTABLES, SYSIBM.SYSCOLUMNS and SYSIBM.SYSINDEXES tables. Other indexes are also automatically created for the security related tables.

For the `SYSIBM.SYSTABLES` table, the index is defined on the `CREATOR` and `NAME` columns. For the `SYSIBM.SYSCOLUMNS` table, the index is defined on the `TBCREATOR` and `TBNAME` columns. Likewise, the `SYSIBM.SYSINDEXES` table has an index defined on the `CREATOR` and `NAME` columns.

When you issue a query against a meta data table, the Catalog Savant automatically picks up the associated `SYSIBM` index information associated with the underlying system table. Therefore, queries against meta data tables that contain `WHERE` clause qualification on the creator and table name will run faster because they are using the underlying indexes to access the requested tables information.

## Ability to Delete Meta Data Tables

You can use the standard `GRANT` and `REVOKE` statements to secure access to the meta data tables. However, unlike the system tables, you can delete meta data table definitions from the eXadas System Catalog using the `DROPTABLE` statement.

# Installing Meta Table in the eXadas System Catalog

Loading the meta data table definitions into a new or existing System Catalog is a multi-step process. Initially, the `SYSCAC.SYSMETATABLES`, `SYSCAC.SYSMETACOLUMNS`, and `SYSCAC.SYSMETAFRAGMENTS` tables must be loaded into the eXadas System Catalog. This is done automatically by running the Meta Data Utility.

If you are creating a new set of catalogs, then when you create your first user table, the `SYSMETA*` tables are automatically created. If you have an existing set of catalogs, then you must run the Meta Data Utility to get these tables loaded into the eXadas System Catalog. You can define a new user table, create an index, perform a grant or revoke operation, or drop an existing table/index definition or a non-existent table or index definition. These table are automatically created if you supply a syntactically correct statement to the Meta Data Utility. The outcome of the execution of the Meta Data Utility has no bearing on the creation of the `SYSMETA*` table definitions, which is why you can load them, for example, by attempting to drop a non-existent table or index, or by issuing a grant or revoke for a non-existent object.

**TIP:** If you want to secure access to the SYSMETA\* or grant public access to these tables, then you can run the Meta Data Utility and supply one or more GRANT statements that reference the SYSMETA\* tables. Although initially these tables do not exist in the eXadas System Catalog, they will exist by the time the GRANT statement is processed.

Once the SYSMETA\* tables have been defined, you can load the DBMS-specific meta table definitions into the eXadas System Catalog. The DBMS-specific meta data tables are loaded into the System Catalog using the local client IVP job CACCLNT. To load the DBMS-specific meta data tables into the System Catalog, make sure that the STATIC CATALOGS configuration parameter in the master configuration member (SCACCONF member name CACDSCF) is set to 0.

**NOTE:** If STATIC CATALOGS is set to 1 and CACCLNT is run, the job will end with a condition code 4095.

The System Catalog referenced by the server must be the standard System Catalog datasets and not linear datasets. For more information about linear catalogs, see [“Static Catalogs,” on page 102](#).

**To have the server use linear catalogs:**

1. Shut the server down.
2. Run CACLCAT to allocate the linear catalogs.
3. Run CACCLNT one or more times.
4. Run CACGRANT to issue the necessary authorizations. Ensure the JCL points to the linear catalogs.
5. Update the server’s JCL to reference the linear catalogs.
6. Restart the server.

In the SCACSAMP library, there are sets of members that are used to load the DBMS-specific meta data table definitions into the eXadas System Catalog. If you do not want to use some of the tables that are automatically loaded, after they have been loaded you can delete them by issuing a DROP TABLE statement. The table below identifies the load member names:

**Table 35: Meta Data Table Load Member Names**

Database Type	Member Name
Adabas	CACOADAB
CA-DATACOM/DB	CACODCOM
DB/2	CACODB2
IDMS	CACOIDMS
IMS	CACOIMS

**Table 35: Meta Data Table Load Member Names**

Database Type	Member Name
Sequential files	CACOSEQ
VSAM files	CACOVSAM

The following steps assume that you have an environment with a working data server and that you have already configured the system and performed the local verification procedure discussed in the *eXadas OS/390 Getting Started Guide*. If you have not, then for the data source that you are interested in, perform [step 1](#) of these procedures to ensure that you have a query processor service configured, and [step 2](#) to ensure that you have a local client configured to communicate with the query processor.

**To load one or more DBMS-specific meta data tables:**

1. Make sure that the CACCAT and CACINDX DD statements are referencing the sequential file versions of the system catalog data sets.
2. Make sure that the STATICCATALOGS configuration parameter is set to 0.
3. Start the eXadas server.
4. Edit the load member name that you want to load in library SCACSAMP and change CACUSER to the name of a user that has SYSADM authority. If you have the SAF exit active in the server, the user name must be an authorized MVS user ID, and you need to change CACPWD to supply a valid password for the user ID that you supplied.
5. Edit member name CACCLNT in the SCACSAMP data set by:
  - a. Supplying a valid JOB card
  - b. Changing the CAC parameter to specify the high-level qualifier of the eXadas data sets.
  - c. Changing the CONFIG parameter to specify the name of the client configuration member (default is CACUSCF) to be used to communicate with the server.
  - d. Changing the SQLIN parameter to specify the name of the meta data table load member name that you want to load into the eXadas System Catalog.
6. Once you have completed customizing CACCLNT, submit the job.
7. Review the results.

If the CACCLNT job ends with a condition code of zeros, the requested set of DBMS-specific meta data table definitions have been loaded into eXadas System



Catalog. If you receive a non-zero return code, then an error was encountered. The following table lists typical condition codes and their corrective action.

**Table 36: Typical CACCLNT Condition Codes**

Condition Code	Corrective Action
204	The SYSCAC.SYSMETA* tables have not been installed in the eXadas System Catalog. Run the Meta Data Utility to install these tables and then re-run CACCLNT.
551	The user ID specified does not have SYSADM authority. Either grant the user ID specified with SYSADM authority, or change the user ID to an ID that does have SYSADM authority. Then re-run CACCLNT.
601	The meta data table that you are attempting to load already exists in the eXadas System Catalog. No action required.
4095	The STATIC CATALOGS configuration parameter is set to 1, or the server is referencing linear catalogs. In the first case, stop the server, change the STATIC CATALOGS setting to 0, re-start the server, and re-run CACCLNT. In the second case, stop the server, modify the CACCAT and CACINDX DD statements to reference the sequential catalogs, re-start the server, and re-run CACCLNT.

Continue changing the SQLIN member and performing [step 4](#) and [5](#) until you have loaded all of the DBMS-specific meta data tables into the eXadas System Catalog that you are interested in.

For full descriptions of the meta data tables, see [Appendix E, “Meta Table Definitions.”](#)



# 15

## System Exits

### Introduction to System Exits

The server is a multi-threaded implementation designed to service large numbers of concurrent users. eXadas includes a set of system exits for security and accounting purposes. All of the system exits are written in Assembler language, with the exception of the Record Processing Exit, and are written to execute in a multi-tasking environment. They are assembled as re-entrant. Additionally the exits are link-edited as AMODE 31 RMODE ANY and REF, RENT, RU. If you customize an exit, make sure that your version also meets these criteria.

eXadas provides support for the following system exits:

- [“Security: SAF Exit Specifics,”](#) on page 220,
- [“Accounting: SMF Exit Specifics,”](#) on page 228,
- [“CPU Resource Governor,”](#) on page 235,
- [“Workload Manager Exit,”](#) on page 242,
- [“DB2 Thread Management Exit,”](#) on page 251, and
- [“Record Processing Exit,”](#) on page 256.

The remainder of this chapter describes each of the eXadas system exits. For each exit the following information is provided:

- an overview of the functionality that the system exit implements,
- step-by-step instructions on how to activate the eXadas sample exit,
- a description of the API interface used to communicate with the system exit, and
- a description of the functions that the system exit is called to perform.

## Security: SAF Exit Specifics

The SAF Exit is used to verify that a user has authority to access a physical file or PSB referenced in an SQL query. The SAF Exit is also called to verify that a user has authority to execute a Stored Procedure program.

**NOTE:** This notice is for the benefit of users that are upgrading to a new version of eXadas who plan to run an existing version of their SAF Exit. The control block for the SAF Exit (CAC SX04) has been modified. The SAFID field has been changed from a CL4 field to a CL3 field followed immediately by a single byte field. This one byte field contains a version number. This position in the control block will not be a blank as it has been in all previous versions of this control block. These modifications do not alter position or alignment of any existing fields. The control block has also been lengthened by 6 bytes.

If you run the exit, it will run without modification. No re-assembly or re-linking is required. If you have modified the eXadas module and you reference the SAFID field as a CL4 field or if you reference SAFID with an explicitly declared length of 4, you must correct the coding to reference SAFID as a 3 byte field, re-assemble, and re-link your exit. Correcting all length references to SAFID will ensure your existing SAF Exit will execute correctly with the new eXadas product even though the new control block features will not be implemented.

To implement the new features provided by the control block modifications, review the supplied SAF Exit and include the changes into your SAF Exit as may be required.

**NOTE:** The eXadas SAF Exit issues RACROUTE calls and must be run from an APF-authorized library. Additionally, when issuing the RACROUTE calls, the exit enters key zero supervisory state. The exit reverts back to problem program state immediately after each RACROUTE call returns.

The section that follows provides a description of the steps to activate and verify the eXadas SAF Exit. This is followed by a description of the initialization, validation, and termination functions that the exit is called to perform in case you need to customize the eXadas version.

## Activating the SAF Exit

The following steps show how to configure the eXadas SAF Exit and verify that it is working.

**NOTE:** The following instructions assume that you have successfully installed eXadas, performed initial configuration, and verified the installation and configuration using the eXadas sample application and data.

1. Ensure that the SAF Exit load module (CACXSX04) is in an APF-authorized library (SCACLOAD).
2. Ensure that the Server JCL references the APF-authorized library in the STEPLIB DD statement where the SAF Exit is located (SCACLOAD). Also, ensure that any other data sets referenced in the STEPLIB DD statement are also APF-authorized.
3. Edit the Server master configuration member (SCACCONF member CACDSCF).

**NOTE:** You can also activate the SAF Exit from a service configuration member. However, CrossAccess recommends that you activate the SAF Exit from the master configuration member. Activating the SAF Exit at the master configuration member level ensures that all users accessing the server have authorization checks performed before they can access your legacy data.

4. Uncomment the SAF Exit parameter. Parameters are specified using a keyword=value format and are comma delimited. The following sub-parameters are supported by the eXadas SAF Exit:
  - **IMS CLASS=Class Name** specifies the name of the RACF resource class that is checked to determine whether the user has authority to schedule or access the PSB(s) associated with the table(s) referenced in a query. The Class Name can be up to eight characters long. This sub-parameter is required when accessing IMS data.
  - **PSB PREFIX=Prefix** specifies a value that is to be prefixed to the PSB name(s) before a RACF authorization call is issued. If specified, the PSB name is appended to the Prefix value, for example, IMSPPSB1 where the Prefix is IMSP and the PSB name is PSB1.

**NOTE:** If you are planning to access IMS data, then you may need to modify the IMS CLASS subparameter to define the RACF class where IMS PSBs are defined at your site.

To use a PSB, a user ID must have at least CONTROL access to that PSB's corresponding RACF profile within the class.

The combination of the length of the PSB name and the length of the Prefix must be eight characters or less. This is a RACF restriction. If a larger PSB name or Prefix combination is encountered, an error message is issued.

- **SPCLASS=FACILITY** specifies the name of a class to be used to check for RACF authorized use of stored procedure names defined in the Meta Data catalogs. These names are stored in the SYSIBM.SYSROUTINES system table. An SPCLASS name can be up to eight characters in length.

In this example the IBM-supplied class of FACILITY is used. You can define an installation class specifically for stored procedures to RACF. Use the FACILITY class as an example and specify the length of the resource name as 39. Replace FACILITY with the name of the new class.

The RACF administrator should define each stored procedure as a resource in this class and grant ALTER access to each user ID that will invoke the stored procedure. If the stored procedure is not defined to RACF or the user ID is not granted access, -5046295 is returned on the attempt to use a stored procedure, and the following message is added to the server log:

```
Stored Procedure Access Denied
```

- **ADACLASS=FACILITY** specifies the name of a class to be used to check for RACF-authorized use of ADABAS view names. An ADACLASS name can be up to eight characters long. In this example the IBM-supplied class of FACILITY is used. You can define an installation class specifically for ADABAS views to RACF. Use the FACILITY class as an example and specify the length of the resource name as 32. Replace FACILITY with the name of the new class. The RACF administrator should define each ADABAS view name as a resource in the class and grant CONTROL access to each user ID that uses that view name. If the ADABAS view name is not defined to RACF, or the user ID is not granted access, the following message is returned on the attempt to pull data from the ADABAS table defined with a view name:

```
Access Denied
```

If the ADABAS table is only defined with a file number (no Predict view name), you will receive the same error message as shown above, and the following message will appear in the server log:

```
CACSX04 NO ADABAS VIEW NAME IN USE GRAMMAR
```

- **EXCLUDE=n** indicates the query processor should NOT provide an ACEE address in commands sent to CA-DATACOM/DB. When the SAF Exit is active, the address of an ACEE is obtained during SAF Exit initialization. This ACEE address is normally passed to CA-DATACOM/DB in each database request and CA-DATACOM/DB authenticates the request using information within the ACEE.

Whenever the SAF Exit is active and database level security checking in CA-DATACOM/DB is to be avoided you must indicate the query processor should exclude the ACEE from the database requests that are sent to CA-DATACOM/DB. Set the value of n to 2 (Heterogeneous Query Processor Datacom Data Savant). This setting will not provide the ACEE address in the call parameters.

- **VALIDATE=Y|N** indicates when called to perform validation processing whether the exit should actually validate that the user has authority to access the database, file, stored procedure or PSB name passed to the exit. Specifying a value of Y informs the exit that it should issue RACROUTE validation call(s) for the resource name passed to the exit. Specifying a value of N informs the exit that it should not perform any validation processing for the resource name passed. The default value for VALIDATE is Y.

The purpose of this parameter is to allow you to eliminate the overhead of verifying that the user has authority to access the resource if you have elected to activate SQL security to control access to tables and stored procedures.

**NOTE:** Use both SQL security and the SAF Exit in conjunction with your site security package to restrict access to your legacy data.

5. Save the master configuration member.
6. Start the server. If your server is already running, shut it down and restart it.

**NOTE:** This operation can also be performed using the MTO Operator Interface. For more information on dynamic configuration see [Chapter 10, “Server Operations.”](#)

7. Uncomment the SELECT statement(s) in SCACSAMP member CACSQI that reflects the datasource you are using.
  - a. Ensure that the user ID is authorized to access the table that is referenced in the query.
  - b. Save the changes after you have completed editing the member.
8. Edit SCACSAMP member CACCLNT to set the SQLIN parameter to reference the name of the member that you edited in Step 7. (SQLIN).
9. Save the member once you have completed making the changes.
10. Submit CACCLNT and review the output. The query should function normally and return the expected result set.
11. Re-edit the SCACSAMP member updated in Step 7 (SQLIN). Update the user ID (CACUSER) to specify an invalid user ID or one that is not authorized to access the file or PSB that is referenced in the query.
12. Save the member once you have completed making changes.
13. Submit CACCLNT. The query should not return a result set and you should see an error message that reads: `Access denied`.
14. Re-edit the SCACSAMP member that was previously updated in Step 7 (SQLIN) and change the user ID to a valid user ID.
15. Save the member once you have completed making changes.

## SAF Exit API Overview

The parameter list passed to the SAF Exit is mapped by macro CACSXPL4 found in the SCACMAC library. The SAFFUNC field indicates the function you are calling: initialization, validation, or termination. The fields and their descriptions are shown in the table that follows.

**Table 37: SAFFUNC Fields and Descriptions**

Field	Description
&DSECT=YES	Used to control whether a DSECT definition is generated or whether the fields are generated in the exit's local storage area.
SAF DSECT	DSECT definition that is generated if &DSECT=YES is specified.
SAF DS 0H	Label that is generated when &DSECT=YES is not specified.
SAFID DC CL3'SAF'	Identifier that should be checked to determine whether the internal storage area for the CACSXPL4 parameter list has been corrupted.
SAFVER DS X	Format version identifier. This position in the parameter list was a blank (not used) prior to eXadas Version 2.2.1.
SAFORG EQU X'40'	Parameter list formatted by eXadas software prior to Version 2.2.1.
SAF42FC EQU X'F1'	Parameter list formatted by eXadas software beginning at Version 2.2.1. Modifications in the parameter list format include: <ul style="list-style-type: none"> <li>• SAFID changed from CL4 to CL3.</li> <li>• SAFVER field added.</li> <li>• SAFEXCLD field added.</li> <li>• SAFACEE field added.</li> <li>• SAFCGRP field added.</li> </ul>
SAFUSER DS A	Word available for SAF Exit use. If the SAF Exit needs to allocate storage for processing, the address of the storage area should be placed in this field during initialization processing.
SAFTYPE DS F	Validation type. The valid values are described in <a href="#">"SAF Exit Validation,"</a> on page 226.
SAFUSERI DS A	Address of the user ID to be validated. The user ID referenced is eight bytes long, left-justified, upper-case, and padded with blanks.
SAFUSERP DS A	Address of the User Password to be validated. The User Password referenced is eight bytes long, left justified, upper case and padded with blanks.



**Table 37: SAFFUNC Fields and Descriptions**

SAFNAME DS A	Address of the file name/PSB/Stored Procedure table name to be checked for access authority. The name is left-justified and terminated with a NULL. See “SAF Exit Validation,” on page 226, for the contents of the SAFNAME field for each type of validation call that can be issued.
SAFFUNC DS H	Function identifier flag.
SAFINIT EQU 0	Initialization.
SAFVAL EQU 4	Validation.
SAFTERM EQU 8	Termination.
SAFEXCLD DS H	Exclusion flags. Identifies which query processor should NOT provide the ACEE address to CA-DATACOM/DB in each command request.
SAFDCHQP EQU 2	The Query Processor (running a Datacom query) should not supply the ACEE address to CA-DATACOM/DB.
SAFACEE DS A	The address of the ACEE created during SAF Exit initialization. This ACEE was created by a RACROUTE ENVIR=CREATE call and is used by CA-DATACOM/DB to authenticate database requests based upon the user ID in the ACEE.
SAFCGRP DS CL8	<p>Allows the exit, when called for initialization processing, to return a user group name this will subsequently be used in SQL security processing. Using group names for SQL security processing allows you to simplify the administration of SQL security since you only need to grant authority to the group name for the tables, views and stored procedures members of that group are allowed to access.</p> <p>On the initialization call, this field is set to spaces. On subsequent calls do not attempt to update this field since, after initialization processing, the contents of this field are copied into an internal control block for use is SQL security processing.</p>

### SAF Exit Initialization

This function is called immediately after the exit is loaded during initialization of the Query Processor Task when a user connects to the service. The SAFNAME parameter points to an (optional) input parameter string that can be passed to the exit. The input parameters are specified on the SAF Exit configuration parameter used to invoke the SAF Exit. On the SAF Exit parameter additional subparameters can be placed after the name of the exit.

The SAFUSERI and SAFUSERP fields contain the User ID and password of the user that is connecting to the service.

When called at this point, the exit should perform initialization processing and allocate any storage or other resources that are required for validation processing. A pointer to the anchor for these resources can be placed in the SAFUSER field of the parameter list. This pointer is preserved and passed to the exit on subsequent invocations.

For example, the eXadas SAF Exit validates that the User ID and password are valid and creates an ACEE for the user. The address of the ACEE is returned to the caller in the SAFACEE field of the parameter list. If the optional input sub-parameter EXCLUDE= is included in the SAF Exit configuration parameter, the specified value is returned to the caller in the SAFEXCLD field of the parameter list. These two fields are provided to implement database level security checking within CA-DATACOM/DB. This ACEE is used during validation processing and is destroyed at exit termination.

Additionally, the eXadas SAF Exit inspects the ACEEGRPL field in the ACEE. If the length field is greater than zero, the contents of the ACEEGRPN field is copied to the SAFCGRP for subsequent use in SQL security processing.

If the exit returns a non-zero return code, Query Processor initialization is halted, the user is disconnected, and the return code issued by the exit is returned to the client application.

## SAF Exit Validation

This exit function is called at different predetermined processing points. The SAFTYPE field identifies the type of validation to be performed. Descriptions of the situations that cause a call to the exit for validation processing for each SAFTYPE follow.

- **SAFIMS:** For IMS access using a DBB/BMP Data Savant interface the exit is called immediately before PCB selection logic is invoked. The name of the PSB that is referenced by the SAFNAME field is the PSB specified in the Server's JCL. The exit should verify that the user has authority to use the specified PSB name identified in the SAFNAME field.

If the exit returns a non-zero return code, processing for the query is terminated and the return code issued by the exit is returned to the client application. The application can still issue another SQL request.

When the DRA interface is used to access IMS data the exit can be called at two different points. The exit is always called immediately before a PSB is to be scheduled. The exit should validate that the user has authority to use the PSB referenced by the SAFNAME field.

If the exit returns a non-zero return code, the PSB is not scheduled, processing for the query is terminated, any other PSBs that have been scheduled for the query are unscheduled, and the return code issued by the exit is returned to the client application. The application can still issue another SQL request.

The exit may also be called when the query contains a JOIN. In this situation, the DRA interface schedules the JOIN PSB specified in the Meta Data Grammar for a table referenced in the query. The exit is invoked as previously

described. For subsequent tables in the JOIN, the Data Savant checks to determine whether the PSB(s) that have already been scheduled contain(s) a PCB that can be used to access the table. If a usable PCB is located, the SAF Exit is called with the name of the primary PSB (as specified in the Meta Data Grammar for the table that is referenced in the query). This PSB is not scheduled, however, an authorization check should be performed to verify that the user has authority to access the primary PSB (referenced by the SAFNAME field) associated with the table.

If the exit returns a non-zero return code, the other PSB(s) that have been scheduled for the query are unscheduled and the return code issued by the exit is returned to the client application. The application can still issue another SQL request.

**NOTE:** There is no indication as to which of the three processing sequences caused the exit to be invoked.

- **SAFVSAM and SAFSEQ:** When a query references a Sequential or VSAM file, the exit is called immediately before the file is to be opened. The exit should validate that the user has authority to access the file name referenced by SAFNAME. When a PDS member is being referenced, the name of the member is not passed to the exit.

If the exit returns a non-zero return code, the file is not opened, processing for the query is terminated, and the return code issued by the exit is returned to the client application. The application can still issue another SQL request.

- **SAFSP:** For stored procedures, the SAF Exit is invoked immediately before the application program associated with a stored procedure definition is loaded for execution. The SAF Exit should validate that the user has authority to execute the program. This is performed indirectly based on the stored procedure table name referenced by the SAFNAME.

If the exit returns a non-zero return code, the program is not loaded or executed, processing for the stored procedure request is terminated, and the return code issued by the exit is returned to the client application. The application can still issue another SQL request.

**NOTE:** The eXadas SAF Exit will bypass performing the above processing if the VALIDATE sub-parameter is set to N. This option should only be activated when you are using SQL security to control access to the tables and stored procedures.

- **SAFADAB:** When a query references an ADABAS database, the exit is called immediately before the database is accessed. The exit should validate that the database has a viewname defined in the tables USE grammar and that the user has authority to access the viewname. The viewname is referenced by SAFNAME.

If the exit returns a non-zero return code, the database is not accessed, processing for the query is terminated, and the return code issued by the exit is returned to the client application. The application can still issue another query.

## SAF Exit Termination

This exit function is called during Query Processor termination processing when the user disconnects from the service task. At this time, the exit can perform any termination processing necessary and must free any resources it has allocated.

# Accounting: SMF Exit Specifics

The SMF Exit is used to report wall-clock time and the CPU time for an individual user session with a Query Processor Task. Additionally, if SQL security is active and an authorization violation is detected by the Query Processor, the exit is called to allow it to log the authorization violation.

**NOTE:** The eXadas SMF Exit writes this information out as user SMF records. SMF requires that an application that writes SMF records be run out of an APF-authorized library.

The following sections describe the steps to activate and verify the eXadas SMF Exit. This is followed by a description of the initialization, recording, and termination-exit functions that the exit performs to allow you to customize the eXadas version.

## Activating the SMF Exit

The following instructions assume that you have successfully installed eXadas, performed initial configuration, and verified the installation configuration using the eXadas sample application and data.

### To configure the eXadas SMF Exit and verify that it is working:

1. Ensure that the SMF Exit load module (CACSX02) is in an APF-authorized library (SCACLOAD).
2. Ensure the Server JCL references the APF-authorized library in the STEPLIB DD statement where the SMF Exit is located (SCACLOAD).
3. Ensure that any other data sets referenced in the STEPLIB DD statement are also APF-authorized.
4. Edit the Server master configuration member (SCACCONF member CACDSCF).

**NOTE:** You can also activate the SMF Exit from a service configuration member. However, CrossAccess recommends that you activate the SMF Exit from the master configuration member. Activating the SMF Exit at the master configuration member level ensures that all

users accessing the Server have accounting information recorded for them.

5. Uncomment the SMF EXIT parameter. The following sub-parameters are supported by the eXadas SMF Exit:
  - a. RECTYPE=nnn: This is a required parameter that defines the SMF user record type. This parameter contains a numeric value between 128 and 255.
  - b. SYSID=xxxx: This is a required parameter that contains the primary JES subsystem ID. SYSID can be a maximum of four characters long.

6. Save the master configuration member.

7. Start the server.

If your server is already running, shut it down and restart it.

**NOTE:** This operation can also be performed using the MTO Operator Interface. For more information on dynamic configuration see [Chapter 10, “Server Operations.”](#)

8. Uncomment the SELECT statement(s) in SCACSAMP member CACSQ1 that reflects the datasource you are using.
  - a. Ensure that the user ID is authorized to access the table that is referenced in the query.
  - b. Save the changes after you have completed editing the member.
9. Edit SCACSAMP member CACCLNT to set the SQLIN parameter to reference the name of the member that you edited in [step 8 \(SQLIN\)](#). Save the member once you have completed making the changes.
10. Submit CACCLNT and review the output. The query should function normally and return the expected result set.
11. Ensure that the SMF record file exists, for example, SYS1.MAN1.
12. DUMP the eXadas-related SMF records into a data set. Sample JCL is shown in the following example.

```

/*INSERT VALID JOB CARD HERE
//STEP1 EXEC PGM=IFASMFDP
//INDD1 DD DISP=SHR,DSN=SYS1.MAN1
//OUTDD1 DD DISP=(NEW,CATLG),DSN=CAC.SMFDDUMP,
// UNIT=SYSDA,VOL=SER=XXXXXX,SPACE=(TRK,(5,5),RLSE),
// DCB=(LRECL=32760,BLKSIZE=27998,RECFM=VBS,DSORG=PS)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
INDD(INDD1,OPTIONS(DUMP))
OUTDD(OUTDD1,TYPE(XXX))

```

Where *Type (xxx)* is the eXadas record type as specified in the RECTYPE= parameter.

13. Run SAS, IDCAMS, or any other tool that processes SMF records. If you run IDCAMS, specify the following SYSIN:

```
//STEP2 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
PRINT INDATASET (CAC.SMFDUMP)
```

Where *CAC.SMFDUMP* is the name of the DUMP file.

14. Verify the output.

The following table describes the DSECT that maps the SMF accounting routine output data from the eXadas SMF Exit.

**Table 38: SMF Accounting File DSECT Field Definitions**

Field	Definition	Length (in bytes)
CACSXSMF DSECT	Data structure for the SMF record.	N/A
RDWLEN DS H	Length of the record.	2
RDWSPAN DS H	Reserved for system use.	2
FLG DS X	Reserved for SMF use.	1
RECTYPE DS AL1	Value specified in the REC TYPE= parameter. This is the SMF record type.	1
ENTIME DS BL4	Time in BIN format from TIME macro. This is the time the event ended.	4
ENDATE DS PL4	Date in BIN format from the TIME macro. This is the date the event ended.	4
SYSID DS CL4	JES subsystem ID from the SYSID= * Parameter.	4
USRTYPE DS BL2	Zero for CPU time and elapsed time or for authorization violation the type of violation being reported. See <a href="#">Table 39, "Authorization Violation Type Codes,"</a> on page 231, for the different types of authorization violations that can be reported.	2
USERID DS CL8	SQL ID from AXPLSQID. Padded with blanks.	8
STTIME DS BL4	Time in BIN format from the TIME macro. This is the time the event started.	4
STDATE DS PL4	Date in BIN format from the TIME macro. This is the date the event started.	4
AGCPU DS BL4	Total CPU time used since the event started. The value is represented in milliseconds.	4
RECLN EQU *-CACSXSMF	Length of the standard SMF reporting record.	
ORG STTIME	Alternate record definition for authorization violation reporting.	

**Table 38: SMF Accounting File DSECT Field Definitions**

OBJNAME DS CL27	Name of the object for which the user is not authorized to access, define, or grant/revoke authority for.	27
TOTLEN EQU *- CACXSXSMF	Length of the authorization violation SMF record.	

For more information on SMF, see the *IBM OS/390/ESA System Management Facilities (SMF)* documentation.

**Table 39: Authorization Violation Type Codes**

Type Code	Authorization Violation Type
101	User is not authorized to issue a DROP TABLE for the requested table.
102	User is not authorized to issue a DROP INDEX for the requested table.
103	User is not authorized to issue a DROP VIEW for the requested view.
104	User is not authorized to issue a DROP PROCEDURE for the requested stored procedure.
200	User is not authorized to create the requested table.
201	User is not authorized to create the requested index
202	User is not authorized to issue the CREATE VIEW statement.
203	User is not authorized to issue the CREATE PROCEDURE statement
300	User is not authorized to issue a SELECT statement for the requested table or view.
301	User is not authorized to issue an UPDATE statement for the requested table.
302	User is not authorized to issue an INSERT statement for the requested table.
303	User is not authorized to CALL the requested stored procedure.
403	User is not authorized to issue a DELETE statement against the requested table.
500	User is not authorized to issue the requested GRANT statement.
501	User is not authorized to issue the requested REVOKE statement.

## SMF Exit API Overview

The parameter list passed to the SMF Exit uses an AVA (Access Validation and Accounting) interface and is mapped by macro CACSXPL, which is found in the SCACMAC library.

The AXPLFUNC field indicates the function you are calling:

- Initialization,
- Accounting,
- Authorization Violations and,
- Termination.

The fields and their descriptions are shown in the table that follows.

**Table 40: SMF Field Definitions**

Field	Definition
&DSECT=YES	Used to control whether a DSECT definition is generated or whether the fields are generated in the exit's local storage area.
AXPL DSECT	DSECT definition that is generated if &DSECT=YES is specified.
AXPL DS 0H	Label that is generated when &DSECT=YES is not specified.
AXPLID DC CL4'AXPL'	Identifier that should be checked to determine whether the internal storage area of the AXPL parameter list has been corrupted.
AXPLUSER DS 3A	Word(s) available for SMF Exit use.
AXPLETYP DS F	Event type.
AXPLESUB DS F	Event sub-type.
AXPLESEQ DS F	Event sequence.
AXPLEBEF EQU 1	Before event execution.
AXPLEAFT EQU 2	After event execution.
AXPLSQID DS CL8	User ID left-justified, blank-filled.
AXPLPENV DS A	Pointer to event specific information.
AXPLTEXT DS A	Pointer to a text buffer (this is usually an SQL statement or the name of the object for which authorization failed).
AXPLTXTL DS F	The decimal length of the text buffer.
AXPLSQLC DS F	The SQLCODE from the SQLCA after processing for the SQL event is completed.
AXPLFUNC DS H	Function identifier flag.
AXFNINIT EQU 0	Initialization.
AXFNVALI EQU 4	Validation/Accounting.
AXFNTERM EQU 8	Termination.



## Initialization

This function is called immediately after the exit is loaded during initialization of the Query Processor Task when a user connects to the service. The AXPLPENV parameter points to an (optional) input parameter string that can be passed to the exit. The input parameters are specified on the SMF EXIT configuration parameter used to invoke the SMF Exit. On the SMF EXIT parameter additional sub-parameters can be placed after the name of the exit.

When called at this point, the exit should perform initialization processing and allocate any storage or other resources that are required for validation/accounting processing. A pointer to the anchor for these resources can be placed in the AXPLUSER field of the parameter list. This pointer is preserved and passed to the exit on subsequent invocations. Upon initialization, the AXPLSQID parameter contains the user ID of the user connecting to the Query Processor. The contents of the other fields in the parameter list are indeterminate.

If the exit returns a non-zero return code, Query Processor Task initialization is halted, the user is disconnected, and the return code issued by the exit is returned to the client application.

## Validation/Accounting

This exit function is called at predetermined processing points. The AXPLETYP, AXPLESUB, and AXPLESEQ fields uniquely identify each processing point. The SMF Exit is called to process SQL events that are identified by an AXPLETYP value of 3. The AXPLESUB field identifies the type of SQL statement that the client application has issued. The client application can issue the following types of SQL statements:

**Table 41: SQL Statement Types**

AXPLESUB Value	Equate Value	Type of SQL Statement
1	DYNEXEC	Dynamic execute. When called at this point, the AXPLTEXT and AXPLTXTL fields will be zero.
2	CLOSE	Close cursor. When called at this point, the AXPLTEXT and AXPLTXTL fields will be zero.
3	DESCRIBE	Describe. When called at this point, the AXPLTEXT and AXPLTXTL fields will be zero.
4	EXECIMED	Execute immediate. When called at this point, the AXPLTEXT field will reference the statement being executed and the AXPLTXTL field will identify the statements length.
5	EXECUTE	Execute. When called at this point, the AXPLTEXT field will reference the statement being executed and the AXPLTXTL field will identify the statements length.

**Table 41: SQL Statement Types**

6	FETCH	Fetch cursor. When called at this point, the AXPLTEXT and AXPLTXTL fields will be zero.
7	OPEN	Open cursor. When called at this point, the AXPLTEXT field will reference the statement being executed and the AXPLTXTL field will identify the statements length if the client application is using static SQL. If the application is using dynamic SQL the AXPLTEXT and AXPLTXTL fields will be zero.
8	PREPARE	Prepare statement. When called at this point, the AXPLTEXT field will reference the statement being executed and the AXPLTXTL field will identify the statements length.
9	SLCTINTO	Select into. When called at this point, the AXPLTEX field will reference the statement being executed and the AXPLTXTL field will identify the statements length.

The AXPLESEQ field identifies whether the exit is being called before (1) (AXPLEBEF) or after (2) (AXPLEAFT) the SQL statement has been processed. The exit is called for each SQL statement issued by the client application. The user has control of the Query Processor service for the duration of the SQL statements execution.

Therefore, when called before the SQL statement is processed, the exit should obtain the TCB time for the current TCB. When called after the SQL statement has been processed, the exit should obtain the current TCB and compute the difference between the after and before SQL statement processing times. This value must be added to an aggregate value that the exits need to maintain for all SQL statements issued by the client application.

For example, the following sequence of SQL statements are issued for a dynamic SQL SELECT query:

- PREPARE,
- OPEN,
- DESCRIBE,
- FETCH (until a SQLCODE of 100 is returned), and
- CLOSE.

To obtain the correct CPU time for the query the exit needs to compute the CPU time used for each of these statements and add them together. Depending on the type of client application different types of SQL statements may be issued. In the case of a client application that has more than one cursor open at a time, the individual SQL statements that are issued by the client application will be interleaved.

For this exit, the type of SQL statement being issued is not important unless the exit wants to capture the text of the SQL statement being issued by the client

application. In those situations, the “Type of SQL Statement” column in [Table 41, “SQL Statement Types,”](#) on page 233, identifies when the AXPLTEXT and AXPLTXL fields are being populated.

**NOTE:** If the exit returns a non-zero return code, Query Processor Task processing of the query is halted and the return code issued by the exit is returned to the client application.

## Authorization Violations

The exit is called if SQL security is active and the Query Processor detected an authorization violation. The AXPLETYP contains a value of 5. The AXPLESUB field identifies the type of authorization exception. The AXPLTEXT field identifies the name of the object for which authorization failed and the AXPLTXL field contains the length of the name in AXPLTEXT. The AXPLESUB field contains one of the values identified in [Table 39, “Authorization Violation Type Codes,”](#) on page 231.

When called at this point the exit can report the violation exception. The eXadas SMF generates the alternate form of the SMF record for authorization violations identified in [Table 38, “SMF Accounting File DSECT Field Definitions,”](#) on page 230.

## Termination

This exit function is called during Query Processor Task termination processing when the user disconnects from the service task. At this time, the exit can perform any termination processing necessary and must free any resources it has allocated.

For example, the eXadas SMF Exit generates the SMF user record to report the CPU time used in milliseconds. The SMF record also contains the clock time and date when the user connected to and disconnected from the Query Processor Task.

# CPU Resource Governor

eXadas execution governor limits are based on the number of data rows examined (which is the number of calls issued to the Data Savant interface) and the number of rows returned in a result set after all query post-processing has been completed. These are fairly coarse governors and depending on the query, a large amount of CPU time can be expended before one of these limits is reached. eXadas includes a CPU Resource Governor Exit that can be used to restrict the amount of CPU time that a user can use for a unit-of-work.

When the CPU Resource Governor Exit is activated it is passed the CPU time that the user is allowed to use. Periodically, the CPU Resource Governor Exit is called to check to see how much CPU time has been used. Once the allotted time is

exceeded the exit returns a return code that stops the query. The frequency at which the exit is called is controlled by eXadas.

The CPU Resource Governor Exit needs to be designed for use with applications using dynamic or static SQL. The exit is responsible for determining the unit-of-work based upon the series of SQL statements issued by the client application.

For example, the eXadas CPU Governor Exit uses the following rules:

- For dynamic SQL applications, the unit-of-work is from the first received PREPARE until all cursors have been closed. For a typical client application that only issues one query at a time, the unit-of-work is the duration of a single query. If multiple cursors are opened the unit-of-work is until all cursors have been closed. Any “additional” SQL statements issued by the client while a query is active, for example, a SELECT INTO, are treated as part of the unit-of-work.
- For static SQL client applications, the unit-of-work is from the first OPEN CURSOR request until all cursors have been closed. The exit assumes that the client application is using static SQL if an OPEN CURSOR request is received without a PREPARE being issued immediately before the OPEN CURSOR request is received.
- If no queries are active, for example, no cursors are open, the exit treats any other SQL request, for example, SELECT INTO, EXECUTE IMMEDIATE, as a single unit-of-work.

The following sections describe the steps that you need to perform to activate and verify the eXadas CPU Resource Governor Exit. This is followed by a description of the initialization, validation, and termination exit functions that the exit is called to perform in case you need to customize the eXadas version.

## Activating the CPU Resource Governor Exit

The following steps describe how to configure the eXadas CPU Resource Governor Exit and validate that it is operational.

**NOTE:** The following instructions assume that you have successfully installed eXadas, performed initial configuration, and verified the installation configuration using the eXadas sample application and data.

1. Ensure that the CPU Resource Governor Exit load module (CACSX03) is in the load library that the Server is using. Based on the following examples, the CPU Resource Governor Exit needs to be in an APF-authorized library (SCACLOAD).

**NOTE:** The eXadas CPU Resource Governor Exit does not require APF authorization. However, the eXadas SAF, SMF, and Work Load Manager Exits require APF authorization. If you are running any of

these other exits, then the CPU Resource Governor Exit must also be placed in an APF-authorized load library.

2. Ensure the Server JCL references the APF-authorized library in the STEPLIB DD statement where the CPU Resource Governor Exit is located (SCACLOAD). Ensure that any other data sets referenced in the STEPLIB DD statement are also APF-authorized.
3. Edit the service configuration member for the CACSAMP data source (SCACCONF member CACQPCF).

**NOTE:** You can also activate the CPU Resource Governor Exit from the master configuration member, however, CrossAccess recommends that you activate the CPU Resource Governor Exit from the service configuration member(s). Activating at the service level allows different services (data sources) to run queries with different performance profiles. For example, if you are running production queries from one data source you can set the CPU time limits to one value, while another data source running ad hoc queries, which may take longer than production queries, would have a higher time limit set.

4. Uncomment the CPU GOVERNOR parameter.
  - a. CPU Resource Governor Exit Name.
  - b. Maximum CPU Time: Required field that specifies the maximum amount of CPU time that a single query can take, or a group of queries can take, in a multiple cursor situation. The following are valid values:
    - nS: the number of seconds where  $n = number$  is a value between 1 and 6000.
    - nM is the number of minutes where  $n = number$  is a value between 1 and 6000.
5. Save the service configuration member.
6. Start the server. If the server is already running then stop and restart the Query Processor service for data source CACSAMP. This can be done using the MTO Interface. For more information on dynamic configuration see [Chapter 10, "Server Operations."](#)
7. Uncomment the SELECT statement(s) in SCACSAMP member CACSQL that reflects the datasource you are using.
  - a. Ensure that the user ID is authorized to access the table that is referenced in the query.
  - b. Save the changes after you have completed editing the member.
8. Edit SCACSAMP member CACSQL to set the SQLIN parameter to reference the name of the member you edited in [step 7](#) (CACSQL).
9. Save the member.
10. Submit CACCLNT and review the output. The query should function normally and return the expected result set.

11. Re-edit the service configuration member for the CACSAMP data source (SCACCONF member CACQPCF). Update the CPU time limit specified on the CPU GOVERNOR parameter. Try setting the limit to 1 second (1S). Depending on the size of your OS/390 machine, the query may take less than a CPU second to execute. The eXadas exit has a minimum limit of one CPU second. Once you have completed making changes save the member.
12. Stop and restart the server.
13. Alternately you can stop and restart the Query Processor service for data source CACSAMP using MTO Interface. For more information on dynamic configuration, see [Chapter 10, “Server Operations.”](#)
14. Submit CACCLNT.  

The query should not return a result set and you should see the following error messages:

```
CPU time exceeded. If the time limit is not exceeded you will see the normal result set.
```
15. Re-edit the service configuration member for the CACSAMP data source (SCACCONF member CACQPCF).
16. Increase the CPU limit specified on the CPU GOVERNOR parameter so that you can run normal queries, or comment out the CPU GOVERNOR parameter.
17. Save the member.
18. Stop the Server and restart it.
19. Alternately you can stop and restart the Query Processor service for data source CACSAMP using MTO Interface. For more information on dynamic configuration, see [Chapter 10, “Server Operations.”](#)

## CPU Resource Governor Exit API Overview

The parameter list passed to the CPU Resource Governor Exit uses an AVA (Access Validation and Accounting) interface and is mapped by the macro CACSXPL, which is found in the SCACMAC library. The AXPLFUNC field indicates the function you are calling: initialization, accounting, or termination.

The fields and their descriptions are shown in [Table 42, “AXPLFUNC Fields and Descriptions.”](#)

**Table 42: AXPLFUNC Fields and Descriptions**

Field	Description
&DSECT=YES	Used to control whether a DSECT definition is generated or whether the fields are generated in the exit's local storage area.
AXPL DSECT	DSECT definition that is generated when &DSECT=YES is specified.
AXPL DS 0H	Label that is generated when &DESCT=YES is not specified.
AXPLID DC CL4'AXPL'	Identifier that should be checked to determine whether the internal storage area of the AXPL parameter list has been corrupted.
AXPLUSER DS 3A	Word(s) available for CPU Resource Governor Exit use.
AXPLETYP DS F	Event type.
AXPLESUB DS F	Event sub-type.
AXPLESEQ DS F	Event sequence.
AXPLEBEF EQU 1	Before event execution.
AXPLEAFT EQU 2	After event execution.
AXPLSQID DS CL8	User ID left-justified, blank-filled.
AXPLPENV DS A	Pointer to event specific information.
AXPLTEXT DS A	Pointer to a text buffer (this is usually an SQL statement).
AXPLTXTL DS F	The decimal length of the text buffer.
AXPLSQLC DS F	The SQLCODE from the SQLCA after processing for the SQL event is completed.
AXPLFUNC DS H	Function identifier flag.
AXFNINIT EQU 0	Initialization.

**Table 42: AXPLFUNC Fields and Descriptions**

AXFNVALI EQU 4	Validation/Accounting.
AXFNTERM EQU 8	Termination.

### Initialization

This function is called immediately after the exit is loaded during initialization of the Query Processor Task when a user connects to the service. The AXPLPENV parameter points to an (optional) input parameter string that can be passed to the exit. The input parameters are specified on the CPU GOVERNOR configuration parameter that is used to invoke the CPU Resource Governor Exit. On the CPU GOVERNOR parameter additional sub-parameters can be placed after the name of the exit.

When called at this point, the exit should perform initialization processing and allocate any storage or other resources that are required for validation processing. A pointer to the anchor for these resources can be placed in the AXPLUSER field of the parameter list. This pointer is preserved and passed to the exit on subsequent invocations. Upon initialization, the AXPLSQID field contains the user ID connecting to the Query Processor. The contents of the other fields in the parameter list are indeterminate.

If the exit returns a non-zero return code, Query Processor Task initialization is halted, the user is disconnected and the return code issued by the exit is returned to the client application.

### Validation/Accounting

This exit function is called at predetermined processing points. The AXPLETYP, AXPLESUB, and AXPLESEQ fields uniquely identify each processing point. The exit is called for each SQL statement issued by the client application. These events are identified by an AXPLETYP of 3. The different types of SQL statements are identified in field AXPLESUB and are documented in [Table 41, “SQL Statement Types,” on page 233](#). The AXPLESEQ field identifies whether the exit is called before (1) or after (2) the SQL statement has been processed.

When called for SQL events, the exit must determine (based on the SQL statement type) whether the client application is beginning a unit-of-work, ending a unit-of-work or is in the “middle” of a unit-of-work.



The following actions should be performed:

- Start unit-of-work: Reset aggregate CPU time and capture current CPU time and enter the “middle” or the unit-of-work.
- End unit-of-work: Reset aggregate CPU time and prepare to start the next unit-of-work.
- Middle of unit-of-work: When the exit determines that it is in the middle of a unit-of-work then before the SQL statement is issued the current CPU time should be obtained. When the exit is called after the SQL statement has been executed, the exit should capture the current CPU time and compute the amount of time taken by that SQL statement. The exit needs to maintain an aggregate CPU time for the unit-of-work, which represents the sum of the individual SQL statement execution times. The exit should check this aggregate time after an SQL statement has been processed to determine whether the CPU time limit has been exceeded.

When called to process SQL statements, the exit needs to take into consideration the following situations:

- If the exit has already generated a “CPU time exceeded” error when it was called during machine execution (described below), it should not generate another error. The exit will still be called to perform “after SQL statement” processing and needs to set some kind of flag that indicates an error has already been reported.
- If after an SQL OPEN statement has been processed the AXPLSQLC field contains a non-zero value an error was detected. Typically this will be a –204 or –206 error, or may be an error generated by another system exit or an eXadas-generated error. The exit will not be subsequently called to close the cursor associated with the statement that failed. In these situations the exit must check to see what “state” the unit-of-work is in to determine whether it should reset its unit-of-work state. If there are no other cursors open, then the exit needs to perform end unit-of-work processing.

The exit will also be called while the SQL program (that was generated for the query) is executing. The AXPLETYP field will contain a 4 in these situations. Currently, the exit is called at the two points where the MAX ROWS EXAMINED and MAX ROWS RETURNED governor checks are performed. The AXPLESUB field will contain the machine instruction number that identifies the instruction being executed and the AXPLESEQ contains a 1.

The exit should not be concerned about the instruction type being executed. When called with an AXPLETYP value of 4, the exit should capture the current CPU time and compute how much time has elapsed since processing of the SQL statement started. This value should be (temporarily) added to the aggregate CPU time for the current unit-of-work. If the time limit is exceeded the exit should issue a return code to halt query processing.

**NOTE:** The CPU Resource Governor Exit uses an AVA interface, which is more complex than a custom interface like the SAF interface. In the future eXadas may extend additional call points in order to make the CPU Resource Governor more granular for tracking the amount of time used for the current unit-of-work. The AVA interface allows additional calls points to easily be accommodated.

If the exit returns a non-zero return code, Query Processor Task terminates execution of the current query and the return code issued by the exit is returned to the client application.

### Termination

This exit function is called during Query Processor Task termination processing when the user disconnects from the service task. At this time, the exit can perform any termination processing necessary and must free any resources it has allocated.

## Workload Manager Exit

The Workload Manager Exit (WLM) is used to interface with the OS/390 Work Load Manager. The WLM Exit can be used to track units-of-work and manage that work in goal or compatibility mode.

A unit-of-work from an eXadas perspective is the execution of an SQL statement that a client application issues. [Table 41, “SQL Statement Types,” on page 233](#), identifies the different types of SQL statements that a client application can issue. When the client issues one of these SQL statements, that user has control of the Query Processor service thread for the period that the Query Processor takes to service that SQL statement.

The eXadas WLM Exit supports most of the parameters accepted by the IWMCLSFY macro in order to identify the service class that will be used to manage and/or report on for the individual units-of-work. The eXadas WLM Exit classifies these units-of-work at the Query Processor SIE level during Query Processor TCB initialization processing. Therefore, all users being serviced for a data source are managed in the same service class. Exit points are available at the user connection level that will allow a customized exit to manage individual users of a Query Processor service, however, for the eXadas WLM Exit, no processing is performed at these exit points.

**NOTE:** The eXadas WLM Exit issues WLM macros that use “enclave task support.” Enclave tasks were added to OS/390 version 2 release 4. This is the minimum OS/390 operating system level required to execute the eXadas WLM Exit. Additionally, the WLM macros require the WLM Exit be executed out of an APF-authorized load library. Therefore, the Server and all associated load modules must reside in an APF-authorized load library.

The next section provides a description of the steps required to activate and verify the eXadas WLM Exit. This is followed by a description of the initialization, validation, and termination exit functions that the exit is called to perform, should you need to customize the version.

## Activating the WLM Exit

The following steps describe how to use the eXadas WLM Exit to report unit-of-work activities and manage queries in WLM goal mode.

**NOTE:** The following instructions assume that you have successfully installed eXadas, performed initial configuration and verified the installation/configuration using the eXadas sample application and data.

1. Ensure that the WLM Exit load module (CACSX06) is in the load library that the Server is using. Based on the previous examples, the WLM Exit must be in an APF-authorized library (SCACLOAD).
2. Make sure the Server JCL references the APF-authorized library in the STEPLIB DD statement where the WLM Exit is located (SCACLOAD). Ensure that any other data sets referenced in the STEPLIB DD statement are also APF-authorized.
3. Edit the Server master configuration member (SCACCONF member CACDSCF).

**NOTE:** The WLM Exit is activated using a SERVICE INFO ENTRY parameter to initialize the exit once when the address space is initialized. The WLM Exit must be able to handle all of the users accessing the Server concurrently.

4. Uncomment the WLM SERVICE INFO ENTRY parameter. Parameters are specified using a keyword=value format and are comma-delimited. The following sub-parameters are required by the eXadas WLM Exit and must be supplied in the task data field on the WLM SERVICE INFO ENTRY parameter:
  - a. Exit Name: Specifies the name of the WLM Exit to be invoked. The eXadas WLM Exit name is CACSX06. Any parameters that the exit needs can be specified in the task data field (after the exit name) on the WLM SERVICE INFO ENTRY parameter. The exit name and its input parameters must be separated by a space or comma.
  - b. SUBSYS=xxxx : Specifies the generic subsystem type under which the unit-of-work is reported in WLM. The SUBSYS type can be up to four bytes in length and must conform to WLM subsystem types. Modify the subsystem type to a type that is valid on your system. Examples are: IMS, CICS, JES, and STC. If the address space of the Query Processor is a started task STC may be a good choice. You may find it best to define a subsystem type to WLM just for eXadas.

- c. **SUBSYSNM=xxxxxxx** : Specifies the name of the subsystem that the unit-of-work is reported under in WLM. The SUBSYSNM name can be up to eight bytes in length. This name and SUBSYS (subsystem type) are used to connect to WLM and classify work received. Using IMS as example:

```
SUBSYS=IMS  SUBSYSNM=IMSA .
```

**WARNING:** The WLM SERVICE INFO ENTRY definition must come before any Query Processor SIEs. Failure to do so will result in a S0C4 abend when the Server is stopped. Additionally, the WLM SIE cannot be stopped if any Query Processor services are active. If the WLM SIE is stopped and a Query Processor service (task) is active, a S0C4 abend will occur when the Query Processor task is stopped.

These abends occur when the WLM Exit is called during WLM service initialization processing, which allocates and references an address space-level control block when the WLM Exit is called from a Query Processor task. When the WLM service is terminated, the address space-level control block is freed. During normal shutdown, the Server terminates tasks in LIFO (last-in, first-out) sequence. The WLM SIE must be defined before any Query Processor SIEs to allow the WLM service to terminate after all Query Processor tasks have completed termination processing.

5. Save the master configuration member.
6. Edit the service configuration member for the CACSAMP data source (SCACCONF member CACQPCF).

**NOTE:** You can also supply WLM unit-of-work information from the master configuration member, however, CrossAccess recommends that you supply unit-of-work information from the service configuration member(s). Supplying this information at the service level allows different services (data sources) to run queries with different performance profiles. For example, if you are running short queries you can give them more resources. For longer running queries you can use period switching to reduce the rate that these types of queries use resources.

7. Uncomment the WLM UOW parameter. Subparameters are specified using a keyword=value format and are comma delimited. All subparameters are optional. The following subparameters are supported by the eXadas WLM Exit:
  - a. **ACCTINFO=xxx...** : Specifies accounting information. A maximum of 143 characters of accounting information can be supplied. The default is NO\_ACCTINFO.
  - b. **COLLECTION=xxx...** : Specifies a customer defined name for a group of associated packages. The maximum collection name that is supported is 64 characters long. The default is NO\_COLLECTION.

- c. `CORRELATION=xxx...` : Specifies the name associated with the user/program creating the work request, which may reside anywhere within the network. The maximum correlation name that is supported is 64 characters long. The default is `NO_CORRELATION`.
  - d. `LUNAME=xxxxxxxx` : Specifies the name of the local LU name associated with the requestor. The maximum LU name is 8 characters long. The default is `NO_LUNAME`.
  - e. `NETID=xxxxxxxx` : Specifies the network identifier associated with the requestor. The maximum identifier is 8 characters long. The default is `NO_NETID`.
  - f. `PACKAGE=xxxxxxxx` : Specifies the package name for a set of associated SQL statements. The maximum package name is 8 characters long. The default is `NO_PACKAGE`.
  - g. `PERFORM=xxxxxxxx` : Specifies the performance group number (PGN) associated with the work request. If specified, the performance group number value must be within the range of 1-999, represented as character data, left-justified and padded with blanks. The default is `NO_PERFORM`.
  - h. `PLAN=xxxxxxxx` : Specifies the name of an access plan for a set of associated SQL statements. The maximum plan name is 8 characters long. The default is `NO_PLAN`.
  - i. `PRCNAME=xxxxxxxxxxxxxxxxxxxx` : Specifies the name of a DB2 Stored Procedure associated with the work request. The maximum name that can be supplied is 18 characters long. The default is `NO_PRCNAME`.
  - j. `PRIORITY=nnnnnnnnnn` : Specifies the priority associated with the work request. The priority is specified as a decimal number. The maximum permitted value is 2147483647. The default is `NO_PRIORITY` (x80000000).
  - k. `SUBSYSPM=xxx...` : Specifies character data related to the work request. This information is passed to the work manager for use in classification. A maximum of 64 characters of information can be supplied. The default is `NO_SUBSYSPM`.
8. `TRXCLASS=xxxxxxxx` : Specifies a class name within the subsystem that the work manager recognizes. The maximum transaction class name that can be supplied is 8 characters long. The default is `NO_TRXCLASS`.
  9. `TRXNAME=xxxxxxxx` : Specifies a transaction name for the work request that the work manager recognizes. The maximum transaction name that can be supplied is 8 characters long. The default is `NO_TRXNAME`.

**NOTE:** A maximum of 254 characters of input parameters can be specified on the WLM UOW configuration parameter. See *OS/390 Planning: Workload Management* for information on how to define service classes and classification rules. This assumes you are going to operate the system in WLM goal mode. The priority for units of work should be less than VTAM and IMS. The discretionary goal may result in

very slow response times. Performance periods allow you to define a high number of service units for short transactions and a smaller number for long running ones.

10. Save the service configuration member.
11. Start the Server.

**NOTE:** If the Server is already running start the WLM service, then stop, and restart the Query Processor service for data source CACSAMP. This can be accomplished using the MTO Interface. For more information on dynamic configuration see [Appendix C, “MTO Command Reference.”](#)

12. Uncomment the SELECT statement(s) in SCACSAMP member CACSQL that reflects the datasource you are using.
  - a. Ensure that the user ID is authorized to access the table that is referenced in the query.
  - b. Save the changes after you have completed editing the member.
13. Edit SCACSAMP member CACCLNT to set the SQLIN parameter to reference the CACSQL member you edited in [step 12](#).
14. Save the member.
15. Start RMF data gathering if it is not already active.
16. Enter START RMF from the system console. See RMF USER'S GUIDE for information on using RMF. Start data gathering with Monitor III ISPF panel from TSO.
17. Submit CACCLNT and review the output.
18. The query should function normally and return the expected result set.
19. A SNAPSHOT of how WLM is managing this workload can be obtained by running the RMF Monitor II workload activity report during execution while the sample is running.
20. To run the RMF Monitor II interactive report:
  - a. Enter RMF from a TSO session.
  - b. From the menu select option 2 for RMF Monitor II. This report only displays the activity as it occurs so the query must be a long one to see its activity.
21. The RMF Monitor III processes data written to VSAM data sets by the data gathered by RMF. It can be accessed interactively from an ISPF panel. This should be started in Step 11 before the sample query is started. The SYSRTD report option reports response time distribution by service class and period. You should see the Query response time in this report by the service class you selected for eXadas. The SYSSUM Sysplex summary report shows goals versus actual for service class periods when the system is in goal mode.

## WLM Exit API Overview

The parameter list passed to the WLM Exit uses an AVA (Access Validation and Accounting) interface and is mapped by the macro CACSXPL, which is found in the SCACMAC library. The AXPLFUNC field indicates the function you are calling: initialization, management/reporting, or termination. The fields and their descriptions are shown in the table that follows.

**Table 43: AXPLFUNC Fields and Descriptions**

Field	Description
&DSECT=YES	Used to control whether a DSECT definition is generated or whether the fields are generated in the exit's local storage area.
AXPL DSECT	DSECT definition that is generated when &DSECT=YES is specified.
AXPL DS 0H	Label that is generated when &DESCT=YES is not specified.
AXPLID DC CL4'AXPL'	Identifier that should be checked to determine whether the internal storage area of the AXPL parameter list has been corrupted.
AXPLUSER DS 3A	Word(s) available for CPU Resource Governor Exit use.
AXPLETYP DS F	Event type.
AXPLESUB DS F	Event sub-type.
AXPLESEQ DS F	Event sequence.
AXPLEBEF EQU 1	Before event execution.
AXPLEAFT EQU 2	After event execution.
AXPLSQID DS CL8	User ID left-justified, blank-filled.
AXPLPENV DS A	Pointer to event specific information.
AXPLTEXT DS A	Pointer to a text buffer (this is usually an SQL statement).
AXPLXTL DS F	The decimal length of the text buffer.
AXPLSQLC DS F	The SQLCODE from the SQLCA after processing for the SQL event is completed.
AXPLFUNC DS H	Function identifier flag.
AXFNINIT EQU 0	Initialization.
AXFNVALI EQU 4	Management/Reporting.
AXFNTERM EQU 8	Termination.

## Initialization

This function is called by the WLM initialization service during server address space initialization. The AXPLTEXT parameter points to any additional parameters that were specified in the task parameter field on the WLM initialization services SERVICE INFO ENTRY after the exit name. The AXPLTXL parameter identifies the length of the input parameters. If no parameters are supplied the AXPLTEXT and AXPLTXL fields are zeros. For more details on the WLM initialization service SERVICE INFO ENTRY parameter see [Appendix A, “Configuration Parameters.”](#)

**NOTE:** The WLM Exit may be called multiple times to perform initialization and termination processing, if the WLM initialization service is stopped and then restarted using the MTO Interface.

When called at this point, the exit should perform initialization processing and allocate any storage or other resources that are required for subsequent processing. At this point, the exit should also register itself with WLM and receive a WLM token that needs to be passed on subsequent calls. A pointer to the anchor for these resources can be placed in the AXPLUSER field of the parameter list. This pointer is preserved and passed to the exit on subsequent invocations. Upon initialization, the contents of the other fields in the parameter list are indeterminate.

**NOTE:** The AXPLUSER field consists of three fullwords that the exit can store anchor blocks in. The intent is to allow the exit to store an anchor for address space level storage acquired during initialization processing. The second fullword should be used to store an anchor block for storage that was acquired during Query Processor service initialization. This storage is “TCB-level” storage and should be allocated for each instance of a Query Processor service. The third fullword is available if the exit needs to allocate additional storage to manage an individual user.

When the exit is called to perform TCB initialization the AXPL storage area passed for initialization processing is cloned and the new copy is passed to the exit. If the exit stores an anchor block in one of the AXPLUSER fullwords, that address is “local” to the TCB being serviced. Likewise, when the exit is called to service a connection request, a copy of the “TCB-level” AXPL storage area is cloned and the new copy is passed to the exit. This copy is passed on subsequent calls to the exit to service individual SQL statements issued by the client application.

If the exit returns a non-zero return code, the WLM initialization service is terminated and a message written to the log. No additional call will be made to the WLM Exit.



## Management/Reporting

This exit function is called at predetermined AXPLETYP, AXPLESUB and AXPLESEQ processing points. The overview that follows lists the different processing points where this exit can be called and what type of processing it can perform.

### TCB Initialization/Termination

The WLM Exit is called with an AXPLETYP value of 1 (AVAETCB) during Query Processor initialization and termination for each Query Processor TCB.

The AXPLESUB field identifies whether the exit is being called for:

- (1) initialization processing (TCBINIT) or
- (2) termination (TCBTERM).

When called at these points, the AXPLTEXT field contains a pointer to the WLM UOW configuration parameter value. The AXPLTXTL field contains the length of the configuration parameter input. The AXPLPENV field contains a point to the data source name of the Query Processor being activated. The AXPLSQID is blank.

The exit should perform any TCB-level initialization/termination processing that is required. For example, on TCB initialization the eXadas WLM Exit performs an IWMCLSFY using the WLM UOW parameters supplied (if any) followed by an IWMECREA to create an enclave TCB environment. On the IWMECREA macro the FUNCTION\_NAME parameter is set to the first eight characters of the data source name. At TCB termination the eXadas exit issues an IWMEDELE to delete the enclave.

If the exit allocates a TCB-level control block and stores it in the AXPLUSER area on initialization processing, then this storage must be freed when the exit is called to perform TCB termination processing.

On initialization processing, if the exit issues a non-zero return code then service initialization is halted and the service is not started. On termination processing, the return code is ignored and normal termination process is continued. In either case the return code value is logged.

**NOTE:** The TCB initialization call is actually deferred until the first user connects to a Query Processor task. This allows the eXadas exit to perform the IWMCLSFY and IWMECREA calls and not violate the WLM recommendation that these two calls should be issued in rapid succession followed by an IWMEJOIN call to associate a unit-of-work with the enclave. Since these calls are not issued when a Query Processor task is physically started (only when the first user connects) the user will (generally) immediately issue an SQL request which causes an IWMEJOIN call to be issued. SQL statement processing is discussed in more detail [“SQL Statement Processing,” on page 250](#).

The exit may defer creating an enclave environment until a user connect call is issued (discussed next.) However, if the Query Processor is executing in multi-

user mode (Max. Users greater than 1) the exit has to manage context switches between users when the client application issues SQL statements. SQL statement processing is discussed in more detail later.

## User Connect/Disconnect

The WLM Exit is called with an AXPLETYP value of 2 (AVAEUSR) when a user connects/disconnects from a Query Processor service. The AXPLESUB field identifies whether the exit is being called for connect processing (CONNECT) or disconnect (DISC). When called at these points, the AXPLTEXT field contains a pointer to the WLM UOW configuration parameter value. The AXPLTXTL field contains the length of the configuration parameter input. The AXPLPENV field is zero. The AXPLSQID contains the user ID of the user connecting/disconnecting from the service.

The exit can perform any user-level processing desired. The eXadas exit does not perform any processing for these call points.

If the exit issues a non-zero return code on initialization processing, the user is disconnected from the service and the return code is returned to the client application. On disconnect, the return code is ignored and the all user related resources are freed, however, the return code is returned to the client application. In either situation the WLM generated return code is logged.

## SQL Statement Processing

The WLM Exit is called with an AXPLETYP value of 3 (AVAESQL) when a client application issues an SQL statement. The different kinds of SQL statements that an application can issue are identified in [Table 41, “SQL Statement Types,” on page 233](#). The table identifies the contents of AXPLESUB values and the contents of the AXPLTEXT and AXPLTXTL fields. The AXPLSQID field contains the user ID of the client issuing the SQL statement.

The AXPLESEQ field identifies whether the exit was called before (AXPLEBEF) or after (AXPLEAFT) the SQL statement has been processed by the Query Processor. The user has control of the Query Processor TCB (thread) for the duration of the SQL request. If running in multi-user mode another user can be serviced on the next SQL statement received by the Query Processor.

The exit should perform any actions required to manage the unit-of-work that the SQL statement represents. For example, the eXadas WLM Exit treats each SQL statement as a single unit-of-work and joins the enclave before the SQL statement is processed and then leaves the enclave once the statement has completed processing.

The duration of an SQL statement varies based on the type of SQL statement being issued and other configuration parameter values. The configuration parameter that has the most impact is PDQ. When PDQ is not active or when a query cannot be processed in PDQ mode, then the OPEN SQL statement will execute the longest since the entire result set is staged for fetch processing. In these instances any describe, fetch, and the close cursor requests will execute very quickly. When running in PDQ mode the work is more evenly distributed between

the open and the fetches since the result set is incrementally built based on the number of rows that can fit into the result set buffer.

If the exit issues a non-zero return code, processing for the query is halted and the return code is returned to the client application. The application can still issue another SQL request. There are situations where the exit generated return code will not be reported to the client. This occurs when the exit is called to performed “after” processing but an error code has been reported by another exit or generated by eXadas. In these cases the “original” error code takes precedence. In either case the WLM generated return code is logged.

## Termination

This exit function is called during WLM initialization service termination processing before the Server address space is shut down. At this time, the exit can perform any termination processing necessary and must free any resources it has allocated. If the exit allocated an address space level control block and stored it in the AXPLUSER area, then that storage must be freed.

**NOTE:** The WLM Exit may be called multiple times to perform initialization and termination processing, if the WLM initialization service is stopped and then restarted using the MTO Interface.

# DB2 Thread Management Exit

The DB2 Thread Management Exit runs under the DB2 Call Attachment Facility (CAF) service in the Server. The CAF service runs as an OS/390 subtask under the server, whose sole responsibility is to create and manage CAF connections to DB2. One instance of the subtask is required for each concurrent DB2 user. With the Thread Management Exit you can:

- validate eXadas clients prior to establishing connections to DB2 using CAF and
- control the duration of CAF connections to DB2.

By default, connections to DB2 itself using CAF are created when an eXadas client connects to DB2 and remains active until the Server is shut down. While this maximizes re-usability of the DB2 connections, the DB2 primary authorization ID for all connections is based on the Server’s started task or job name. In many cases, this level of security checking will not be adequate for your particular installation.

The eXadas DB2 Thread Management/Security Exit modifies the default behavior of connecting to and disconnecting from DB2. In addition, this exit performs SAF

calls to validate the user ID of the eXadas client and establishes the correct primary authorization ID for the client in DB2.

**NOTE:** A minor modification to the DB2-supplied authorization exit DSN3SATH is required in order to establish the correct primary authorization ID in DB2.

**WARNING:** The eXadas exit issues RACROUTE calls and must be run from an APF-authorized library. Additionally, when issuing the RACROUTE calls, the exit enters key zero supervisory state. The exit reverts back to problem program state immediately after each RACROUTE call returns.

Once activated, the DB2 Thread Management/Security Exit is invoked to perform the following functions:

- Initialization is called once at initial subtask start up.
- Client Connection is called each time a new eXadas client has acquired the CAF subtask.
- Another Connection to DB2 is called after each attempt is made to connect to DB2.
- Client Disconnection is called each time a client has released its connection to the CAF subtask.
- Termination is called once at subtask termination.

The eXadas exit CACSX07 performs the following functions at each of these invocation points:

- Initialization requires no processing. The exit returns a successful return code.
- Client Connection validates that the user ID and password for the incoming client are valid using a SAF call and establishes an ACEE control block for the TCB so the DB2 identified authorization exit can establish the correct primary authorization ID prior to requesting a connection to DB2.
- After Connection to DB2, the exit deletes the ACEE established when the client connection request was issued.
- Termination requires no processing. The exit returns a successful return code.

## Activating the DB2 Thread Management Exit

The following instructions assume that you have successfully installed eXadas, performed initial configuration, and verified the installation and configuration using the verification procedures as described in [Chapter 6, “Bringing DB2 On Line,”](#) in the *eXadas OS/390 Getting Started Guide*.

**To configure the eXadas DB2 Thread Management Exit and verify that it is working:**

1. Ensure that the DB2 Thread Management Exit load module CACSX07 is in an APF-authorized library (SCACLOAD).
2. Ensure the Server JCL references the APF-authorized library in the STEPLIB DD statement where the DB2 Thread Management Exit is located (SCACLOAD). Also ensure that any other data sets referenced in the STEPLIB DD statement are also APF-authorized.
3. Edit the Server master configuration member (SCACCONF member CACDSCF). Find the SERVICE INFO ENTRY for the DB2 CAF service and add the value CACSX07 to the PLAN name located in the task data field of the entry. The following examples show before and after definitions of the DB2 CAF SERVICE INFO ENTRY.

```
SERVICE INFO ENTRY = CACCAF DSN 2 1 5 1 4 5M 5M CACPLAN
SERVICE INFO ENTRY = CACCAF DSN 2 1 5 1 4 5M 5M
CACPLAN , CACSX07
```

4. Save the Master Configuration member.

**NOTE:** Steps 5 through 7 must be performed by a DB2 Systems Programmer, as these steps update a DB2 system and could disable users of DB2. See the *IBM DB2 Administration Guide* for information about updating DSN3SATH.

5. Update the DB2-supplied sample identify authorization exit DSN3SATH and insert the assembler logic, found in the eXadas SCACSAMP member CACSXDSN, immediately before the label SATH019.
6. Reassemble the identity authorization exit DSN3SATH.

```
//*DSN3ADD PROVIDE VALID JOB CARD
//*
//DSN3ADD PROC CAC='CAC', CAC HIGH-LEVEL QUAL
// DB2='DB2' DB2 HIGH-LEVEL QUAL
//ASSEMBLE EXEC PGM=ASMA90, PARM='LIST,NODECK,RENT'
//SYSLIB DD DISP=SHR,DSN=&DB2..MACLIB
// DD DISP=SHR,DSN=&DB2..ADSNMACS
// DD DISP=SHR,DSN=&DB2..AMODGEN
// DD DISP=SHR,DSN=&CAC..SCACMAC
//SYSLIN DD DISP=SHR,DSN=&&TEMP
//SYSUT1 DD
DSN=&&SYSUT1,UNIT=VIO,SPACE=(1700,(2000),,,ROUND)
//SYSPRINT DD SYSOUT=*
//SYSIN DD DISP=SHR,DSN=&DB2..ASM(DSN3SATH)
//*
//LINK EXEC PGM=IEWL,COND=(4,LT),
//
PARM='LIST,OL,RENT,REUS,AMODE=31,RMODE=ANY'
//SYSLIN DD DISP=SHR,DSN=&&TEMP
// DD *
NAME DSN@SATH(R)
//SYSLMOD DD DISP=SHR,DSN=&DB2..SDSNEXIT(DSN@SATH)
```

```
//SYSUT1 DD
UNIT=SYSDA,SPACE=(1024,(120,120),,ROUNT),DCB=BUFNO=1
//SYSPRINT DD SYSOUT=*
//
```

7. If your DB2 SDSNEXIT library is in the OS/390 linklist, refresh the linklist with the OS/390 **'F LLA,REFRESH'** command.
8. Start the server. If the server is already running, restart it.
9. Verify the exit is working correctly by running an eXadas client that issues DB2 queries. While the client is active, you can view the client's connection to DB2 by issuing the DB2 command `'-DIS THD (*)'` from an OS/390 console or SDSF. The OS/390 LOG output from the command should display the Server as the ID of the thread owner and the client-supplied user ID as the primary authorization ID.

## Developing Your Own DB2 Thread Management Exit

If your installation has special processing requirements that can be addressed by a DB2 Thread Management Exit, you can update the eXadas exit or create a custom exit of your own. This section describes the DB2 Thread management parameter list and processing options for developing your own customer DB2 Thread Management Exit.

The DB2 Thread Management Exit is called to perform the following functions:

- **Initialization:** Each time a new DB2 CAF service is started from the eXadas Region Controller, an initialization call is made to do any one-time initialization processing for the started task. If your exit must allocate any storage for use throughout the life of the task, it can allocate it in this call and place it in the user field DB2TUSRW supplied in the exit parameter structure.
- **Client Connection:** Called each time a new eXadas client has acquired the CAF subtask. Generally, this call is made immediately before connecting to DB2 on behalf of the client. The one exception is when the exit is set to leave connections to DB2 active even when eXadas clients disconnect.
- **After Connection to DB2:** Called after each attempt is made to connect to DB2. This call is made regardless of whether or not the connect was successful. The parameters field DB2STAT indicates whether or not the connection attempt was successful.
- **Client Disconnection:** Called each time a client has released its connection to the CAF subtask.
- **Termination:** Called once at subtask termination. If you allocated any exit memory at initialization, this is when it needs to be freed.

The parameter passed to the Thread Management Exit is shown in the example that follows.

The parameter descriptions follow:

- **DB2TUSRW:** Initialized to binary zeros prior to the initialization call and left unchanged after that point in processing. Use of this field is determined by the user exit.
- **DB2TUSRP:** Points to a NULL terminated user parameter as defined on the SERVICE INFO ENTRY for the DB2 CAF task. This parameter includes any text included after the exit name itself in the task data field. For example, to pass the string USERPARM to the exit from the SERVICE INFO ENTRY, the task data for the exit in the Master Configuration would be:

```
CACPLAN, CACSX07 USERPARM
```

- **DB2TSSN:** Points to the four character subsystem name as defined in the Service Name field of the SERVICE INFO ENTRY for the task. In most cases, this field is for informational purposes only. However, the exit can change this field on client connection calls to designate a new subsystem to connect to, if necessary. If this field is updated, it will remain updated until the thread disconnects from DB2. At that point, it will be changed back to its original SERVICE INFO ENTRY value.
- **DB2TPLAN:** Points to the 8-character DB2 PLAN name as defined in the Task Data field of the SERVICE INFO ENTRY for the task. In most cases, this field is for informational purposes only. However, the exit can change this field on client connection calls to designate a new DB2 PLAN to open a DB2 connection. If this field is updated, it remains updated until the thread disconnects from DB2. At that point, it is changed back to its original SERVICE INFO ENTRY value.
- **DB2TUID:** Points to the user ID provided by the eXadas client when it connected to the Server. This field is binary zeros on initialization and termination calls as no user is available when these calls are issued. This field is for reference purposes only and must not be changed by the exit.
- **DB2UPWD:** The DB2UPWD field points to the user password provided by the eXadas client when it connected to the Server. This field is binary zeros on initialization and termination calls as no user is available when these calls are issued. This field is used for reference purposes only and must not be changed by the exit.
- **DB2TSTAT:** The DB2TSTAT field identifies whether or not a current CAF connection exists to DB2. This field is used for reference purposes only.
- **DB2TFUNC:** The DB2TFUNC field identifies the function of the call as described previously. Defined values for this field are:
  - **DB2TINIT:** Initialization function
  - **DB2TCCON:** Client Connection function
  - **DB2TDB2C:** DB2 Post Connection/Plan Open function
  - **DB2TCDIS:** Client Disconnect function
  - **DB2TTERM:** Termination function

- **DB2TRFNC:** The DB2TRFNC field can be used by the exit to:
  - explicitly request connection or reconnection to the DB2 subsystem,
  - explicitly request disconnection from the DB2 subsystem, and
  - notify eXadas that a user ID or password validation error has occurred.

The field is used to alter the default DB2 subsystem connection/disconnection behavior in the DB2 CAF service. The default behavior is to connect to DB2 when the first user requests DB2 access and disconnect from DB2 at Server shutdown. The exit should set this field on each call to one of the following values:

- **DB2TRDFL:** Do default connection processing.
- **DB2TRCON:** Connect to DB2. If a connection already exists, terminate that connection and create a new connection using the subsystem and plan name in the fields DB2TSSN and DB2TPLAN. This value is valid for Initialization and Client Connection functions.
- **DB2TRDIS:** Disconnect from DB2 if a connection exists. This value is valid for the Client Disconnect function.
- **DB2TRUER:** User or password information is invalid. This value can be returned on the Client Connection function to return an Access Denied error to the requesting client.

The return code (register 15 value) for successful completion of the user exit should always be set to 0. Any other value causes an error message to be returned to the requesting client.

To activate your exit in the Server, follow the directions in [“Activating the DB2 Thread Management Exit,” on page 252](#), replacing the system default exit name CACSX07 in Step 3 with the name of your exit. You can skip steps 5 and 6 if your exit is not creating a TCB level ACEE for the DB2 primary authorization ID setting.

## Record Processing Exit

The Record Processing Exit, available in the VSAM and Sequential Query Processors, is used to modify the characteristics of the record to make it easier for eXadas to process. A sample exit, CACSX08, is supplied in the sample library.

The exit must be re-entrant, save registers on entry and restore them on exit, and be `AMODE(31)`, `RMODE(ANY)`. The exit will be called for initialization, processing, and termination and executes as part of the eXadas product. Any errors in the exit routine may affect the operation of the product as a whole.



The exit may be written in any language, but since the exit will be called on every record, consideration should be given as to the performance of the chosen language.

Register contents at entry to the exit routine:

- R1 contains a pointer to a parameter list.
- R13 points to a register save area in standard format.
- R14 contains the return address.
- R15 contains the entry point of the routine.

All other registers must be restored upon return from exit.

There are 7 address pointers to parameters passed to the exit. They are defined in [Table 44, “Record Exit Input Parameters.”](#)

**Table 44: Record Exit Input Parameters**

Field	Description	Length
Function	INIT, PROCESS, UPDATE, or TERM. These values are padded on the right with spaces.	7 bytes
Table Description	Name of the table being processed.	18 bytes
Input Record	Record that was read.	
Input Record Length	Length of the input record.	Binary fullword
Output Record	Record that eXadas is to process.	
Output Record Length	Length of the output area.	Binary fullword
User Word	A word passed to the exit that can be used to anchor additional information.	Binary fullword
Return Code	Completion code of the exit call. Return code set to zero indicates processing completed normally. A return code less than zero will terminate the query. A return code greater than zero will skip processing of the current record and read the next record.	Binary fullword

## Initialization

Initialization is called during the open of the table. During this function, the exit may acquire resources needed for later processing.

## Process

Process is called after the eXadas Query Processor has read a record. It passes the record to the exit. The exit can modify the record and rebuild it in the output record area. The output record area is the record that eXadas will process. Meta Data Grammar should be based on the output record.

## Termination

Termination is called when the table is closed. The exit may release any acquired resources and perform any other clean-up tasks.

## Update

Update is called when the input record is the updated record from an SQL INSERT or UPDATE request. The output record is the location that the record needs to be placed when writing back to the file.

If an SQL UPDATE was performed, the input record will be the output record from a previous PROCESS call, with changes made based on the SQL UPDATE request.

## Verification

In order to certify that the Record Exit installation is operational, CrossAccess provides a procedure that uses the sample databases from the Server. The following section provides the steps required for the installation verification process.

The SCACSAMP library contains a sample Record Exit source (CACSX08). It is an example of how a VSAM file record can be changed before returning to the user. The program accesses the sample VSAM employee data that was delivered with the original eXadas implementation. This sample was developed specifically for a VSAM data file. If the eXadas environment supports other non-relational databases, such as IMS, then the source code should be reviewed and modified to comply.

A sample load module CACSX08 is provided in SCACLOAD, therefore there is no need to assemble and link-edit in order to make the sample work.

**NOTE:** The following instructions assume that you have successfully installed the Server, performed initial configuration, and verified the installation and configuration using the eXadas sample application and data.

1. Add a DD statement (the DD name must be CACIVP) to the Server job and Meta Data Utility's CACMETAU job in the SCACSAMP library to point to the eXadas employee data VSAM file.
2. Regenerate the USE Grammar, in DataMapper, for the sample VSAM employee data to indicate the record exit name (CACSX08) and the maximum length (100) at the Create VSAMTable dialog box.
3. Generate and FTP the grammar to the host and include it as input to the Meta Data Utility CACMETAU in SCACSAMP.
4. Submit the job.

See the Bringing VSAM on line chapter of the *eXadas OS/390 Getting Started Guide* for additional information.

The following sample Record Exit statement is added into the USE Grammar:

```
RECORD EXIT CACSX08 MAXLENGTH 100
```

5. Uncomment the Language Environment Initialization Service in the Master configuration member (CACDSCF) if necessary. See [“Performance Considerations,” on page 259](#), for more information. In this example, the sample exit CACSX08 is written in Assembler, so there is no need to uncomment the SERVICE INFO ENTRY parameter.
6. Start the Server.
7. Review CACSQL and comment out other queries if you use this member as the input for the client job in [step 8](#). Use the VSAM SQL statement to verify the Record Exit processing.
8. Submit the client job CACCLNT and verify the return results. The record is filtered out if the column ENAME starts with M. Otherwise, the 16th byte is changed to a tilde ( ~ ) if the record does not start with M.

**NOTE:** To recover the original data in the catalog table after completing this sample exit verification, remove the Record Exit statement in the USE Grammar and rerun the CACMETAU job.

## Performance Considerations

If you will be writing the Record Processing Exit in a language other than Assembler, use the Language Environment Initialization Service to reduce the overhead of initializing and terminating the environment on each call. This service also allows program storage to remain constant between calls and files not to be closed on exit. For exits written using IBM's Language Environment, uncomment the following SERVICE INFO ENTRY in the master configuration:

```
SERVICE INFO ENTRY = CACLE LANGENV 2 1 1 100 4 5M 5M CEEPIPI
```

For exits written using COBOL II, add the following SERVICE INFO ENTRY in the master configuration:

```
SERVICE INFO ENTRY = CACLE LANGENV 2 1 1 100 4 5M 5M IGZERRE
```

The following conditions apply if you will be using the Language Environment Initialization Service with COBOL II:

- If program variables are changed, they will not be refreshed to their original values until all use of the exit in the address space has completed.
- Only one user can be active in the same exit at one time.

# 16

## Stored Procedures

### Introduction to Stored Procedures

This chapter describes how to define, write, and invoke stored procedures. The following topics are discussed:

- [“Stored Procedure Overview,”](#) on page 262,
- [“Defining Stored Procedures,”](#) on page 274,
- [“Writing Stored Procedures,”](#) on page 284,
- [“Invoking Stored Procedures,”](#) on page 292,
- [“CICS Interface Description,”](#) on page 297,
- [“CA-DATACOM/DB Interface Description,”](#) on page 310,
- [“IMS DRA Interface Description,”](#) on page 320,
- [“Invoking Existing IMS Transactions,”](#) on page 325, and
- [“Support Routine Descriptions,”](#) on page 331.

# Stored Procedure Overview

A stored procedure is an application program that is designed to perform work that cannot be performed using normal SQL DELETE, INSERT, SELECT, and UPDATE operations, or for updates to databases that currently do not support update operations. These programs are written in C, COBOL, or Assembler language, with COBOL the usual language of choice.

This section provides an overview of the environment(s) in which a stored procedure application program executes, and the eXadas-supplied interfaces and support routines that the stored procedure application program can call to assist during execution.

The following topics are covered:

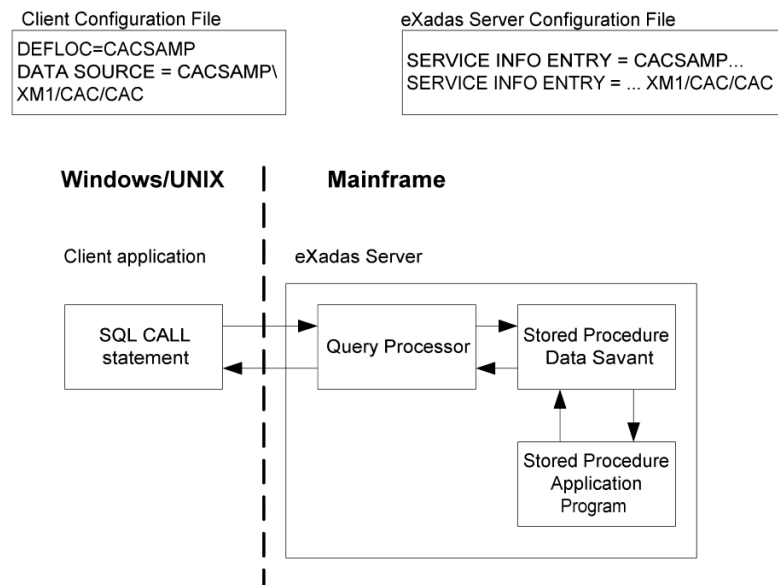
- [“General Concepts,” on page 262,](#)
- [“Residency and Language Environment,” on page 264,](#)
- [“Interfacing with CICS,” on page 267,](#)
- [“Interfacing With IMS,” on page 269,](#)
- [“Interfacing with CA-DATACOM/DB,” on page 271,](#)
- [“Support Routines,” on page 272,](#)
- [“Samples,” on page 273.](#)

## General Concepts

Stored procedures are a form of remote procedure call that operate in a client-server environment. That is, a client application executing on Windows or UNIX invokes the stored procedure application by issuing an SQL CALL statement. This results in the application program associated with the stored procedure referenced on the CALL statement to be executed in the server’s address space. [Figure 35: “Stored Procedure Execution Flow,” on page 263,](#) depicts how stored procedures work.

When the Query Processor receives the CALL statement, it verifies that:

- the stored procedure identified in the CALL statement exists in the Meta Data Catalog,
- the correct number of parameters have been supplied,
- all required parameters have been supplied,
- for each parameter (for which data was supplied), the data type supplied by the client application is compatible with its defined data type.

**Figure 35: Stored Procedure Execution Flow**

Assuming that no problems were found, the Query Processor forwards the CALL request to the Stored Procedure Data Savant for processing. When the stored procedure is defined, you specify the name of an OS/390 load module that the Stored Procedure Data Savant is to execute when a CALL statement for that stored procedure is received.

The Stored Procedure Data Savant determines whether there is an executable copy of the stored procedure application program already loaded in memory. If there isn't, it loads the requested program in the server address space. The Stored Procedure Data Savant then branch-enters the stored procedure application program, passing a standard parameter list that contains the data values that were sent from the client application. The stored procedure application program can perform any processing desired, and once completed, returns control to the Stored Procedure Data Savant.

If the stored procedure application program updates any databases, the application should explicitly issue a commit to apply any changes made by the application before control is returned to the Stored Procedure Data Savant. Alternately, if the stored procedure application program detects an error during processing, it should issue a rollback to ensure that any changes that have been made are backed out. If the stored procedure application program accesses or updates a file (for example, a VSAM file) it must open the file upon entry and ensure that it is closed when control is returned to the Stored Procedure Data Savant.

When defining the parameters that are passed to the stored procedure application program, you identify the SQL data type for each parameter. You also define how the parameter is to be used.

A parameter can be used in one of the following ways:

- as an input parameter,
- as an output parameter,
- as both an input and an output parameter.

These parameters are passed to the stored procedure application program in a standard SQLDA format. The stored procedure application program extracts these parameters from the SQLDA. Based on the parameter values, it performs the processing that the stored procedure application is designed to perform.

While executing, the stored procedure application program can update the contents of output and input-output parameters that were passed to the application. After control has returned from the stored procedure application program, the Query Processor accesses the parameter list passed to the stored procedure application program, extracts the contents of any non-null output and input-output parameters. All of the original input parameters and the original/updated output and input-output parameters are then returned to the client application that issued the CALL statement.

In addition to passing the parameters supplied by the client application, the stored procedure application program is also passed an SQLCA structure that the stored procedure application program can update with an error SQLCODE that the Query Processor will return to the client application. Alternately, if the stored procedure application program returns a non-zero return code to the Stored Procedure Data Savant, the return code will be returned as the SQLCODE of the CALL statement to the client application. The SQLCODE from the SQLCA takes precedence over the stored procedure application program return code.

## Residency and Language Environment

As shown in [Figure 35: “Stored Procedure Execution Flow,” on page 263](#), the stored procedure application program executes in the server’s address space. Therefore, the stored procedure application program competes with the server and other stored procedure applications for resources in the address space.

By the time the stored procedure application program executes, the server has already allocated the memory it needs for the message pool. Any memory that the stored procedure application program allocates is therefore memory not being managed by the server. It is imperative that any memory allocated be freed before the stored procedure application program returns control to the Stored Procedure Data Savant.

The stored procedure application program’s load module also takes up memory that is not being managed by the server. Since it is likely that you may be executing multiple copies of the stored procedure application program simultaneously, these applications must be written as reentrant and should be link-edited as re-entrant, reusable, and refreshable (RENT,REUS,REFR).



Before executing the stored procedure application program, the Stored Procedure Data Savant checks to see whether a copy of the stored procedure application program needs to be loaded. This processing behavior is an option that you control when you define the Stored Procedure and is referred to as *residency*. It is specified using the STAY RESIDENT parameter. For details on how to define a Stored Procedure and the parameters that can be specified on a stored procedure definition see [“Defining Stored Procedures,” on page 274](#)

If STAY RESIDENT is set to NO, the stored procedure application program is considered *non-resident*, and each time a CALL statement is issued, the Stored Procedure Data Savant loads a copy of the stored procedure application program and when control is returned an un-load is issued. When STAY RESIDENT is set to YES, the stored procedure application program is referred to as *resident*.

For resident-stored procedure application programs, the Stored Procedure Data Savant maintains a list of currently-loaded stored procedure application programs. When a CALL is issued the Stored Procedure Data Savant checks the list of currently loaded applications and if the stored procedure application program is not on the list, it is loaded and added to the list before it is called.

Resident stored procedure application programs remain loaded until the Query Processor is terminated. Each active Query Processor instance maintains its own list of resident stored procedure application programs. If the stored procedure application program is re-entrant, there may be one copy of the stored procedure application program loaded, however, its use count can be greater than one (if multiple Query Processor instances are executing). In these situations the stored procedure application program will not be physically unloaded from memory until all Query Processor instances that have issued a load for the stored procedure application program have terminated processing.

Additionally, the Stored Procedure Data Savant will load multiple copies of the stored procedure application program if the stored procedure application program (for example, the load module name) is associated with multiple stored procedure definitions. Currently, there is no method to determine how many stored procedure applications programs are loaded within the server nor how many have been loaded by a particular Query Processor/Stored Procedure Data Savant.

While you are developing your stored procedure application program you want to define the stored procedure as non-resident. Once testing is completed change the stored procedure to be resident for performance reasons.

**NOTE:** While testing your stored procedure application program, if the stored procedure is defined as resident and you have run the stored procedure once and then made some modifications to the stored procedure application program and want to re-test it, you will have to shut down the Query Processor instance (that executed the stored procedure application program the first time) in order to get the updated copy of the stored procedure application program loaded. This type of situation happens frequently, so during initial development it is a good idea to display some kind of version identifier so that you can quickly ascertain which version of your stored procedure application program is actually being executed by the Stored Procedure Data Savant.

Generally, your stored procedure application program is developed using a high-level language, such as COBOL. All of the IBM-supplied high-level languages use the OS/390 Language Environment (LE). Language Environment provides common memory allocation, error reporting, and other services that can be used by any of the IBM-supplied high-level languages or Assembler Language programs.

The stored procedure application program is branch entered when it is called. If your stored procedure application program is written in a high-level language, then when the program is entered, the run-time environment for the language that the program is written in is initialized before your actual application code is executed. By default, this run-time initialization processing is performed each time a CALL statement is issued for that stored procedure.

However, if you have activated the LE Initialization Service within the server, this behavior changes. For details on activating the LE Initialization Service, see [Appendix A, “Configuration Parameters,”](#) or [Chapter 15, “System Exits.”](#)

When the LE Initialization Service is active, the LE Pre-initialization service routine (CEEPIPI) has been loaded into the servers address space. CEEPIPI is an IBM-supplied routine that provides services for environment initialization, application invocation and environment termination.

Additionally, when the LE Initialization Service is active, the Stored Procedure Data Savant no longer branch enters your stored procedure application program and instead uses CEEPIPI services to invoke your application program. Specifically, the stored procedure application program is executed as a *dependent subroutine*.

Some of the characteristics of a dependent subroutine are that it:

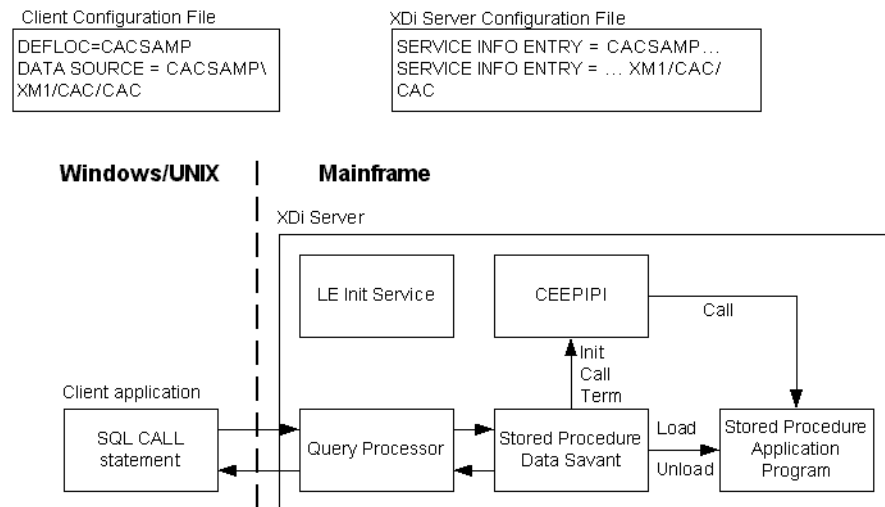
- creates and initializes a new Language Environment process and enclave to allow the execution of the stored procedure application program multiple times,
- sets the environment dormant so that exceptions are percolated out of it, and
- ensures that when the environment is dormant, it is immune to other Language Environment enclaves that are created or terminated.

When executing in an LE environment, after the Stored Procedure Data Savant loads the stored procedure application program, it calls CEEPIPI to initialize the environment, then calls CEEPIPI to execute the stored procedure application program as a subroutine. When the stored procedure application program is unloaded by the Stored Procedure Data Savant, prior to the unload, CEEPIPI is called to terminate the environment. [Figure 36: “LE Execution Environment,” on page 267](#) depicts the execution environment when the LE Initialization Service is active.

**WARNING:** If you activate the LE Initialization Service it must be defined in the Master Configuration Member before the Query Processor Service Info. entries. Failure to do so is likely to cause a S0C4 abend when the Server is shutdown. Services are unloaded in first-in-last-out order so that if any resident stored procedures have

been executed using LE, the CEEPIPI interface must still be loaded when the Stored Procedure Data Savant attempts to terminate the LE environment.

**Figure 36: LE Execution Environment**



Additionally, when you define the stored procedure you can specify custom run-time information using the RUN OPTIONS parameter. This allows you to tailor the execution environment for your stored procedure application program if it has special requirements. If you do not specify RUN OPTIONS information the default of ALL31(OFF) is used. For more information on LE initialization parameters see *OS/390 Language Environment for OS/390 & VM Programming Reference*.

If your stored procedure application program is written in a high-level language, activate the LE Initialization Service and run the stored procedure application program in resident mode. In this configuration the overhead of loading the stored procedure application program and establishing the run-time environment is incurred the first time the stored procedure application program is invoked. On subsequent invocations you will notice a dramatic performance improvement.

eXadas has extended the use of the RUN OPTIONS parameter for instances where you have activated the LE Initialization Service yet need to execute a stored procedure application program that is not LE-enabled. Specifying NO\_LE informs the Stored Procedure Data Savant that the stored procedure application program is not LE-enabled and the application is called in branch-entry mode.

## Interfacing with CICS

If you need to create a stored procedure application program that updates a VSAM file, you may find that CICS has exclusive control of the file and that it cannot be updated from the server address space. For these situations, and potentially others, eXadas provides facilities that allows you to invoke a CICS application program.

The CICS application program is passed a copy of the data from the client application that issued the CALL statement. Like a stored procedure application running within the server's address space, the CICS application program can update the values for output and input-output parameters for return to the client application.

Unlike stored procedure application programs running in the server's address space, the CICS application program does not have accessibility to an SQLCA structure for error reporting. Instead, it has addressability to an application return code that is returned to the stored procedure application program running in the server address space and will automatically be percolated to the client application. If a CICS abend is detected it will be percolated back to the stored procedure application program, and by default, to the client application.

Communications with CICS is performed using VTAM LU 6.2 with the eXadas VTAM Connection Handler. eXadas provides an API interface (CACSPBR) load module that can be called by a stored procedure application running in the Server address space that communicates with the VTAM Connection Handler to:

- establish a session with CICS,
- send data,
- receive data,
- perform address translation for the updated parameter list returned from the CICS application program, and
- end the session.

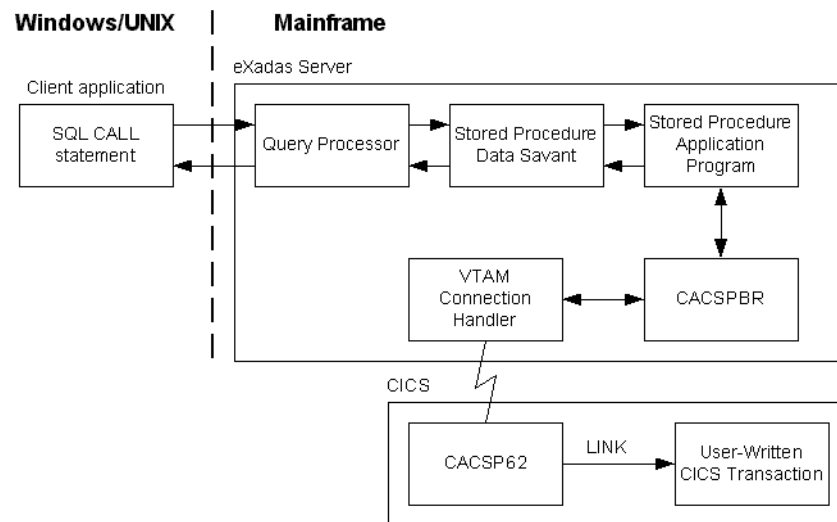
eXadas also supplies a CICS LU 6.2 application (CACSP62), which is the partner for CACSPBR. This application is responsible for:

- performing address translation for the parameter list being passed to the CICS transaction,
- invoking the specified CICS application program via a EXEC CICS LINK,
- percolating return code or CICS abend codes back to the server, and
- deallocate the session in the case of a CICS abend.

[Figure 37: "CICS Processing Flow," on page 269](#), shows the processing flow when the stored procedure application program interfaces with CICS.

The CACSPBR interface allows a stored procedure application program to send and receive multiple transmissions between itself and CICS. In normal situations only a single send and receive is required. For these situations, eXadas supplies an Assembler Language stored procedure application program (CACSPVTM) that sends the client parameter list to CICS and receives (a possibly) updated parameter list from CICS. For most situations, using CACSPVTM eliminates the need for you to develop your own stored procedure application program in order to invoke a CICS application program.

Figure 37: CICS Processing Flow



Additionally, eXadas has made extensions to the RUN OPTIONS parameter that allows you to specify the required CICS transaction scheduling information in order to invoke your user written CICS application program. See [“Specifying CICS Transaction Scheduling Information,” on page 282](#), for more information on how to specify CICS transaction scheduling information.

**NOTE:** If you want to write your own stored procedure application program that dynamically invokes CICS applications based upon client-supplied information and interfaces with CACSPBR do not specify CICS transaction scheduling information on the RUN OPTIONS parameter. If specified, RUN OPTIONS CICS transaction scheduling information overrides any transaction scheduling information passed to CACSPBR.

For examples of the VTAM and CICS resource definitions required to execute a CICS application program using a Stored Procedure, see [Appendix B, “Sample Stored Procedure VTAM and CICS Definitions.”](#)

## Interfacing With IMS

If your stored procedure application program needs to access/update IMS data you cannot do that using standard DL/I calls. The primary reason is that your stored procedure application program does not have addressability to a PSB and the list of PCBs contained in the PSB.

To get around this problem, eXadas supplies the CACTDRA interface load module. The CACTDRA interface allows you to schedule a PSB and returns a pointer to the list of PCBs in the PSB. You can then establish addressability to one or more of these PCBs and issue ISRT, GU, GHU, and REPL DL/I calls against the PCB. Before exiting your stored procedure application you call CACTDRA one last time to unschedule the PSB.

By default, when you un-schedule the PSB any changes your stored procedure application program made are committed to IMS. If your stored procedure application program does not want changes committed it should issue a ROLLBACK call on the I/O PCB before unscheduling the PSB.

**WARNING:** Do not issue a ROLLBACK call. If your stored procedure application program does, an IMS abend occurs and the Query Processor instance that received the CALL statement becomes unusable.

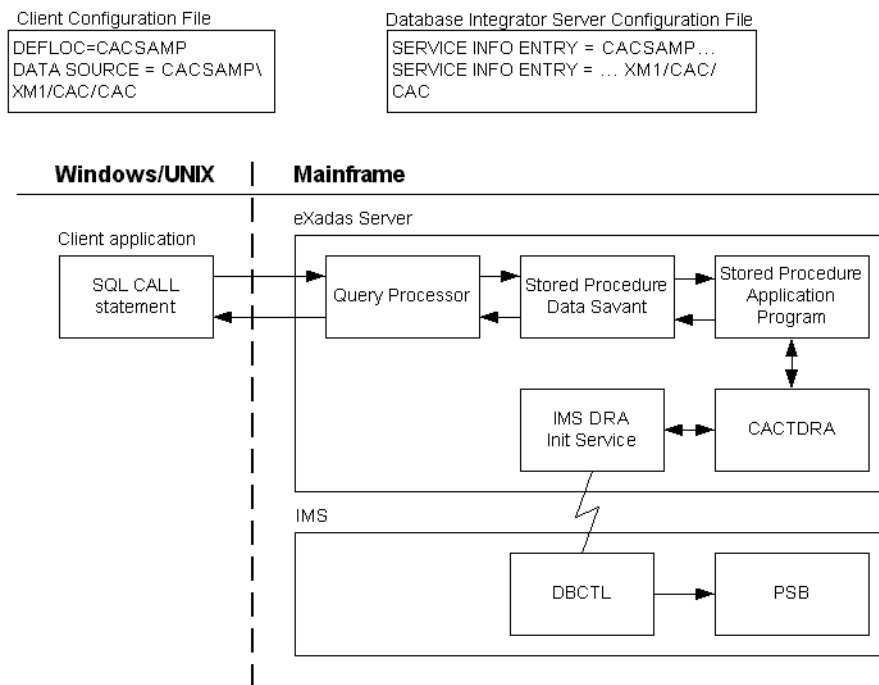
The CACTDRA interface uses the IMS DRA Initialization Service to request PSB scheduling, issuing your applications DL/I calls and finally unscheduling the PSB. For more information on how to set up an environment where DRA can be used and how to activate the IMS DRA Initialization Service, see [Chapter 3, “Server Setup for IMS Access.”](#)

Figure 38: “IMS DRA Processing Flow,” on page 270, shows the processing flow when your stored procedure application program needs to access IMS data.

To make using the CACTDRA easier to use, the DL/I call formats are identical to a normal CBLTDLI (or ASMTDLI) call syntax with the exception that the first parameter in the parameter list must be the address of the SQLDA parameter passed to your stored procedure application program. Additionally, CACTDRA uses two other DL/I function codes, they are:

- SCHD - Used to identify the name of a PSB to be scheduled.
- TERM - Used to un-schedule the PSB.

**Figure 38: IMS DRA Processing Flow**



## Interfacing with CA-DATACOM/DB

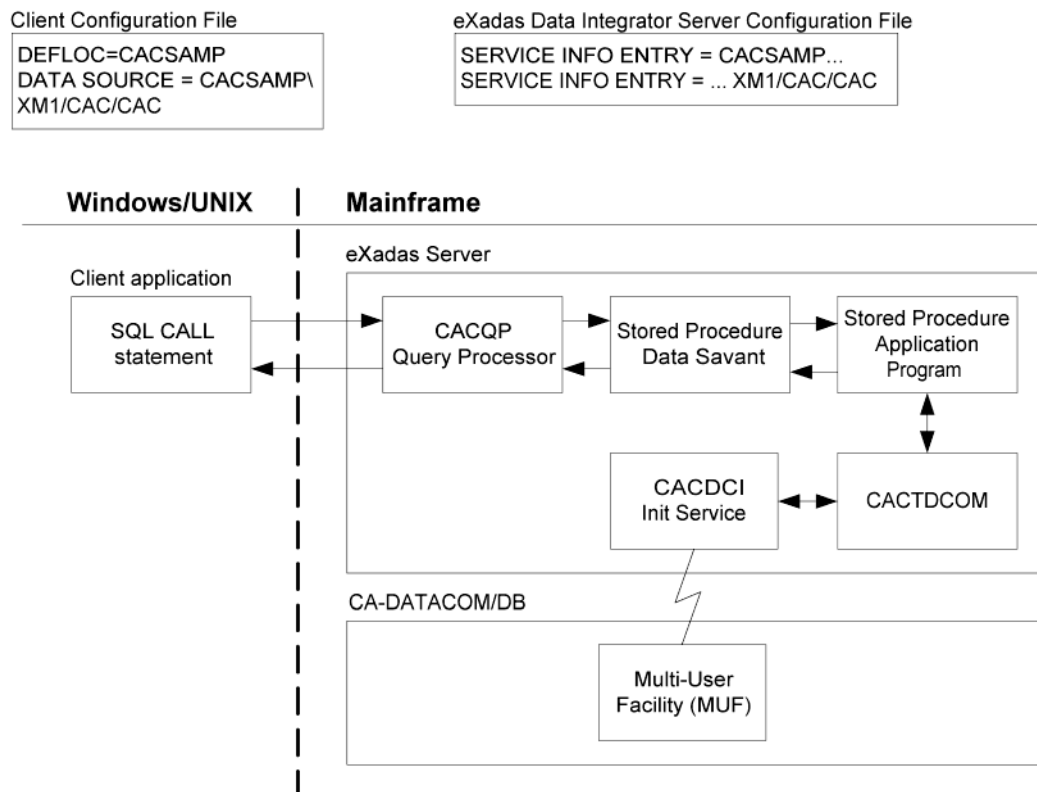
If your stored procedure application program needs to access or update CA-DATACOM/DB data, you can use the eXadas-supplied CACTDCOM interface load module. The CACTDCOM interface module allows you to open your URT from within the eXadas address space and issue CA-DATACOM/DB calls to the database. Before exiting your stored procedure application, you must call the CACTDCOM interface module one last time to close the URT.

By default, when you close the URT, any changes made by your stored procedure application program are committed to the database. If your stored procedure application program does not want changes committed, it should issue a ROLLBACK call prior to closing the URT.

The CACTDCOM interface load module communicates with the Datacom Initialization Service to connect with CA-DATACOM/DB for opening/closing your URT and issuing your application calls. For more information on how to activate the Datacom Initialization Service, see [Chapter 5, “Server Setup for CA-DATACOM/DB.”](#)

[Figure 39: “CA-DATACOM/DB Processing Flow”](#) shows the processing flow when your stored procedure application program needs to access CA-DATACOM/DB data.

**Figure 39: CA-DATACOM/DB Processing Flow**



To make using the CACTDCOM interface module easier, the CA-DATACOM/DB call formats are identical to the call syntax used by native CA-DATACOM/DB, with one exception: the first parameter in the parameter list passed to the CACTDCOM interface module must be the address of the SQLDA parameter passed to your stored procedure application program by the Stored Procedure Data Savant. Calls to CA-DATACOM/DB are performed as if the program were written for direct access to CA-DATACOM/DB.

For example:

- To open a URT you must provide:
  - a properly formatted User Information Block (UIB), and
  - a Request Area containing the command OPEN.
- To begin reading records using Set At A Time commands, you must provide:
  - a properly formatted User Information Block (UIB),
  - a Request Area containing the command SELFR along with the table name and DBID if required,
  - a work area to receive the retrieved data,
  - an element list describing the data to be retrieved, and
  - a Request Qualification Area containing selection criteria and other parameters.
- To close a URT you must provide:
  - a properly formatted User Information Block (UIB), and
  - a Request Area containing the command CLOSE.

Success or failure of every command is returned in the Request Area Return Code and Internal Return Code as documented in the *CA-DATACOM/DB Programmer's Guide*.

## Support Routines

In addition providing interfaces to CICS, CA-DATACOM/DB, and IMS, eXadas also supplies three support routines that can be called by your stored procedure application program. [Table 45, "Support Routines,"](#) identifies the subroutine names and purpose of each of the support routines.

**Table 45: Support Routines**

Routine Name	Purpose
CACSPGRO	Copies the value of the RUN OPTIONS parameter into an application storage area.
CACSPGPW	Copies the value of the user password into an application storage area. The user password was captured when the client application connected to the server.



**Table 45: Support Routines**

Routine Name	Purpose
CACSPGUI	Copies the value of the user ID into an application storage area. The user ID was captured when the client application connected to the server.

Unlike the CICS, CA-DATACOM/DB, and IMS interfaces, which are distributed as load modules, these support routines are supplied in object-module form for direct inclusion in your stored procedure application program. These routines are written in Assembler Language and use standard OS linkage conventions. All routines are passed two parameters, the first of which must be the address of the SQLDA parameter passed to the stored procedure application program. The second parameter is an address where the value is copied.

## Samples

eXadas does not supply a sample process for stored procedures. However, it does supply various samples that are intended to assist you in development of your own stored procedure application programs. Most of these samples are supplied in COBOL. The following table identifies the different samples that are supplied. For each sample, the table identifies the library where the sample is contained, its member name, and a short description of what the sample is.

**Table 46: Stored Procedure Samples**

Library Name	Member Name	Description
SCACSAMP	CACSPCCC	Sample compile and link deck for stored procedure applications calling CACSPBR.
SCACSAMP	CACSPCCL	Sample compile and link deck for sample local stored procedure.
SCACSAMP	CACSPCCR	Sample compile and link deck for the sample CICS stored procedure.
SCACSAMP	CACSPCOM	Generic stored procedure application program to invoke a CICS application program.
SCACSAMP	CACSPCP	Sample stored procedure definitions containing a parameter definition for each supported data type.
SCACSAMP	CACSPCPY	COBOL definitions for the argument data passed to the stored procedure. This should be included in your stored procedure application. Sample local stored procedure application using the IMS DRA interface.

**Table 46: Stored Procedure Samples**

Library Name	Member Name	Description
SCACSAMP	CACSPDC1	Sample local stored procedure application program using the interface module CACTDCOM to access CA-DATACOM/DB. Statically link module CACTDCOM with CACSPDC1 as described in <a href="#">“Compiling and Linking Applications That Use CACTDCOM,”</a> on page 315.
SCACSAMP	CACSPCLL	Sample local stored procedure application.
SCACSAMP	CACSPREM	Sample remote stored procedure application executing in CICS.
SCACSAMP	CACSPDFH	COBOL version of the CICS communications area passed to a CICS application invoked by CACSP62.
SCACSAMP	CACSPGRO	Get RUN OPTIONS support routine object module.
SCACSAMP	CACSPGPW	Get Password support routine object module.
SCACSAMP	CACSPGUI	Get User ID support routine object module.
SCACSAMP	CACSPSCA	COBOL SQLCA structure for inclusion in your stored procedure application program.
SCACSAMP	CACSPSDA	COBOL SQLDA structure for inclusion in your stored procedure application program and/or CICS application program.
SCACSAMP	CACSPVTM	COBOL APPC function and data structures for interfacing with CACSPBR.

## Defining Stored Procedures

Stored procedures definitions are stored in the eXadas Meta Data Catalogs and defined using the Meta Data Utility. General Stored Procedure information is stored in the SYSIBM.SYSROUTINES system table. The parameters that are supplied by the client application and passed to the stored procedure application program are stored in the SYSIBM.SYSPARMS system table.

To define a stored procedure you use a CREATE PROCEDURE statement. To delete a stored procedure definition from the Meta Data Catalogs you use a DROP PROCEDURE statement.

You can use the RUN OPTIONS parameter on the CREATE PROCEDURE statement to supply custom LE run-time options during environment

initialization, when the stored procedure application program is executed in an LE environment. In addition, eXadas has extended the use of the RUN OPTIONS parameter to allow deactivation of the LE environment for a particular Stored Procedure. The RUN OPTIONS parameter also allows you to specify CICS transaction scheduling information for CICS interfacing or CA-DATACOM/DB resource information for CA-DATACOM/DB interfacing.

The following topics are discussed:

- [“CREATE PROCEDURE Syntax and Description,”](#) on page 275,
- [“DROP PROCEDURE Syntax and Description,”](#) on page 281,
- [“Deactivating the LE Environment,”](#) on page 282,
- [“Specifying CICS Transaction Scheduling Information,”](#) on page 282, and
- [“Specifying CA-DATACOM/DB Resource Information,”](#) on page 283.

## CREATE PROCEDURE Syntax and Description

To create a stored procedure definition use the CREATE PROCEDURE statement. Sample code on [page 289](#) shows an example of deleting and defining a stored procedure.

[Figure 40: “CREATE PROCEDURE Syntax,”](#) on page 276, shows the syntax used to define a stored procedure. [Table 47, “CREATE PROCEDURE Parameters and Descriptions,”](#) describes each of the parameters that can be specified on the CREATE PROCEDURE statement.

Once you have created a CREATE PROCEDURE statement, run it through the Meta Data Utility to define it in the eXadas Meta Data Catalogs. For instructions on how to run the Meta Data Utility, see [Chapter 13, “Utilities.”](#)

Figure 40: CREATE PROCEDURE Syntax

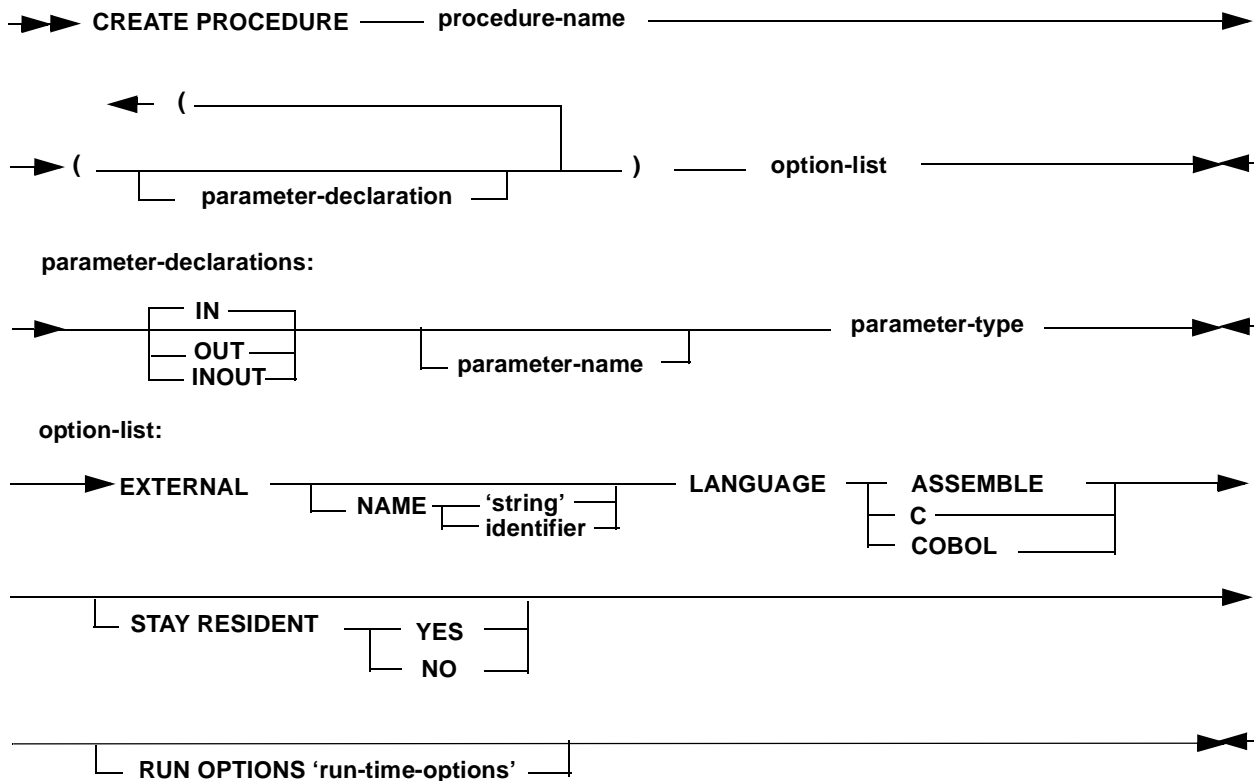


Table 47: CREATE PROCEDURE Parameters and Descriptions

Parameter	Description
<code>CREATE PROCEDURE procedure-name</code>	<p>A required keyword phrase that defines a stored procedure. <i>procedure-name</i> names the stored procedure. The name is implicitly or explicitly qualified by an owner. The name, including the implicit or explicit qualifier, must not identify an existing stored procedure at the current server.</p> <p>To explicitly specify an owner use the syntax <i>owner-name.procedure-name</i>. The <i>owner-name</i> can be 1-to-8 characters long and the <i>procedure-name</i> can be 1-to-18 characters long. If an <i>owner-name</i> is not specified, the implicit <i>owner-name</i> is the TSO user ID of the person that runs the Meta Data Utility to define the stored procedure.</p>

**Table 47: CREATE PROCEDURE Parameters and Descriptions**

Parameter	Description
parameter-declaration	<p>Specifies the parameters that are passed to the stored procedure application program and that must be supplied by the client application when the stored procedure is invoked. At least one parameter must be defined for a stored procedure. There is no fixed upper limit on the number of parameters that can be defined for a stored procedure. The maximum number of parameters that can be defined is dependent on the size of the resultant SQLDA. The maximum data size for all data and indicator variables is 32767-bytes.</p> <p>A stored procedure parameter may be used for input, output or both input and output. The options are:</p> <ul style="list-style-type: none"> <li>• IN: The default. It identifies the parameter as an input parameter to the stored procedure application program. The parameter contains its original value when the stored procedure returns control to the client application.</li> <li>• OUT: Identifies the parameter as an output parameter. The stored procedure application program returns a value to the client application or a null-indicator that indicates no value is being returned.</li> <li>• INOUT: Identifies the parameter as both an input and an output parameter. The client application must supply a value for an INOUT parameter, upon return the stored procedure application program may have changed this value.</li> </ul>

**Table 47: CREATE PROCEDURE Parameters and Descriptions**

Parameter	Description
parameter-name	<p>The <i>parameter-name</i> specifies the name of the parameter in the parameter-declaration. The name can be up to 30 characters long, must be unique within the stored procedure definition and cannot be IN, OUT, or INOUT.</p> <p>Parameter names are optional, but specifying a <i>parameter-name</i> is highly recommended. When initially testing your stored procedure your client application may receive various SQL codes indicating that an incompatible data type was passed, a NULL parameter is not allowed, or other similar codes. To resolve these problems you can activate tracing in the server. When activated, a log message is generated for the parameter in error and the information identifies the <i>parameter-name</i>, its data type, and length. Naming your parameters, therefore, makes problem resolution easier in these situations.</p>
parameter-type	<p>Required keyword that identifies the SQL data type of the parameter. The list of stored procedure supported SQL data types is found in <a href="#">Table 48, “Stored Procedure Supported Data Types.”</a></p>
option-list	<p>List of options to be used for the stored procedure.</p>
EXTERNAL NAME ‘string’ or identifier	<p>Specifies the OS/390 load module of the application program that the Server should load to satisfy a call for the stored procedure. If you do not specify the NAME clause, NAME procedure-name is implicit. The procedure-name is limited to 8 characters.</p> <p>When an explicit name is specified it can be from 1-to-8 characters long and can be supplied either as a quoted string or as an identifier. A quoted string is required if the application program name matches any of the keywords supplied on the CREATE PROCEDURE statement.</p>

**Table 47: CREATE PROCEDURE Parameters and Descriptions**

Parameter	Description
LANGUAGE	<p>Required parameter that identifies the programming language in which the stored procedure application program was written. All programs should be designed to run in IBM's Language Environment (LE). Valid values are:</p> <ul style="list-style-type: none"> <li>• ASSEMBLE: Assembler Language</li> <li>• COBOL: IBM COBOL.</li> <li>• C: IBM or SAS/C C Language</li> </ul>
STAY RESIDENT	<p>Identifies whether the Stored Procedure Data Savant should unload the stored procedure application program after it has been executed. Options are:</p> <ul style="list-style-type: none"> <li>• NO: Unload the program after each execution. When initially testing your stored procedure you should use this option. This allows you to modify your stored procedure application program and re-test it without having to shutdown the Query Processor that received the client application request to execute your stored procedure application program.</li> <li>• YES: Do not unload the stored procedure application program after it has completed execution. For performance purposes, YES should be specified once your stored procedure application program has been tested.</li> </ul>

**Table 47: CREATE PROCEDURE Parameters and Descriptions**

Parameter	Description
RUN OPTIONS <i>run-time-options</i>	<p>Specifies the Language Environment run-time options to be used for the stored procedure application program. You must specify run-time-options as a character string no longer than 254 bytes. If you do not specify RUN OPTIONS or pass an empty string, the Stored Procedure Data Savant passes a value of ALL31(OFF).</p> <p>For a description of the LE run-time options, see <i>IBM's OS/390 Language Environment for OS/390 &amp; VM Programming Reference</i>.</p> <p>Additionally, eXadas has extended the use of the RUN OPTIONS parameter to disable executing a stored procedure application program in an LE environment and to supply CICS transaction scheduling information or CA-DATACOM/DB resource information. For details on each of these extensions see <a href="#">“Deactivating the LE Environment,” on page 282</a>, <a href="#">“Specifying CICS Transaction Scheduling Information,” on page 282</a>, and <a href="#">“Specifying CA-DATACOM/DB Resource Information,” on page 283</a>.</p> <p><b>NOTE:</b> RUN OPTIONS information must be supplied on a single input line. If you need to specify a RUN OPTIONS string that exceeds 80 characters, you can use a variable length file.</p>

A stored procedure parameter can be defined as one of the types listed in [Table 48, “Stored Procedure Supported Data Types.”](#) Also see the code sample on [page 290](#) for examples on how to define each of these data types in a COBOL program.

**Table 48: Stored Procedure Supported Data Types**

eXadas-Supported Data Types	Description (n is always a decimal integer)
INTEGER	Fullword signed hexadecimal, 32-bits, no decimal point.
SMALLINT	Halfword signed hexadecimal, 16-bits, no decimal point.
DECIMAL( <i>p</i> [, <i>s</i> ])	Packed decimal $1 \leq p \leq 31$ and $0 \leq s < p$ where: <ul style="list-style-type: none"> <li><i>p</i> is the precision (total number of digits) and</li> <li><i>s</i> is the total number of digits to the right of the decimal point.</li> </ul>
FLOAT	4-byte single precision floating point number.



**Table 48: Stored Procedure Supported Data Types**

eXadas-Supported Data Types	Description (n is always a decimal integer)
DOUBLE	8-byte double precision floating point number.
CHAR (n)	Fixed-length character string of length n where $1 \leq n \leq 254$ .
VARCHAR (n)	Variable-length character string where $1 \leq n \leq 32704$ .
GRAPHIC (n)	Fixed-length, double-byte character set (DBCS) string where $1 \leq n \leq 127$ . The value of n specifies the number of DBCS characters. For example, GRAPHIC(10) specifies a parameter that occupies 20 bytes of storage.
VARGRAPHIC (n)	Variable-length, DBCS string where $1 \leq n \leq 16351$ . The value of n specifies the number of DBCS characters. For example, VARGRAPHIC(10) specifies a parameter that occupies 20 bytes of storage.

## DROP PROCEDURE Syntax and Description

To delete a stored procedure definition from the eXadas Meta Data Catalogs, use the DROP PROCEDURE statement. Sample code on [page 289](#) shows an example of how to delete and then define a stored procedure definition. The following table describes the parameters supplied on the DROP PROCEDURE statement.

**Table 49: DROP PROCEDURE Parameters and Descriptions**

Parameter	Description
DROP PROCEDURE	Statement that removes an existing stored procedure from the Meta Data Catalogs. This statement is required when replacing an existing stored procedure with a stored procedure of the same name.
<i>procedure-name</i>	Identifies the stored procedure to be dropped. The name must identify a stored procedure that has been defined with the CREATE PROCEDURE statement at the current server. When a procedure is dropped, all privileges on the procedure are also dropped.

## Deactivating the LE Environment

If your stored procedure application program cannot be executed in an LE environment (for example, it is written in Assembler, and not using LE services), specify `NO_LE` on the `RUN OPTIONS` parameter. When `NO_LE` is specified and the Stored Procedure Data Savant executes the stored procedure application program is branch-entered regardless of whether the LE Initialization Service is active or not.

If you are using the `RUN OPTIONS` parameter to also specify CICS transaction scheduling information or `CA-DATACOM/DB` resource information, `NO_LE` **must** be specified at the beginning of the `RUN OPTIONS` parameter, followed by the CICS transaction scheduling information or `CA-DATACOM/DB` resource information.

**NOTE:** If your stored procedure application program cannot run in an LE environment, specify `NO_LE` even if the LE Initialization Service is not going to be used in the server. This will prevent your stored procedure application program from abending at a later time when some other group develops a LE-enabled application and decides to active the LE Initialization Service for performance reasons.

## Specifying CICS Transaction Scheduling Information

If you are using the eXadas-supplied `CACSPVTM` stored procedure application program to communicate with CICS, then you must specify the information necessary for `CACSPVTM` to communicate with CICS and invoke the CICS transaction using the `RUN OPTIONS` parameter. If you are writing your own stored procedure application program to communicate with CICS, then you should specify this information using the `RUN OPTIONS` parameter instead of hard coding it in your stored procedure application program. Doing so allows you to simply drop the stored procedure definition, update the `CREATE PROCEDURE` statement and then re-define the stored procedure when your environment changes without requiring program changes.

**WARNING:** If you are specifying CICS transaction scheduling information for use by `CACSPVTM`, then you must also specify `NO_LE` on the `RUN OPTIONS` parameter.

CICS transaction scheduling information is identified by the `_CICS` keyword in the `RUN OPTIONS` parameter. The format of the `_CICS` keyword is:

```
_CICS(Local-LU-Name, CICS-Applid, Logmode-Table-Name, Transaction-ID, Program-Name)
```

All sub-parameters must be supplied, comma delimited as shown, and must not contain any spaces. Descriptions of the size and purpose of each sub-parameter are identified in the following table.

**Table 50: CICS Transaction Scheduling Sub-Parameters**

Sub-Parameter Name	Maximum Length	Description
Local-LU-Name	8	Identifies the name of a pool of VTAM logical units that can be used by CACSPBR to communicate with CICS, such as CACPPC0*.  In the above example, CACSPBR initially attempts to open an ACB for LU name CACPPC00. If an ACB open error is returned, or a CNOS negotiation error is reported by CICS, then CACSPBR attempts to open an ACB named CACPPC01. If that fails LU names CACPPC02-CACPPC09 are tried until no errors are reported or all names have been attempted.  Only one set of wild-card characters can be specified (for example, CAC*PC0* is invalid). Up to seven wild card characters can be supplied (not recommended).
CICS-Applid	8	Identifies the APPLID of the CICS target subsystem.
Logmode-Table-Name	8	Identifies the VTAM logmode table entry to be used. This name must identify an entry in the Logon Mode Table definition, in the Local LU Name APPL definitions and in the CICS SESSIONS definitions.
Transaction-ID	4	Identifies the name of the CICS transaction that has been defined for CACSP62 to allow communications between the Server and CICS. This name must be 4 characters long.
Program-Name	8	Identifies the name of the CICS program that CACSP62 is to LINK to. This name must be defined as a PROGRAM to CICS.

## Specifying CA-DATACOM/DB Resource Information

When writing your own stored procedure application program to communicate with CA-DATACOM/DB, you should specify the User Requirements Table name using the RUN OPTIONS parameter instead of hard coding it in your stored procedure application program. Doing so allows you to simply drop the stored procedure definition (DROP PROCEDURE), update the stored procedure definition and then re-catalog the stored procedure definition (CREATE PROCEDURE) when your environment changes, without requiring any changes to your stored procedure application program. If you wish to supply the User

Requirements Table name programmatically, that is also possible. The sample stored procedure application program for CA-DATACOM/DB demonstrates how this can be done.

CA-DATACOM/DB resource information is identified by the `_DATACOM` keyword in the `RUN OPTIONS` parameter. The complete format of CA-DATACOM/DB resource information entry is:

```
_DATACOM(urt-name)
```

The CA-DATACOM/DB resource information entry is separated from preceding keyword entries by a comma. Order of the keyword entries is not mandated except that if you are using the `NO_LE` keyword in the `RUN OPTIONS` statement, it must be the first keyword specified. The following examples show `RUN OPTIONS` statements with keyword entries specified in differing order. Regardless of the order, the resultant stored procedure processing is identical.

Example 1:

```
RUN OPTIONS 'NO_LE,_DATACOM(urt-name),  
_CICS(transaction-scheduling-info)'
```

```
RUN OPTIONS 'NO_LE,  
_CICS(transaction-scheduling-info),_DATACOM(urt-name)'
```

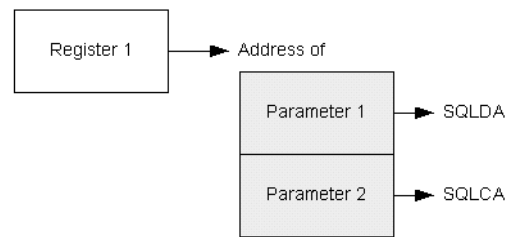
Example2:

```
RUN OPTIONS '_DATACOM(urt-name),  
_CICS(transaction-scheduling-info)'
```

```
RUN OPTIONS '_CICS(transaction-scheduling-info),  
_DATACOM(urt-name)'
```

## Writing Stored Procedures

A stored procedure application program is invoked using standard Assembler Language linkage conventions. These programs can be written in C, COBOL or Assembler Language. The stored procedure application program is always passed two parameters: an `SQLDA` structure and an `SQLCA` structure. The following figure shows how these parameters are passed. All parameters and the data and indicator values contained in the `SQLDA` are 31-bit addresses.

**Figure 41: Parameters Passed to the Stored Procedure Application Program**

Upon return from the Stored Procedure application program, Register 15 is assumed to contain a return code value. If a non-zero value is found in Register 15, it will be returned to the client application unless there is a non-zero value in the SQLCODE field in the SQLCA.

**WARNING:** Your stored procedure application program must be coded as a subroutine and must not issue a function call that causes the run-time environment to be terminated. For example, a COBOL stored procedure application program must not issue `STOP RUN` and must instead return control using the `GOBACK` statement. If you terminate the run-time environment (for example, using `STOP RUN` in COBOL), this causes the Query Processor task to also be terminated.

The SQLDA structure consists of a 16-byte header followed by a variably-occurring array of SQLVAR structures. Each SQLVAR represents a parameter-declaration from the `CREATE PROCEDURE` statement. The SQLVARs are passed in the sequence defined on the `CREATE PROCEDURE` statement. Each SQLVAR structure is 44-bytes long.

The sample member `CACSPSDA` is a COBOL copybook that shows the structure and contents of the SQLDA structure. The following figure shows the contents of the SQLDA header. [Table 51, “SQLDA Header Contents,”](#) describes the format and contents of the SQLDA header.

The SQLDA structure is as follows:

```

01  ARG-DATA.
   05  ARG-SQLDAID                PIC X(8) .
   05  ARG-SQLDABC                PIC 9(8) COMP .
   05  ARG-SQLN                  PIC 9(4) COMP .
   05  ARG-SQLD                  PIC 9(4) COMP .
   05  ARG-SQLVAR OCCURS 1 TO 2000 TIMES
        DEPENDING ON ARG-SQLN.
      10  ARG-SQLTYPE              PIC 9(4) COMP .
         88  ARG-SQL-VARCHAR        VALUE 449 .
         88  ARG-SQL-CHAR          VALUE 453 .
         88  ARG-SQL-VARGRAPHIC    VALUE 465 .
         88  ARG-SQL-GRAPHIC       VALUE 469 .
         88  ARG-SQL-FLOAT         VALUE 481 .
         88  ARG-SQL-DECIMAL       VALUE 485 .
         88  ARG-SQL-INTEG        VALUE 497 .
         88  ARG-SQL-SMALLINT      VALUE 501 .

      10  ARG-SQLLEN                PIC 9(4) COMP .
      10  ARG-SQLDATA              POINTER .
      10  ARG-SQLLIND              POINTER .
      10  ARG-SQLNAME .
  
```

```

20 ARG-NAME-LEN          PIC 9(4) COMP.
20 ARG-NAME-LABEL       PIC X(30).

```

The following table describes the format and contents of the SQLDA header.

**Table 51: SQLDA Header Contents**

COBOL Name	SQL Data Type	Description
ARG-SQLDAID	CHAR(8)	Signature that identifies the structure as an SQLDA. Always contains the value "SQLDA" followed by three spaces.
ARG-SQLDABC	INTEGER	Identifies the length of the SQLDA structure and is computed as $16 + (\text{SQLN} * 44)$ .
ARG-SQLN	SMALLINT	Identifies the number of SQLVAR entries contained in the SQLDA.
ARG-SQLD	SMALLINT	Identifies the number of SQLVAR entries contained in the SQLDA. Same as ARG-SQLN.

The following table describes the format and contents of the SQLVAR structure.

**Table 52: SQLVAR Contents**

COBOL Name	SQL Data Type	Description
ARG-SQLTYPE	SMALLINT	Identifies the type of data that is referenced by field ARG-SQLDATA. See <a href="#">Table 53, “SQL Data Type Descriptions,”</a> for a description of the different data types that are supported for stored procedures.
ARG-SQLLEN	SMALLINT	Identifies the length of the data that is referenced by field ARG-SQLDATA.
ARG-SQLDATA	INTEGER	Pointer to the argument data. Before using this pointer value you must check the 2-byte data value referenced by ARG-SQLIND to determine whether the argument data is null (for example, a value was not supplied by the client application). If the argument data is null, the data referenced is low-values and should not be referenced. If the data is not null, the size of the data referenced is identified by ARG-SQLLEN.
ARG-SQLIND	INTEGER	Pointer to a 2-byte (half-word - SMALLINT) indicator field that identifies whether ARG-SQLDATA is null or not. If the indicator field contains zeros, the data referenced by ARG-SQLDATA is not null and contains valid data matching the SQL data type identified in ARG-SQLTYPE. If the indicator field contains -1 (x'ffff') then the data referenced by ARG-SQLDATA is binary zeros and should not be referenced.
ARG-SQLNAME	VARCHAR(30)	The <i>parameter-name</i> specified in the CREATE PROCEDURE statement for this parameter. If no <i>parameter-name</i> was specified, ARG-NAME-LEN is zeros and ARG-NAME-LABEL is low-values. If a <i>parameter-name</i> was specified, ARG-NAME-LEN identifies how long <i>parameter-name</i> is and ARG-NAME-LABEL contains the <i>parameter-name</i> left justified and padded with blanks.

The following table describes the SQL data types that can be passed to the stored procedure application program and the length of the data referenced by ARG-

SQLDATA for each data type.

**Table 53: SQL Data Type Descriptions**

COBOL Name	Value	Value in ARG-SQLLEN and Corresponding Length of Data Referenced by ARG-SQLDATA
ARG-SQL-VARCHAR	449	Identifies that ARG-SQLDATA references a variable length character field. ARG-SQLLEN identifies the maximum length that the variable length character field may be, excluding the 2-byte length field. The actual size of the data is identified in the 2-byte length field.
ARG-SQL-CHAR	453	Identifies that ARG-SQLDATA references a fixed length character field. ARG-SQLLEN identifies the length of the character field.
ARG-SQL-VARGRAPHIC	465	Identifies that ARG-SQLDATA references a variable length graphic field. ARG-SQLLEN identifies the maximum length (in DBCS characters) that the variable length graphic field may be, excluding the 2-byte length field. The actual size of the data (in DBCS characters) is identified in the 2-byte length field.
ARG-SQL-GRAPHIC	469	Identifies that ARG-SQLDATA references a fixed length graphic field. ARG-SQLLEN identifies the length of the graphic field in DBCS characters.
ARG-SQL-FLOAT	481	Identifies that ARG-SQLDATA references a floating point number. The field is a single precision floating point number that is 4-bytes long if FLOAT was specified on the parameter-declaration on the CREATE PROCEDURE statement. If DOUBLE was specified, then the field is a double-precision floating point number that is 8-bytes long.
ARG-SQL-DECIMAL	485	Identifies that ARG-SQL-DATA references a signed packed decimal field. The first byte of ARG-SQL-LEN identifies the scale (number of digits) in the decimal field. The second byte of ARG-SQL-LEN identifies the precision (implied decimal point) of the decimal data. If the scale is an even number, the physical length of the decimal data is scale / 2. If the scale is an odd number, the physical length of the data is (scale +1)/2.
ARG-SQL-INTEGER	497	Identifies that ARG-SQL-DATA references a signed full-word field that is 4-bytes long.
ARG-SQL-SMALLINT	501	Identifies that ARG-SQL-DATA references a signed half-word field that is 2-bytes long.

The SQLCA structure is much simpler than the SQLDA. Sample member



CACSPSCA is a COBOL copybook that shows the structure and contents of the SQLCA structure. The following sample code shows the contents of this copy book, and [Table 54, “SQLCA Contents,”](#) describes the contents of the SQLCA.

The SQLCA structure is as follows:

```
01 ARG-SQLCA
   05 ARG-SQLCAID          PIC X(8) .
   05 ARG-SQLCABC         PIC 9(8) COMP .
   05 ARG-SQLCODE        PIC 9(8) COMP .
```

**Table 54: SQLCA Contents**

COBOL Name	SQL Data Type	
ARG-SQLCAID	CHAR(8)	Signature that identifies the structure as an SQLCA. Always contains the value “SQLCA” followed by three spaces.
ARG-SQLCABC	INTEGER	Identifies the length of the SQLCA structure.
ARG-SQLCODE	INTEGER	Full-word 32-bit signed integer that is returned to the client application when SQLCODE contains a non-zero value.

If you inspect the ARG-SQLCABC field you will find that its contents report a length that is longer than the structures contents identified above. A full DB2-style SQLCA is passed to your stored procedure application program. These additional fields are not being documented, as they are not inspected by the Stored Procedure Data Savant or the Query Processor and are not returned to the client application.

Sample member CACSPCP shows a sample stored procedure definition that contains parameter definitions for all of the different SQL data types that can be passed to a stored procedure application program. Its contents are shown in the following figure.

The following is a sample DROP and CREATE procedure definition:

```
DROP PROCEDURE CAC.DATA_TYPES;
CREATE PROCEDURE CAC.DATA_TYPES
(IN DT_SMALLINT SMALLINT,
 IN DT_INTEGER INTEGER,
 IN DT_CHAR CHAR(10),
 IN DT_VARCHAR VARCHAR(20),
 IN DT_FLOAT FLOAT,
 IN DT_DOUBLE DOUBLE,
 IN DT_DECIMAL DECIMAL(9,2),
 IN DT_GRAPHIC GRAPHIC(10),
 IN DT_VARGRAPHIC VARGRAPHIC(20))
EXTERNAL NAME CACSPDAT
LANGUAGE COBOL
STAY RESIDENT NO;
```

This figure shows that the stored procedure application program name is CACSPDAT and it is written in COBOL. For a COBOL program, the field definitions for the actual data values and their associated indicator variables must

be defined in the LINKAGE SECTION. Sample code on [page 290](#) shows the corresponding LINKAGE SECTION data and indicator definitions for the parameters for the CAC.DATA\_TYPES stored procedure definition.

While the COBOL field names do not have to match the parameter names specified on the stored procedure CREATE PROCEDURE definition statement, it is a good practice. Establish a standard for naming null indicator variables. Sample code on [page 290](#) shows how to append IND to the name of the field that references the null indicator value for each parameter.

In COBOL, addressability to the data associated with a parameter and its null indicator field is established by using the SET ADDRESS OF statement. Sample code [page 291](#) shows how to establish addressability to the parameter data and associated null indicators for the CAC.DATA\_TYPES stored procedure.

You should also establish addressability to a parameter's null indicator and test to see whether the value is not null before establishing addressability to its data value. Likewise, once addressability is established before referencing a data value check its associated null indicator before attempting to manipulate a data value. This practice should be done even if you know that a value does not contain a null value. This practice prevents abends when, at a later date, maintenance is performed on the client application and the developer does not follow the "rules."

The following are sample COBOL data and indicator definitions:

```
LINKAGE SECTION.
COPY CACSPSDA.
COPY CACSPSCA.

01 DT-SMALLINT                PIC S9(4) COMP.
01 DT-=SMALLINT-IND          PIC S9(4) COMP.

01 DT-INTEGGER                PIC S9(9) COMP.
01 DT-INTEGGER-IND          PIC S9(4) COMP.

01 DT-CHAR                    PIC X(10).
01 DT-CHAR-IND              PIC S9(4) COMP.

01 DT-VARCHAR.
   05 DT-VARCHAR-LEN          PIC 9(4) COMP.
   05 DT-VARCHAR-DATA        PIC X(1)
                              OCCURS 20 TIMES
                              DEPENDING ON DT-VARCHAR-LEN.
01 DT-VARCHAR-IND          PIC S9(4) COMP.

01 DT-FLOAT                   COMP-1.
01 DT-FLOAT-IND             PIC S9(4) COMP.

01 DT-DOUBLE                  COMP-2.
01 DT-DOUBLE-IND           PIC S9(4) COMP.

01 DT-DECIMAL                 PIC S9(7)V99 COMP-3
01 DT-DECIMAL-IND          PIC S9(4) COMP.

01 DT-GRAPHIC                 PIC G(10)
                              USAGE DISPLAY-1.
01 DT-GRAPHIC-IND          PIC S9(4) COMP.
01 DT-VARGRAPHIC.
```

```

05 DT-VARGRAPHIC-LEN      PIC 9(4) COMP.
05 DT-VARGRAPHIC-DATA    PIC G(1)
                           USAGE DISPLAY-1
                           OCCURS 20 TIMES
                           DEPENDING ON DT-VARGRAPHIC-LEN.
01 DT-VARGRAPHIC-IND     PIC S9(4) COMP.

```

The following shows how to establish addressability:

PROCEDURE DIVISION USING ARG-DATA, ARG-SQLCA.

```

SET ADDRESS OF DT-SMALLINT-IND TO ARG-SQLIND(1).
IF DT-SMALLINT-IND = ZEROS
    SET ADDRESS OF DT-SMALLINT TO ARG-SQLDATA(1).

SET ADDRESS OF DT-INTEGGER-IND TO ARG-SQLIND(2).
IF DT-INTEGGER-IND = ZEROS
    SET ADDRESS OF DT-INTEGGER TO ARG-SQLDATA(2).

SET ADDRESS OF DT-CHAR-IND TO ARG-SQLIND(3).
IF DT-CHAR-IND = ZEROS
    SET ADDRESS OF DT-CHAR TO ARG-SQLDATA(3).

SET ADDRESS OF DT-VARCHAR-IND TO ARG-SQLIND(4).
IF DT-VARCHAR-IND = ZEROS
    SET ADDRESS OF DT-VARCHAR TO ARG-SQLDATA(4).

SET ADDRESS OF DT-FLOAT-IND TO ARG-SQLIND(5).
IF DT-FLOAT-IND = ZEROS
    SET ADDRESS OF DT-FLOAT TO ARG-SQLDATA(5).

SET ADDRESS OF DT-DOUBLE-IND TO ARG-SQLIND(6).
IF DT-DOUBLE-IND = ZEROS
    SET ADDRESS OF DT-DOUBLE TO ARG-SQLDATA(6).

SET ADDRESS OF DT-DECIMAL-IND TO ARG-SQLIND(7).
IF DT-DECIMAL-IND = ZEROS
    SET ADDRESS OF DT-DECIMAL TO ARG-SQLDATA(7).

SET ADDRESS OF DT-GRAPHIC-IND TO ARG-SQLIND(8).
IF DT-GRAPHIC-IND = ZEROS
    SET ADDRESS OF DT-GRAPHIC TO ARG-SQLDATA(8).

SET ADDRESS OF DT-VARGRAPHIC-IND TO ARG-SQLIND(9).
IF DT-VARGRAPHIC-IND = ZEROS
    SET ADDRESS OF DT-VARGRAPHIC TO ARG-SQLDATA(9).

```

Your stored procedure application program **must not** modify the contents of the SQLDA structure. Your program can modify the contents of the data referenced by ARG-SQLDATA and ARG-SQLIND fields. Data that is modified for output or input-output fields is returned to the client application.

When your application modifies an SQLIND indicator field to indicate that the corresponding data is null, then any modifications to the corresponding data field are not returned to the client application. Additionally, your application should not modify an SQLIND indicator field that identifies the corresponding data field as being null to indicate that the data field (upon return from your stored procedure application program) now contains valid data. If you do so this is likely to cause problems in the client application since it is not expecting to receive data for a parameter that it knows is null.

Your stored procedure application program can call other stored procedure application programs or one or more of your existing application programs. However, if you call existing applications ensure that these are written as subroutines.

If you are not using the eXadas-supplied CICS, CA-DATACOM/DB, or IMS interfaces, you compile and link your stored procedure application program using the standard procedures for the language that the application program is written in. If you are using the CICS, CA-DATACOM/DB, and/or IMS interfaces, the procedures are slightly modified so that the interface load modules are included in the link step. These modifications are described in detail in [“CICS Interface Description,” on page 297](#), [“CA-DATACOM/DB Interface Description,” on page 310](#), and [“IMS DRA Interface Description,” on page 320](#).

Once you have linked an executable copy of your stored procedure application then either copy it into one of the load libraries referenced in the Server JCL or update the server JCL to include the library that your application resides in on the STEPLIB DD statement. Also add any additional DD statements in the Servers JCL that your stored procedure application program references.

Once these modifications have been performed start the Server. You are now ready to start testing your stored procedure application program. You will need to do this from a client application. Descriptions of the possible techniques that you can use to invoke a stored procedure are discussed next.

## Invoking Stored Procedures

As shown in [Figure 35: “Stored Procedure Execution Flow,” on page 263](#), your stored procedure application program is invoked by a client application that issues a SQL CALL statement. All of the supplied client connectors contain support for the CALL statement.

The remainder of this chapter describes the basic CALL statement syntax that is used to invoke a stored procedure using SQL. Then, for each connector, we describe how to obtain meta data about stored procedures and the connector-specific API(s) that you can use to issue the CALL statement.

The following topics are discussed:

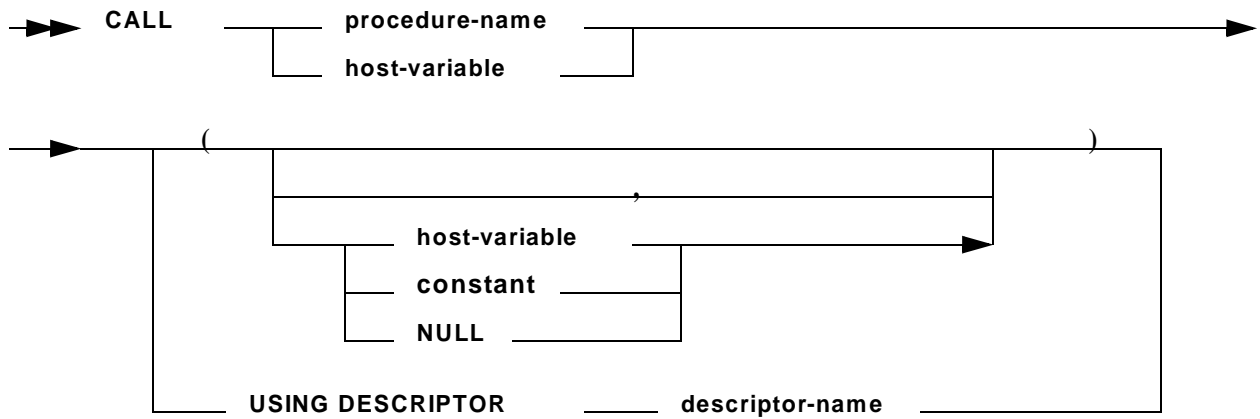
- [“CALL Statement Syntax,” on page 293](#), and
- [“ODBC Stored Procedure Support,” on page 295](#).

## CALL Statement Syntax

The following figure shows the format of the CALL statement supported by eXadas.

**NOTE:** Client applications using ODBC or JDBC Connectors cannot use the 'USING DESCRIPTOR' form of this statement and cannot use indicator host-variables (they use parameter markers instead).

**Figure 42: CALL Statement Syntax**



The CALL parameters are described the following table.

**Table 55: CALL Statement Parameters**

Parameter	Description
<i>procedure-name</i> or <i>host-variable</i>	<p>Identifies the name of the stored procedure to be executed. The name should generally be specified in the form <i>owner.name</i>. If an owner is not specified, the connected user's user ID is used for the owner. If a user ID is not supplied on connect the owner is PUBLIC.</p> <p>You can either specify the name of the stored procedure explicitly as an identifier <i>procedure-name</i>, or you can use a host-variable reference, where <i>host-variable</i> contains the name of the stored procedure to be executed.</p>
host-variable	<p>The name of a <i>host-variable</i> that, for input or input-output parameters contains the data that is passed to the stored procedure application program. For output parameters, the <i>host-variable</i> identifies a storage location that will be updated with the value supplied by the stored procedure application program upon return from the CALL statement.</p> <p>The host variables data type must be compatible with the data type defined for the parameter.</p> <p>If a null indicator variable is specified it can only contain a -1 for an output parameter.</p>
constant	<p>A numeric or string constant that will be passed to the stored procedure application program. Constants can only be specified for input parameters. Additionally, the constant must match the data-type for the parameter. That is, a numeric constant of the appropriate scale and precision must be supplied for INTEGER, SMALLINT, DECIMAL, FLOAT and DOUBLE parameter types. For CHAR or VARCHAR parameters, the constant must be in single quotes. Constants cannot be specified for GRAPHIC or VARGRAPHIC data types.</p>
NULL	<p>Identifies that no value is being supplied for the parameter. NULL can only be specified for input parameters.</p> <p>This causes the parameters associated null indicator to be set to -1 when the stored procedure application program is called.</p>
USING DESCRIPTOR <i>descriptor-name</i>	<p>Identifies that the parameters to be passed to the stored procedure application program are contained in the SQLDA structure referenced by <i>descriptor-name</i>. The correct number and types of parameters must be contained in the SQLDA. Additionally, SQLN must identify the number of parameters being passed and SQLDABC must contain the correct value - (SQLN * 44) + 16.</p>

## ODBC Stored Procedure Support

The eXadas ODBC Connector implements the standard ODBC API interfaces used to obtain meta data information about the stored procedures defined for the data source that the ODBC Connector is connected to and to invoke a stored procedure. However, there are slight nuances between the eXadas implementation and the descriptions in the *Microsoft ODBC Programmers Reference and SDK Guide*.

Obtaining the list of stored procedures that are defined in the Meta Data Catalogs for a data source is achieved by using the SQLProcedures call. To retrieve parameter information about one or more stored procedure definitions use the SQLColumnProcedures call. eXadas supports the syntax and options described in the Microsoft ODBC documentation for these calls with the following exceptions:

- eXadas does not support qualifier names so szProcQualifier should always be null and cbProcQualifier should always be zero.
- When retrieving the result sets from these calls, TABLE\_QUALIFIER is always null.
- The REMARKS field is always spaces.
- For SQLProcedure calls, PROCEDURE\_TYPE is always SQL\_PT\_UNKNOWN.
- When retrieving the result set from the SQLProcedureColumns call, the COLUMN\_TYPE will only be one of the following values:
  - SQL\_PARAM\_INPUT
  - SQL\_PARAM\_INPUT\_OUTPUT
  - SQL\_PARAM\_OUTPUT
- RADIX is always 10.
- NULLABLE is always set to SQL\_NULLABLE\_UNKOWN.

For more information on how to issue the SQLProcedures and SQLProcedureColumns calls and retrieve their result sets see the *Microsoft ODBC Programmer's Reference and SDK Guide*.

To invoke a stored procedure use the SQLExecDirect call. eXadas supports the following two formats of the CALL statement that can be supplied on the SQLExecDirect call:

- The shorthand syntax which is “[[?]=]call *procedure-name*(*parameter*,[*parameter*],...)”
- “CALL *procedure-name*(*parameter*,[*parameter*],...)”

Before issuing the SQLExecDirect call you may have to issue one or more SQLBindParameter calls for any parameter markers that are contained in the CALL statement.

Following are the rules for when parameter markers can and/or must be used:

- For output or input-output parameters, *parameter* must be a parameter marker
- For input parameters, *parameter* can be a literal, a parameter marker or NULL.
- For the short format, the return value must be a parameter marker.
- *procedure-name* may either be supplied as an identifier or using a parameter marker.

Unlike the ODBC documentation, all parameters defined for the stored procedure **must** be supplied on the CALL statement. Additionally, the eXadas implementation does not support returning multiple result set rows from a stored procedure.

If you use the second format of the CALL statement or the shorthand format without a return value and your stored procedure application returns a non-zero return code (or the stored procedure application program sets a non-zero value for SQLCODE in the SQLCA) then:

- the RETCODE will either be set to SQL\_ERROR or SQL\_SUCCESS\_WITH\_INFO, depending on the return code (or SQLCODE) value, or you will need to
- issue the SQLError call and inspect the pfNativeError parameter to obtain the return code (or SQLCODE) value.

If you wish to use the shorthand version of the CALL and specify a return code value, then the following rules must be followed when defining the stored procedure:

- The first *parameter-declaration* must be OUT.
- The first *parameter-declaration parameter-name* must be RC.
- The first *parameter-declaration parameter-type* must be INTEGER.

**NOTE:** If your stored procedure application program uses non-zero return code values to report warning conditions that are not errors, you must use the short-hand format specifying an RC value if you want to inspect any data returned from the stored procedure application program. When a non-zero return code is encountered, the Query Processor will not update the SQLDA returned to the client with any updated output or input-output parameters that the stored procedure application program may have modified. If you specify an RC parameter and a non-zero return code is encountered, the Stored Procedure Data Savant updates the RC parameter with the return code value and returns zeros to the Query Processor. This allows any updated values to be returned to the client and the SQLExecDirect call to report SQL\_SUCCESS.

For more information on how to issue the SQLExecDirect and SQLBindParameter calls to invoke a stored procedure see the *Microsoft ODBC Programmer's Reference and SDK Guide*.



---

# CICS Interface Description

[“Interfacing with CICS,” on page 267](#), explained how eXadas supplies the CACSPBR interface that allows a stored procedure application program running in the server’s address space to communicate with CICS to execute a CICS application program. eXadas supplies the CACSPVTM stored procedure application program that uses CACSPBR to initiate an APPC conversation with CICS and then LINK to an application program identified in the RUN OPTIONS parameter definition for the stored procedure. CACSPVTM sends the SQLDA supplied by the client application to the CICS application program. Once the application program completes execution, CACSPVTM receives (a possibly) updated version of the SQLDA (from the CICS application), deallocates the APPC conversation and returns the SQLDA to the client application.

In the majority of situations where you need to execute a CICS application program from a stored procedure application program, you can use CACSPVTM simply by writing a CICS application program. For those rare situations when CACSPVTM does not fit your needs, this section documents the API interface to CACSPBR. Writing your own stored procedure application program to interface with CICS is required when you need to invoke two or more CICS applications or when you need to perform processing in the server’s address space and then execute a CICS application program.

CACSPVTM is written in Assembler and source code is not available. To assist you in understanding how to interface with CACSPBR eXadas supplies a COBOL version of CACSPVTM in sample library SCACSAMP (CACSPCOM).

The remainder of this section describes how to interface with CACSPBR, how to compile and link applications that call CACSPBR, describes the possible error return codes that your stored procedure application may receive from CACSPBR, and possible CICS abend codes that you may receive from CACSP62.

The following topics are discussed:

- [“CACSPBR Interface Description,” on page 297](#),
- [“Parameters Passed to the CICS Application Program,” on page 303](#),
- [“Compiling and Linking Applications that Use CACSPBR,” on page 304](#),
- [“CACSPBR Return Codes,” on page 305](#), and
- [“CACSP62 Abend Codes,” on page 306](#).

## CACSPBR Interface Description

The CICS interface bridge (CACSPBR) is distributed as a separate load module that may be dynamically called or statically linked into your stored procedure application program. See [“Compiling and Linking Applications that Use](#)

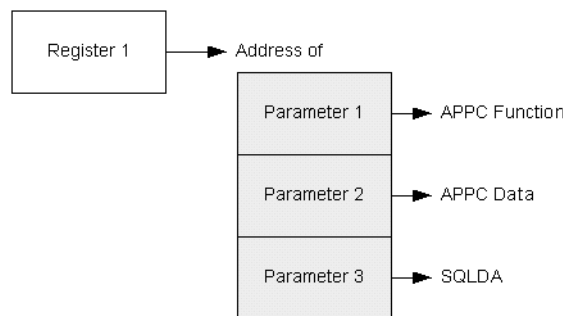
[CACSPBR,” on page 304](#), for an example of how to statically link CACSPBR into your stored procedure application program.

CACSPBR uses standard Assembler linkage conventions. CACSPBR is always passed three parameters, they are:

- an APPC function structure,
- an APPC data structure, and
- the SQLDA passed to the stored procedure application program.

The following figure shows how parameters need to be passed to CACSPBR. Additionally, CACSPBR must be called using variable-argument list calling conventions, the high-order bit of the SQLDA must be set to one. This is the standard calling convention for most high-level languages, for example COBOL. Additionally, CACSPBR is linked as an AMODE(31) RMODE(ANY) application. All addresses passed to CACSPBR must be passed in 31-bit addressing mode.

**Figure 43: Parameters Passed to CACSPBR**



Upon return from CACSPBR Register 15 contains a return code value that indicates whether the requested function completed successfully or not. See [“CACSPBR Return Codes,” on page 305](#), for a list and descriptions of the possible return codes.

Sample member CACSPVTM is a COBOL copybook that shows the structure and contents of both the APPC function and APPC data structures. The following shows the contents of that copybook:

```

01  APPC-FUNCTION.
    05  APPC-REQUEST                                PIC X(8) .

01  APPC-DATA.
    05  APPC-DATA-IDENTIFIER                        PIC X(8) .
    05  LOCAL-LU-NAME                              PIC X(8) .
    05  CICS-SYSTEM-APPLID                          PIC X(8) .
    05  APPC-MODE-ENTRY-NAME                        PIC X(8) .
    05  CICS-TRANSACTION-ID                         PIC X(4) .
    05  CICS-SP-PROGRAM-NAME                        PIC X(8) .
    05  CICS-SP-RETCODE                             PIC 9(8) COMP .
    05  CICS-SP-ABENDCODE
        REDEFINES CICS-SP-RETCODE                   PIC X(4) .
    05  COMM-RETCODE                               PIC S9(8) COMP .
  
```

The following table describes the contents of the APPC function structure.

**Table 56: APPC Function Structure Contents**

COBOL Name	SQL Data Type	Description
APPC-REQUEST	CHAR(8)	<p>Identifies the function that CACSPBR needs to perform. Valid functions are:</p> <ul style="list-style-type: none"> <li>• OPEN - Start a conversation with CICS.</li> <li>• SEND - Send the SQLDA to CICS.</li> <li>• RECEIVE - Receive an updated copy of the SQLDA from CICS.</li> <li>• CLOSE - Deallocate the CICS conversation.</li> </ul>

The following table describes the contents of the APPC Data structure.

**Table 57: APPC Data Structure Contents**

COBOL Name	SQL Data Type	Description
APPC-DATA-IDENTIFIER	CHAR(8)	Signature field that identifies the structure as the APPC Data structure. Must contain the value "APPCDATA".
LOCAL-LU-NAME	CHAR(8)	<p>Identifies the name of a pool of VTAM logical units that can be used by CACSPBR to communicate with CICS, for example, CACPPC0*.</p> <p>In the above example, CACSPBR initially attempts to open an ACB for LU name CACPPC00. If an ACB open error is returned, or a CNOS negotiation error is reported by CICS, then CACSPBR attempts to open an ACB named CACPPC01. If that fails LU names CACPPC02-CACPPC09 are tried until no errors are reported or all names have been attempted.</p> <p>Only one set of wild-card characters can be specified (for example, CAC*PC0*) is invalid. Up to seven wild card-characters can be supplied (not recommended).</p>
CICS-SYSTEM-APPLID	CHAR(8)	Identifies the APPLID of the CICS target sub-system.

**Table 57: APPC Data Structure Contents**

COBOL Name	SQL Data Type	Description
APPC-MODE-ENTRY-NAME	CHAR(8)	Identifies the VTAM logmode table entry to be used. This name must identify an entry in the Logon Mode Table definition, in the Local LU Name APPL definitions and in the CICS SESSIONS definitions.
CICS-TRANSACTION-ID	CHAR(4)	Identifies the name of the CICS transaction that has been defined for CACVT62 to allow communications between the server and CICS. This name must be 4-characters long.
CICS-SP-PROGRAM-NAME	CHAR(8)	Identifies the name of the CICS program that CACVT62 is to LINK to. This name must be defined as a PROGRAM to CICS.
CICS-SP-RETCODE	INTEGER	CICS error return code. Value is zero if no CICS errors are being reported. If CICS-SP-RETCODE is less than zero, then a CICS abend has occurred, the stored procedure application program should then inspect CICS-SP-ABENDCODE for the abend code.
CICS-SP-ABENDCODE	CHAR(4)	Abend code returned by CICS.
COMM-RETCODE	INTEGER	A communications error has been detected by CACSPBR. COMM-RETCODE contains an eXadas return code. See the <i>eXadas System Messages Guide</i> to determine what error is being reported and how to resolve the situation.

**NOTE:** Generally you should supply a value of spaces for the LOCAL-LU-NAME, CICS-SYSTEM-APPLID, APPC-MODE-ENTRY-NAME, CICS-TRANSACTION-ID, and CICS-SP-PROGRAM-NAME fields and instead define this information on the RUN OPTIONS parameter on the CREATE PROCEDURE statement. CACSPBR always uses the information supplied in RUN OPTIONS in preference to anything supplied in the APPC data fields. For more information on how to specify the above information on the RUN OPTIONS parameter, see [“Specifying CICS Transaction Scheduling Information,”](#) on page 282.

---

Each time CACSPBR is called the CICS-SP-RETCODE and COMM-RETCODE must contain zeros. Upon return from CACSPBR, the following checks must be made:

- Is the RETURN-CODE (Register 15) non-zero. If so, COMM-RETURN code may contain an eXadas communications return code. See [Table 60, “CACSPBR Return Codes,” on page 305](#), for the return code values that CACSPBR can return and whether there is a value in COMM-RETURN.
- Inspect CICS-SP-RETCODE for a non-zero value. If the value is less than zero then CICS-SP-ABENDCODE can be inspected to determine the abend code that CICS is reporting.

**NOTE:** If a non-zero COMM-RETURN or CICS-SP-RETCODE is received, it will be reported to the client application. You can suppress sending these return codes to the client application by setting the SQLCODE field in the SQLCA structure to a non-zero value.

CACSPBR must be called with the APPC-REQUEST field set to the values identified in [Table 56, “APPC Function Structure Contents,” on page 299](#), in the following sequence:

1. OPEN
2. SEND
3. RECEIVE
4. CLOSE

Multiple sends and receives may be issued, however this should only be performed when multiple CICS applications need to be executed. The stored procedure application program must pass the SQLDA structure it was passed to CACSPBR. Additionally, the SQLDA structures contents cannot be altered only the data referenced by ARG-SQLDATA and the associated null indicators referenced by ARG-SQLIND. The following table provides an overview of the

processing performed by CACSPBR and its counterpart CACSP62 for each APPC-REQUEST function.

**Table 58: Processing Overview by Function**

FUNCTION	CACSPBR Actions	CACVT62 Actions
OPEN	<p>Attempts to open a VTAM ACB based on the information supplied in fields:</p> <ul style="list-style-type: none"> <li>• LOCAL-LU-NAME</li> <li>• CICS-SYSTEM-APPLID</li> <li>• APPC-MODE-ENTRY-NAME</li> <li>• CICS-TRANSACTION-ID</li> </ul> <p>If the ACB open fails and LU pooling information was supplied in field LOCAL-LU-NAME, a new name is generated and another attempt is made until all LU names have been exhausted.</p> <p>In addition to passing the above information on the open, if a user ID and password were supplied for the connected user executing the stored procedure the user ID is also passed to CICS for security checking. If no user ID was supplied, the value NO_USER and a blank password is sent to CICS.</p>	<p>Saves the conversation ID supplied by CICS and allocates work buffers.</p>
SEND	<p>Sends the CICS-SP-PROGRAM-NAME and the SQLDA to CICS.</p>	<p>Receives a copy of the SQLDA and performs address translation so that the CICS application program can access the contents of the SQLDA ARG-SQLDATA and ARG-SQLIND fields referenced in the SQLDA.</p> <p>Issues a LINK for the program name contained in CICS-SP-PROGRAM-NAME and waits for control to return and sends the copy of the SQLDA back to CACSPBR.</p> <p>If CICS reports an abend, the abend code is sent back to CACSPBR.</p>
RECEIVE	<p>Receives a copy of the SQLDA from CICS and performs address translation so that the new copy can be referenced by the stored procedure application program.</p>	

**Table 58: Processing Overview by Function**

<b>FUNCTION</b>	<b>CACSPBR Actions</b>	<b>CACVT62 Actions</b>
CLOSE	Closes the VTAM ACB, which causes the conversation with CICS to be terminated.	Receives deallocation request and frees resources allocated when the conversations was started.

## Parameters Passed to the CICS Application Program

CICS application programs invoked by CACSP62 are passed three parameters in the communications area. Sample member CACSPDFH is a COBOL copybook that defines the communications area. The following shows the contents of that copybook:

```

01 DFHCOMMAREA .
   05 APP-RETURN-CODE          PIC 9(8) COMP .
   05 ARG-DATA-POINTER        POINTER .
   05 ARG-DATA-LENGTH         PIC 9(8) COMP .

```

The following table describes the contents of each of the fields in the communications area. Your application program must establish addressability to the SQLDA before inspecting/updating its contents.

For example, in COBOL you would issue:

```
SET ARG-DATA TO ARG-DATA-POINTER.
```

The above example assumes that you included a copy of SCACSAMP member CACSPSDA in the linkage section of your CICS application program.

**Table 59: Communication Area Contents**

COBOL Name	SQL Data Type	
APP-RETURN-CODE	INTEGER	Fullword return code value that the CICS application uses to report whether processing was successful or not. The contents of this field are placed in the CICS-SP-RETCODE field in the APPC data area for inspection by your stored procedure application program running in the Server's address space. This return code will be percolated to the client application program, unless your stored procedure application program overrides this code using the SQLCODE in the SQLCA.
ARG-DATA-POINTER	INTEGER	Address of the copy of the SQLDA that was sent to CICS. Your CICS application program must not modify this field.
ARG-DATA-LENGTH	INTEGER	The length of the SQLDA passed to your CICS application program. Your CICS application program must not modify this field.

Once your CICS application program has established addressability to the SQLDA it establishes addressability to the data and null indicator values in the SQLDA using the same techniques shown on [page 291](#).

## Compiling and Linking Applications that Use CACSPBR

The member CACSPCCC in the SCACSAMP library is a sample job stream that demonstrates how to compile and link CACSPCOM including CACSPBR. The following example provides a copy of CACSPCCC.

### To compile and link CACSPCOM using the supplied JCL:

1. Supply an appropriate job card.
2. Modify the PROC parameters (LE, COBOL, SOUT, and so on) to specify correct values for your site.
3. Modify the COMPILE step SYSIN DD statement to specify the correct source library.
4. Modify the LKED step OBJ DD statement to specify the correct object library.



5. Modify the LKED step SYSLMOD DD statement to specify the correct load library.
6. Modify the PROC statement to specify the correct high level name of the application libraries.

There are COBOL procedures, supplied by IBM, available to run compile and link programs that you may choose to use in place of the supplied JCL. If you elect to use the supplied procedures you will need to identify:

- the library containing the CACSPCOM source member;
- the library containing the CACSPBR object module;
- the load library into which the executable CACSPCOM member will be linked; and
- other variable parameters required by your chosen procedure.

For examples of the VTAM and CICS resource definitions required to execute a CICS application program using a Stored Procedure, see [Appendix B, “Sample Stored Procedure VTAM and CICS Definitions.”](#)

## CACSPBR Return Codes

If CACSPBR detects an error, or an error is reported by the VTAM LU 6.2 communication handler, CACSPBR returns a negative return code. [Table 61, “CACSP62 Abend Codes,” on page 307](#), identifies the different codes that CACSPBR can return. For each return code a brief description of the problem is described, possible methods to resolve the problem and whether a more detailed explanation of the error condition can be found in the server’s output log.

**Table 60: CACSPBR Return Codes**

Return Code	Description	Logged
-102	Invalid buffer version identifier. Either your stored procedure application program did not pass the SQLDA that was passed to your application as the third parameter to CACSPBR, or the SQLDA has been corrupted. Verify that the SQLDA is being passed as the third parameter to CACSPBR. If you are passing the SQLDA as the third parameter, then contact CrossAccess Technical Support.	No
-116	Invalid stored procedure internal identifier. See the description of return code -102 for problem resolution information.	No
-117	Invalid sqln value in the SQLDA. See the description of return code -102 for problem resolution information.	No

**Table 60: CACSPBR Return Codes**

<b>Return Code</b>	<b>Description</b>	<b>Logged</b>
-121	Incorrect number of parameters were passed to CACSPBR. Verify that the correct number of parameters are being passed to CACSPBR. If the correct number of parameters are being passed, contact CrossAccess Technical Support.	No
-122	Invalid APPC function was passed. Valid functions are OPEN, SEND, RECEIVE, and CLOSE.	No
-123	Invalid local LU name was passed to CACSPBR. Probable cause is that the LU name contains multiple sets of LU pooling wildcard characters or that too many wildcard characters were supplied. Verify that a correct local LU name is being passed.	No
-131	Attempt by CACSPBR to register itself with the VTAM LU 6.2 connection handler failed. Review the Server output log for the error being reported.	Yes
-132	Error reported by the VTAM LU 6.2 connection handler while processing the OPEN function. Review the Server output log for the error being reported.	Yes
-141	Error reported by the VTAM LU 6.2 connection handler while processing the SEND function. Review the Server output log for the error being reported.	Yes
-151	Error reported by the VTAM LU 6.2 connection handler while processing a RECEIVE function. Review the Server output log for the error being reported.	Yes
-152	Disconnect being reported when processing the RECEIVE function. If the disconnect is being reported due to an error condition, then the error condition is logged in the Server output. If the disconnect occurred because CACSP62 decided to de-allocate the conversation, then no error is logged.	Maybe
-162	Error reported by the VTAM LU 6.2 connection handler while processing the CLOSE function. Review the Server output log for the error being reported.	Yes

## CACSP62 Abend Codes

As part of the eXadas Stored Procedure implementation, a CICS- based LU6.2 communication program is supplied (CACSP62). This program is needed when a user-written stored procedure program requests communication, using the server, with a CICS system. The communication functions performed by this program are

specialized, designed explicitly to support the eXadas Stored Procedure implementation.

This CICS program executes as a transaction. The transaction is initiated when the user-written stored procedure program tells the Server to OPEN a communication session with a specific CICS system, identifying this program by transaction name. Transaction name is determined by the user site personnel and assigned when the software is installed in the CICS system.

Certain failures may occur within this communication process. The user is notified of failures in this communication processor with standard CICS abends. Each error has been assigned a specific abend code for easier problem determination. Normal CICS abend handling is allowed to produce diagnostic materials as defined by the user site. The abend code format is SPnn where *nn* is replaced by alphanumeric characters. The abend code prefix SP is fixed and not user-configurable. The possible abends that may occur and the causes of these abends are as described in the following table.

**Table 61: CACSP62 Abend Codes**

Code	Description
SP01	The transaction was initiated by a means other than an ALLOCATE from the Server. The transaction is specifically designed to execute as a communications server with a specific protocol used by the eXadas Stored Procedure implementation. Do not attempt to initiate the transaction by any other means.
SP02	An error condition has occurred during an LU6.2 RECEIVE operation. The Server that sent the message has terminated. CICS resources have been freed and this abend was issued to generate a transaction dump. EIB error information from the failing RECEIVE has been captured for diagnostic purposes. If the cause of the failure cannot be independently determined by reviewing the reason the Server terminated, contact CrossAccess Technical Support. You will be asked for the transaction dump and associated output. You will also be asked for Server output.
SP03	An error condition has occurred during an LU6.2 RECEIVE operation. The Server was notified of the error and it issued a DEALLOCATE effectively terminating the communication session. CICS resources have been freed and this abend was issued to generate a transaction dump. EIB error information from the failing receive has been captured for diagnostic purposes. Contact CrossAccess Technical Support. You will be asked for the transaction dump and associated output. You will also be asked for Server output.
SP04	An error condition has occurred during an LU6.2 RECEIVE operation. The Server was notified of the error and it has issued an ISSUE ERROR effectively indicating the communication session should be terminated. The Server was sent an ISSUE ABEND. CICS resources have been freed and this abend was issued to generate a transaction dump. EIB error information from the failing receive has been captured for diagnostic purposes. Contact CrossAccess Technical Support. You will be asked for the transaction dump and associated output. You will also be asked for Server output.

**Table 61: CACSP62 Abend Codes**

Code	Description
SP05	An error condition has occurred during an LU6.2 RECEIVE operation. The Server was notified of the error and it asked for additional information. The attempt to SEND EIB error information to the Server has also failed. The Server was sent an ISSUE ABEND. CICS resources have been freed and this abend was issued to generate a transaction dump. EIB error information from both the failing RECEIVE and the failing SEND has been captured for diagnostic purposes. Contact CrossAccess Technical Support. You will be asked for the transaction dump and associated output. You will also be asked for Server output.
SP06	An LU6.2 SYNCPOINT request has been detected. The eXadas Stored Procedure LU6.2 communications implementation does not support SYNCPOINT processing. The communication partner was sent an ISSUE ABEND. The eXadas Stored Procedure LU6.2 Connection Handler program is designed explicitly to support the eXadas Stored Procedure communications with a Server. Verify the communication partner is a Server. If the cause of the failure cannot be resolved, contact CrossAccess Technical Support. You will be asked for the transaction dump and associated output. You will also be asked for Server output.
SP07	An LU6.2 SYNCPOINT ROLLBACK request has been detected. The eXadas Stored Procedure LU6.2 communications implementation does not support SYNCPOINT ROLLBACK processing. The communication partner was sent an ISSUE ABEND. The eXadas Stored Procedure LU6.2 Connection Handler program is designed explicitly to support the eXadas Stored Procedure communications with a Server. Verify the communication partner is a Server. If the cause of the failure cannot be resolved, contact CrossAccess Technical Support. You will be asked for the transaction dump and associated output. You will also be asked for Server output.
SP08	An LU6.2 ISSUE SIGNAL request has been detected. The eXadas Stored Procedure LU6.2 communications implementation does not support ISSUE SIGNAL processing. The communication partner was sent an ISSUE ABEND. The eXadas Stored Procedure LU6.2 Connection Handler program is designed explicitly to support the eXadas Stored Procedure communications with a Server. Verify the communication partner is a Server. If the cause of the failure cannot be resolved, contact CrossAccess Technical Support. You will be asked for the transaction dump and associated output. You will also be asked for Server output.
SP09	An error condition has occurred during an LU6.2 RECEIVE operation. An illogical condition between incomplete data received and no data received was detected. The Server was sent an ISSUE ABEND. CICS resources have been freed and this abend was issued to generate a transaction dump. EIB error information from the failing receive has been captured for diagnostic purposes. Contact CrossAccess Technical Support. You will be asked for the transaction dump and associated output.

**Table 61: CACSP62 Abend Codes**

Code	Description
SP10	An LU6.2 RECEIVE operation completed normally but was accompanied by a DEALLOCATE indicator. This means the Server is not in RECEIVE mode, thereby preventing the CICS component from returning (SENDing) any processed information to the Server. The CICS resources have been freed and this abend was issued to generate a transaction dump. EIB error information has been captured for diagnostic purposes. If the reason the Server issued a DEALLOCATE cannot be independently determined, contact CrossAccess Technical Support. You will be asked for the transaction dump and associated output. You will also be asked for Server output.
SP11	An LU6.2 RECEIVE operation completed normally but no data was received and the Server has issued a DEALLOCATE. This means the Server is not in RECEIVE mode, thereby preventing the CICS component from returning (SENDing) any processed information to the Server. The CICS resources have been freed and this abend was issued to generate a transaction dump. EIB error information has been captured for diagnostic purposes. If the reason the Server issued a DEALLOCATE cannot be independently determined, contact CrossAccess Technical Support. You will be asked for the transaction dump and associated output. You will also be asked for Server output.
SP12	The user written stored procedure program that executes in CICS receives a COMMAREA that contains the address of (a pointer to) the argument data buffer and the length of that buffer. The buffer cannot be moved or lengthened. The user-written stored procedure program may indicate a shorter argument data buffer is to be returned to the Server by changing the buffer length field in the COMMAREA. This abend was issued to generate a transaction dump. You must request CLOSE to terminate the communication session and release the CICS resources. Modify the user-written stored procedure program to prevent moving the argument data buffer.
SP13	An error condition has occurred during an LU6.2 SEND operation. The Server was sent an ISSUE ABEND. CICS resources have been freed and this abend was issued to generate a transaction dump. EIB error information from the failing send has been captured for diagnostic purposes. If the cause of the failure cannot be independently determined, contact CrossAccess Technical Support. You will be asked for the transaction dump and associated output. You will also be asked for Server output.
SP14	An LU6.2 RECEIVE operation completed normally. The data received is not the eXadas argument data buffer. The format and content of the data is unknown. No further processing can be performed. This abend was issued to generate a transaction dump. You must request CLOSE to terminate the communication session and release the CICS resources. If the reason incorrect data was received cannot be independently determined, contact CrossAccess Technical Support. You will be asked for the transaction dump and associated output. You will also be asked for Server output.

**Table 61: CACSP62 Abend Codes**

Code	Description
SP15	An LU6.2 RECEIVE operation completed normally. The argument data buffer received was not identified correctly. Either the buffer storage was corrupted or a Server did not create this buffer. The format and content of the data are suspect. No further processing can be performed. This abend was issued to generate a transaction dump. You must request CLOSE to terminate the communication session and release the CICS resources. If the reason the buffer is incorrectly identified cannot be independently determined, contact CrossAccess Technical Support. You will be asked for the transaction dump and associated output. You will also be asked for Server output.
SP16	An LU6.2 RECEIVE operation completed normally. The argument data buffer is not compatible with the eXadas Connection Handler transaction program that issued this abend. No further processing can be performed. This abend was issued to generate a transaction dump. You must request CLOSE to terminate the communication session and release the CICS resources. If you have recently upgraded your eXadas product suite, verify all components have been correctly installed. If the reason the buffer is incompatible cannot be independently determined, contact CrossAccess Technical Support. You will be asked for the transaction dump and associated output. You will also be asked for Server output.

## CA-DATACOM/DB Interface Description

The CACTDCOM interface module allows a stored procedure application program to access CA-DATACOM/DB data locally (from within the eXadas server address space). Access is performed using native CA-DATACOM/DB commands and control blocks. The CACTDCOM interface requires a stored procedure application program to first open a User Requirements Table. Once the URT is open, any series of CA-DATACOM/DB commands may be issued. When the stored procedure application program completes processing, the URT must be closed. Using the CACTDCOM interface allows the stored procedure application program to safely access and update CA-DATACOM/DB data, and supports transaction isolation from other stored procedure application programs that may also be accessing CA-DATACOM/DB data.

Using the CACTDCOM interface from a stored procedure application program is closely equivalent to accessing or updating CA-DATACOM/DB databases using a batch program. Although the interface macro DBURINF is never included when generating a User Requirements Table for use by the eXadas suite of products, there remain some similarities to a batch application. The URT name is available from the RUN OPTIONS statement in your Stored Procedure definition rather than from the LOADNAM= parameter in the DBURINF macro. The requirement that your stored procedure application program open and close the URT is the

same as if you had coded the OPEN=USER parameter in the DBURINF macro for your batch application.

The CACTDCOM interface requires the Datacom Initialization Service be active within the eXadas server. CACTDCOM interfaces with the Datacom Initialization Service to connect to and communicate with the CA-DATACOM/DB Multi-User Facility. This allows CACTDCOM to log errors. Additional diagnostic information is also available when an error occurs, if the Datacom Initialization Service trace level is set to a value less than three. See [Chapter 5, “Server Setup for CA-DATACOM/DB”](#) for instructions on how to define a Datacom Initialization Service.

## CACTDCOM Interface Description

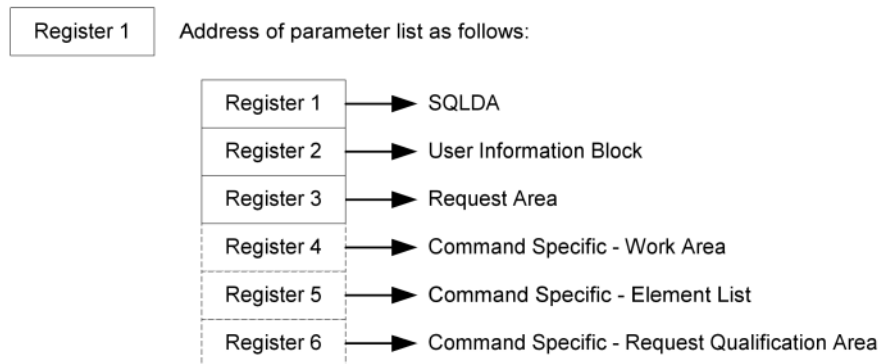
The CACTDCOM interface is distributed as a separate load module that may be dynamically called or statically linked into your stored procedure application program. See [“Compiling and Linking Applications That Use CACTDCOM,” on page 315](#), for an example of how to statically link CACTDCOM into your stored procedure application program.

The CACTDCOM interface is called using a parameter list where the first parameter passed in the list is the SQLDA argument passed to the stored procedure application program. The remaining entries in the parameter list are the normal parameters used to access CA-DATACOM/DB data. Each database operation requires from two to five parameters. For a description of the parameters required, an explanation of how they are used and what information they should contain for the call (or will contain upon return), reference the *Commands* chapter in the *CA-DATACOM/DB Programmer Guide*.

[Figure 44: “Parameters Passed to CACTDCOM”](#) shows how parameters need to be passed to CACTDCOM. CACTDCOM must be called using variable-argument list calling conventions. That is, the high-order bit of the last parameter must be set to one. This is the standard calling convention for most high-level languages, including COBOL. CACTDCOM is linked as an AMODE(31) RMODE(ANY)

application. All addresses passed to CACTDCOM must be passed in 31-bit addressing mode.

**Figure 44: Parameters Passed to CACTDCOM**



As indicated by the diagram above, parameters 4, 5, and 6 are conditional, based upon the type of CA-DATACOM/DB command specified in the Request Area.

The first call issued by any stored procedure application program must be to OPEN the User Requirements Table. Opening the URT allows any table specified in the URT to be accessed and those tables defined with UPDATE=YES may also be updated. The following code example shows how to issue the OPEN command in a COBOL stored procedure application program. Included below is the portion of the stored procedures DATA DIVISION and the LINKAGE SECTION that is referenced in the executable code and the executable code to prepare for and issue the OPEN command.

All examples of code in this section of the documentation are hard-coded to process the table named "DEMO-DEM-POH". This table is delivered as part of the installation process for CA-DATACOM/DB. Also note, in this example, the User Requirements Table that is specified in the RUN OPTIONS of the cataloged Stored Procedure will be opened because no override \_DATACOM keyword was supplied.

```

WORKING-STORAGE SECTION.
01  USER-INFO-BLOCK                                PIC X(32).

01  POH-REQUEST-AREA.
    02  POH-REQ-COMMAND                            PIC X(5).
    02  POH-REQ-TABLE-NAME                        PIC X(3) VALUE "POH".
    02  FILLER                                    PIC X(5).
    02  POH-REQ-RETURN-CODE                      PIC X(2).
    02  POH-REQ-INTRNL-RTNCD                    PIC X(1).
    .      .      .      .      .
    .      .      .      .      .

LINKAGE SECTION.
01  SQLDA-DATA.
    05  ARG-SQLDAID                              PIC X(8).
    05  ARG-SQLDABC                              PIC 9(8) COMP.
    05  ARG-SQLN                                  PIC 9(4) COMP.
    .      .      .
    .      .      .
  
```



```

* BUILD REQUEST AREA AND OPEN URT
9300-OPEN-URT.

        MOVE "MYPROGID"      TO USER-INFO-BLOCK.
        MOVE "OPEN"         TO POH-REQ-COMMAND.
        MOVE SPACES         TO POH-REQ-RETURN-CODE.
        MOVE 0              TO POH-REQ-INTRNL-RTNCD.

* IF YOUR URT CONTAINS MULTIPLE "POH" TABLES, SUPPLY THE
* DATABASE ID WHERE TABLE "DEMO-DEM-POH" IS INSTALLED.
* THIS IS NORMALLY DATABASE 1 (DBID=00001).
* MOVE 1                    TO POH-REQ-DATABASE-ID.

* TO OVERRIDE THE URT IN THE META DATA "RUN OPTIONS",
* SUPPLY A URT NAME IN THE WORK AREA AND PASS THE WORK AREA
* ADDRESS AS THE FOURTH PARAMETER ON THE CALL THAT FOLLOWS.
* MOVE "_DATACOM(URTNAME)" TO POH-WORK-AREA.

DISPLAY "CACSPDC1 CALLING CACTDCOM TO OPEN URT."
        UPON CONSOLE.
CALL "CACTDCOM" USING SQLDA-DATA,
                    USER-INFO-BLOCK,
                    POH-REQUEST-AREA.
IF POH-REQ-RETURN-CODE NOT EQUAL SPACES
    PERFORM 9900-DISPLAY-RC
END-IF.

IF RETURN-CODE NOT EQUAL ZEROS
    DISPLAY "CACSPDC1 OPEN URT ERROR. RC=" RETURN-CODE
        UPON CONSOLE
    GOBACK
END-IF.

```

Once the URT is successfully opened, the stored procedure application program can issue any CA-DATACOM/DB command against the database. The following snippet of code shows how to issue the ADDIT command for table "DEMO-DEM-POH" in a COBOL stored procedure application program. Included below is a portion of the stored procedures DATA DIVISION and the LINKAGE SECTION that is referenced in the executable code and the executable code to prepare for and issue the ADDIT command:

```

WORKING-STORAGE SECTION.
01  USER-INFO-BLOCK                PIC X(32) .

01  POH-REQUEST-AREA.
    02  POH-REQ-COMMAND              PIC X(5) .
    02  POH-REQ-TABLE-NAME          PIC X(3) VALUE "POH" .
    02  FILLER                      PIC X(5) .
    02  POH-REQ-RETURN-CODE        PIC X(2) .
    02  POH-REQ-INTRNL-RTNCD       PIC X(1) .
    :  :  :  :  :  :  :  :  :  :
    :  :  :  :  :  :  :  :  :  :

01  POH-WORK-AREA                  PIC X(20) .
01  POH-RECORD REDEFINES POH-WORK-AREA.
    02  POH-PO                      PIC X(5) .
    02  POH-LI                      PIC X(3) .
    02  POH-RECORD-UPDATE.
        03  POH-VENDR                PIC X(3) .
        03  POH-TPVAL                PIC X(8) .
        03  POH-LATE                 PIC X .

```

```

01 POH-ELEMENT-LIST.
  02 POH-ELM1                PIC X(5) VALUE "PO".
  02 POH-SEC-CD1             PIC X   VALUE " ".
  02 POH-ELM2                PIC X(5) VALUE "LI".
  02 POH-SEC-CD2             PIC X   VALUE " ".
  02 POH-ELEMENT-LIST-UPDATE.
    03 POH-ELM3              PIC X(5) VALUE "VENDR".
    03 POH-SEC-CD3           PIC X   VALUE " ".
    03 POH-ELM4              PIC X(5) VALUE "TPVAL".
    03 POH-SEC-CD4           PIC X   VALUE " ".
    03 POH-ELM5              PIC X(5) VALUE "LATE".
    03 POH-SEC-CD5           PIC X   VALUE " ".
    03 END-OF-ELEMENTS       PIC X(5) VALUE SPACES.

LINKAGE SECTION.
01 SQLDA-DATA.
  05 ARG-SQLDAID             PIC X(8).
  05 ARG-SQLDABC             PIC 9(8) COMP.
  05 ARG-SQLN                PIC 9(4) COMP.
  .   .   .                 .   .   .
  .   .   .                 .   .   .

* BUILD CONTROL BLOCKS AND ISSUE ADDIT COMMAND TO INSERT RECORD
2200-ISSUE-ADDIT.

  MOVE "ADDIT"                TO POH-REQ-COMMAND.
  MOVE SPACES                  TO POH-REQ-RETURN-CODE.
  MOVE 0                       TO POH-REQ-INTRNL-RTNCD.
  DISPLAY "CACSPDC1 CALLING CACTDCOM TO ADDIT." UPON CONSOLE.
  CALL "CACTDCOM" USING SQLDA-DATA,
                        USER-INFO-BLOCK,
                        POH-REQUEST-AREA,
                        POH-RECORD,
                        POH-ELEMENT-LIST.
IF POH-REQ-RETURN-CODE NOT EQUAL SPACES
  PERFORM 9900-DISPLAY-RC
END-IF.

IF RETURN-CODE NOT EQUAL ZEROS
  PERFORM 10000-ERROR-CLOSE-URT
  DISPLAY "CACSPDC1 ADDIT ERROR. RC=" RETURN-CODE
  UPON CONSOLE
  GOBACK
END-IF.

```

Once database modification has been completed successfully, issue a COMMIT command. If the database modification is determined to be unsuccessful or incorrect, issue a ROLBK command to reverse all database modifications done since the last COMMIT or ROLBK command. If the stored procedure application program has issued neither COMMIT nor ROLBK, all database modifications done since the OPEN command are reversed. Issuing a CLOSE command executes an implied COMMIT, so the actual COMMIT command may be bypassed. Follow your site standards and procedures regarding the use of explicit or implicit COMMIT processing.

When the desired processing is completed, the last call issued by any stored procedure application program must be to CLOSE the User Requirements Table. The following code sample shows how to issue the CLOSE command in a COBOL stored procedure application program. Included below is a portion of the

stored procedures DATA DIVISION and the LINKAGE SECTION that is referenced in the executable code and the executable code to prepare for and issue the CLOSE command:

```

WORKING-STORAGE SECTION.
01  USER-INFO-BLOCK                               PIC X(32).

01  POH-REQUEST-AREA.
    02  POH-REQ-COMMAND                           PIC X(5).
    02  POH-REQ-TABLE-NAME                         PIC X(3) VALUE "POH".
    02  FILLER                                     PIC X(5).
    02  POH-REQ-RETURN-CODE                       PIC X(2).
    02  POH-REQ-INTRNL-RTNCD                      PIC X(1).
    .      .      .      .      .
    .      .      .      .      .

LINKAGE SECTION.
02  SQLDA-DATA.
    05  ARG-SQLDAID                               PIC X(8).
    05  ARG-SQLDABC                               PIC 9(8) COMP.
    05  ARG-SQLN                                  PIC 9(4) COMP.
    .      .      .
    .      .      .

* BUILD REQUEST AREA AND CLOSE URT
9500-CLOSE-URT.

    MOVE "CLOSE"                                TO POH-REQ-COMMAND.
    MOVE SPACES                                  TO POH-REQ-RETURN-CODE.
    MOVE 0                                       TO POH-REQ-INTRNL-RTNCD.
* NO NEED FOR URT NAME HERE. IT IS REMEMBERED FROM THE OPEN.

    DISPLAY "CACSPDC1 CALLING CACTDCOM TO CLOSE URT."
    UPON CONSOLE.
    CALL "CACTDCOM" USING SQLDA-DATA,
                                USER-INFO-BLOCK,
                                POH-REQUEST-AREA.
IF POH-REQ-RETURN-CODE NOT EQUAL SPACES
PERFORM 9900-DISPLAY-RC
END-IF.

IF RETURN-CODE NOT EQUAL ZEROS
    DISPLAY "CACSPDC1 CLOSE URT ERROR. RC=" RETURN-CODE
    UPON CONSOLE
    GOBACK
END-IF.

```

## Compiling and Linking Applications That Use CACTDCOM

eXadas does not supply sample JCL to compile and link a stored procedure application program that uses the CACTDCOM interface. Your stored procedure application program is compiled (or assembled) using your site standard procedures. Linking the CACTDCOM interface, again using your site standard

procedures, requires only that you include the following control statement in your link edit input stream:

```
INCLUDE LOAD(CACTDCOM)
```

where `LOAD` is the name of a DD statement supplied in your link edit JCL to identify the location of the `CACTDCOM` load module. This DD statement will probably identify your eXadas installation load library.

## CACTDCOM Return Codes

`CACTDCOM` returns a zero return code in normal situations. If `CACTDCOM` detects an error, a non-zero return code is returned. Database errors reported by `CA-DATACOM/DB` are returned directly to the stored procedure application in the Request Area and are not intercepted or interpreted by the `CACTDCOM` interface. There are a number of different error return codes. The following table identifies the possible return code values and what, if any, information can be found in the eXadas server's output log when one of these errors is detected.

**NOTE:** Return code values used by the `CACTDCOM` interface are between 101 and 199. To eliminate any confusion regarding the source of an error code, the stored procedure application program may choose to use return codes starting at the number 200 or above.

**Table 62: CACTDCOM Return Code Information**

Value	Description	Log Information
101	An invalid number of parameters were passed to <code>CACTDCOM</code> . Additional calls can be issued to <code>CACTDCOM</code> . This error should occur only during development of the stored procedure application program.	Message 5701813 (0x005700B5) is generated.
102	The first parameter passed to <code>CACTDCOM</code> cannot be identified as an <code>SQLDA</code> or no parameter was passed to <code>CACTDCOM</code> . Additional calls to <code>CACTDCOM</code> should not be issued. This error should occur only during development of the stored procedure application program.	None. The information necessary to issue log calls is not available.
103	The identifier in the internal control block used to manage stored procedure processing by the <code>CACTDCOM</code> interface has been corrupted. Additional calls to <code>CACTDCOM</code> should not be issued.  This is an internal error. If encountered, contact CrossAccess Technical Support.	None. The information necessary to issue log calls is not available.

**Table 62: CACTDCOM Return Code Information**

Value	Description	Log Information
104	<p>The length in the internal control block used to manage stored procedure processing by the CACTDCOM interface has been corrupted. Additional calls to CACTDCOM should not be issued.</p> <p>This is an internal error. If encountered, contact CrossAccess Technical Support.</p>	None. The information necessary to issue log calls is not available.
105	<p>The buffer version in the internal control block used to manage stored procedure processing by the CACTDCOM interface is not correct. Additional calls to CACTDCOM should not be issued. This indicates stored procedures of a prior release have been installed with the current release of eXadas. Any prior stored procedure applications must be re-linked using the new CACTDCOM interface.</p>	None. The information necessary to issue log calls is not available.
106	<p>CACTDCOM environment not properly initialized. Either the Stored Procedure Data Savant control block pointer is zero or the Datacom Service anchor pointer is corrupted. Additional calls to CACTDCOM should not be issued.</p> <p>This is an internal error. If encountered, contact CrossAccess Technical Support.</p>	<p>1. None. The information necessary to issue log calls is not available if the control block pointer is zero, or</p> <p>2. Message 5701890 (0x00570102) is generated if the anchor pointer is corrupted.</p>
107	<p>CACTDCOM environment not initialized. Either the global system anchor pointer is zero or the Datacom Service anchor pointer is zero. Additional calls to CACTDCOM should not be issued.</p> <p>This is an internal error. If encountered, contact CrossAccess Technical Support.</p>	<p>1. None. The information necessary to issue log calls is not available if the global system anchor pointer is zero, or</p> <p>2. Message 5701889 (0x00570101) is generated if the Datacom Service anchor pointer is zero.</p>
108	<p>The stored procedure application program is attempting to issue a second OPEN command. Only one User Requirements Table can be open at a time. Additional calls can be issued to CACTDCOM. This error should occur only during development of the stored procedure application program.</p>	Message 5701816 (0x005700B8) is generated.

**Table 62: CACTDCOM Return Code Information**

Value	Description	Log Information
109	CACTDCOM was unable to allocate memory for internal control blocks. Additional calls to CACTDCOM should not be issued. This error should only be received when the eXadas server is not properly configured.	Message 5701793 (0x005700A1) is generated.
110	No User Requirements Table name was provided to CACTDCOM. Either the RUN OPTIONS statement from the catalog or the programmatic override did not contain the keyword _DATACOM, or the _DATACOM keyword was not followed immediately by a (urtname) clause. Additional calls to CACTDCOM should not be issued. This error should occur only during development of the stored procedure application program.	Message 5701814 (0x005700B6) is generated.
111	The User Requirements Table name provided to CACTDCOM was less than one character or greater than eight characters in length. Additional calls to CACTDCOM should not be issued. This error should occur only during development of the stored procedure application program.	Message 5701815 (0x005700B7) is generated.
112	The User Requirements Table program could not be loaded. Additional calls to CACTDCOM should not be issued. The URT program must be in a load library that is included in the STEPLIB concatenation of the eXadas server.	Message 5701817 (0x005700B9) is generated.
113	The User Requirements Table program that was loaded does not appear to be a known format. Additional calls to CACTDCOM should not be issued. A portion of the URT is dumped in binary to the eXadas server log when the trace level is set to 4 or less. Review the URT content and determine if it is valid. Changes in the format of User Requirement Tables require code changes in CACTDCOM. Contact CrossAccess Corporation Technical Support if you believe the CACTDCOM interface requires code changes.	Message 5701818 (0x005700BA) is generated.
114	An attempt to connect with the eXadas Datacom Initialization Service has failed. One of the error messages shown in the column to the right has been logged. Additional calls to CACTDCOM should not be issued. This probably indicates the Datacom Initialization Service module is not active.	1. Message 5701897 (0x00570109) is generated 2. Message 5701903 (0x0057010F) is generated 3. Message 5701905 (0x00570111) is generated

**Table 62: CACTDCOM Return Code Information**

Value	Description	Log Information
115	<p>An error has occurred while attempting to send an OPEN command to CA-DATACOM/DB. Additional calls to CACTDCOM should not be issued. The User Information Block (UIB) and the Request Area (RA) passed to CACTDCOM are dumped in binary to the eXadas server log when the trace level is set to 2 or less. Review the control blocks and determine if they contain valid data. Depending upon the eXadas return code information that is logged, this may be an internal error that should not occur or it may be an error that could occur only during development of the stored procedure application program.</p>	Raw eXadas system return code information is logged.
116	<p>Resources required to communicate with CA-DATACOM/DB are not available. All retries have been exhausted. Additional calls to CACTDCOM should not be issued. The CA-DATACOM/DB control blocks passed to CACTDCOM are dumped in binary to the eXadas server log when the trace level is set to 2 or less.</p> <p>This is an internal error. If encountered, contact CrossAccess Technical Support.</p>	Message 5701819 (0x005700BB) is generated
117	<p>The stored procedure application program has called CACTDCOM with a database command without first OPENing the URT. Additional calls can be issued to CACTDCOM. This error should occur only during development of the stored procedure application program.</p>	Message 5701820 (0x005700BC) is generated
118	<p>A CACTDCOM interface environment pointer is zero. Additional calls to CACTDCOM should not be issued.</p> <p>This is an internal error. If encountered, contact CrossAccess Technical Support.</p>	Message 5701821 (0x005700BD) is generated
119	<p>An error has occurred while attempting to send a CLOSE command to CA-DATACOM/DB. Additional calls to CACTDCOM should not be issued.</p> <p>The User Information Block (UIB) and the Request Area (RA) passed to CACTDCOM are dumped in binary to the eXadas server log when the trace level is set to 2 or less. Review the control blocks and determine if they contain valid data. Depending upon the eXadas return code information that is logged, this may be an internal error that should not occur or it may be an error that could occur only during development of the stored procedure application program.</p>	Raw eXadas system return code information is logged.
120	<p>The CACTDCOM interface was processing a command other than OPEN or CLOSE when it received a signal from the eXadas Query Processor to immediately terminate processing. The Query Processor is probably being stopped.</p>	Message 5701894 (0x00570106) is generated

**Table 62: CACTDCOM Return Code Information**

Value	Description	Log Information
121	An error has occurred while attempting to send a command other than OPEN or CLOSE to CA-DATACOM/DB. Additional calls to CACTDCOM may be attempted to COMMIT or ROLBK as may be required by the stored procedure application program. Success of any subsequent call is dependent upon the type of error that occurred previously. The CA-DATACOM/DB control blocks passed to CACTDCOM are dumped in binary to the eXadas server log when the trace level is set to 2 or less.  This is an internal error. If encountered, contact CrossAccess Technical Support.	Raw eXadas system return code information is logged.
122	The User Requirements Table contains at least one table enabled for update processing. When the OPEN command was sent by the stored procedure application program, the Internal Return Code field in the Request Area was coded with the letter N, indicating no update processing was to be allowed. The stored procedure application program has sent an ADDIT, DELET or UPDAT command which has been rejected. Additional calls can be issued to CACTDCOM. This error should occur only during development of the stored procedure application program.	Message 5701822 (0x005700BE) is generated

## IMS DRA Interface Description

The CACTDRA interface module allows a stored procedure application program to access IMS data locally (within the Server address space) using the DRA interface. The CACTDRA interface allows a stored procedure application program to schedule a PSB, issue a series of standard DL/I calls, and then un-schedule the PSB. The CACTDRA interface allows full function IMS databases, for example, HDAM or HIDAM, and Fast Path DEDB databases to be accessed/updated.

Using the CACTDRA interface allows the stored procedure application program to safely access/update IMS data and supports transaction isolation from other stored procedure application programs that may also be accessing IMS data. Using the CACTDRA interface the stored procedure application program can act very similarly to an IMS/DC on-line transaction program.

The CACTDRA interface requires that the IMS DRA Initialization service be active within the Server. CACTDRA interfaces with the IMS DRA Initialization service. This allows CACTDRA to log errors and, if the IMS DRA Initialization services trace level is set to a value less than three, the actual DL/I calls issued by the stored procedure application program calling CACTDRA. See [Chapter 3](#),



“[Server Setup for IMS Access](#),” for instructions on how to define an IMS DRA Initialization service.

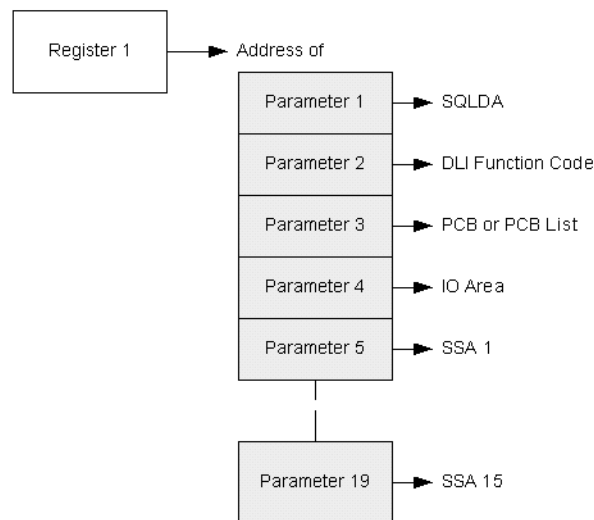
## CACTDRA Interface Description

The CACTDRA is distributed as a separate load module that may be dynamically called or statically linked into your stored procedure application program. See “[Compiling and Linking Applications That Use CACTDRA](#),” on page 323, for an example of how to statically link CACTDRA into your stored procedure application program.

The CACTDRA interface uses calling conventions that are very similar to the way a normal program accesses IMS data. The primary difference is that the first parameter passed to CACTDRA must be the SQLDA argument list passed to the stored procedure application program.

[Figure 45: “Parameters Passed to CACTDRA,” on page 321](#), shows how parameters need to be passed to CACTDRA. Additionally, CACTDRA must be called using variable-argument list calling conventions, the high-order bit of the last parameter must be set to one. This is the standard calling convention for most high-level languages, for example COBOL. Additionally, CACTDRA is linked as an AMODE(31) RMODE(ANY) application. All addresses passed to CACTDRA must be passed in 31-bit addressing mode.

**Figure 45: Parameters Passed to CACTDRA**



The actual number of parameters that need to be passed to CACTDRA depend on the type of DL/I function being issued and the structure of the database being

accessed. The first call that must be issued to CACTDRA is a CICS-like SCHD call that schedules a PSB. This call differs slightly from a normal DL/I call in that:

- the name of the PSB is passed in the I/O area,
- CACTDRA returns the address of the list of PCBs that are defined within the PSB.

One of these PCBs is passed on subsequent calls to CACTDRA in order to access/update IMS data. The I/O PCB can also be passed to issue checkpoint or rollback calls.

The following shows how to issue the SCHD call in a COBOL stored procedure application program. It shows the stored procedures DATA DIVISION, part of the LINKAGE SECTION, and shows how to prepare for and issue the SCHD call:

```

WORKING-STORAGE SECTION.
01  PSB-NAME                PIC X(8)
                                VALUE "DFSSAM09".
01  DLI-FUNC-CODE           PIC X(4).
01  DRA-PCB-LIST           POINTER.
01  DRA-PCB-LIST           POINTER.
   10  SSA-SEG-NAME         PIC X(9)
                                VALUE "PARTROOT".
01  IO-AREA.
   10  PART-KEY             PIC X(17).
   10  DESCRIPT            PIC X(20).
   10  FILLER              PIC X(13).
LINKAGE SECTION.
COPY CACSPSDA.
COPY CACSPSCA.
01  PCB-LIST.
   05  IO-PCB              POINTER.
   05  DB-PCB1            POINTER.
   05  DB-PCB2            POINTER.
01  RET-PCB.
   10  PCB-DBD             PIC X(8).
   10  PCB-SEG-LVL        PIC 99.
   10  PCB-STATUS-CODE    PIC XX.
   10  PCB-PROCOPT        PIC X(4).
   10  FILLER             PIC X(4).
   10  PCB-SEG-NAME       PIC X(8).
   10  PCB-KFBA-LEN       PIC 9(8) COMP.
   10  PCB-SENSEGS        PIC 9(8) COMP.
   10  PCB-KFBA           PIC X(33).

PROCEDURE DIVISION
                                USING ARG-DATA, ARG-SQLCA.
MOVE "SCHD" TO DLI-FUNC-CODE.
CALL "CACTDRA" USING ARG-DATA,
                                DLI FUNC CODE,
                                DRA-PCB-LIST
                                PSB-NAME.
IF RETURN-CODE NOT EQUAL ZEROS
ERROR OCCURED
GOBACK
ELSE
SET ADDRESS OF PCB-LIST
TO DRA-PCB-LIST.

```

In this example, the DFSSAM09 IMS sample PSB is being scheduled. This PSB allows updates to be performed to all segments of the DI2IPART IMS sample database. Once the PSB is successfully scheduled, the stored procedure application program can obtain addressability to one or more of the PCBs in the PSB and issue normal DL/I calls against any of the PCBs that are available.

The following figure is a singlet of code from a COBOL program that shows how to insert a new root segment in the DI2IPART database in a COBOL program.

Like normal DL/I calls, up to 15 SSAs can be passed to CACTDRA. In the same manner, DL/I calls must be issued in the proper sequence. For example, to update a segment a GHU call must be issued first, followed by a REPL call.

Once all of the access/update DL/I calls have been issued, a CHKP does not have to be issued. When the PSB is unscheduled, a CHKP call is automatically issued. If the stored procedure application program decides that any updates should not be applied a ROLLBACK call can be issued. The stored procedure application program should not issue a ROLLBACK call that causes an abend. If a

ROLLBACK call is issued, the Query Processor task servicing the stored procedure application program becomes unusable.

The following is sample code for an ISRT call:

```

SET ADDRESS OF RET-PCB TO DB-PCB1.

MOVE SPACES TO IO-AREA.
* GET ADDRESSABILITY TO SQLDA PARAMETERS AND INITIALIZE
* IO-AREA FOR INSERT
MOVE "ISRT" TO DLI-FUNC-CODE.

CALL "CACTDRA" USING ARG-DATA,
                   DLI-FUNC-CODE,
                   RET-PCB,
                   IO-AREA,
                   SSA.
IF RETURN-CODE NOT EQUAL ZEROS
  MOVE RETURN-CODE TO ORIG-RC
  ERROR HAS OCCURED
ELSE
  IF PCB-STATUS-CODE NOT = SPACES
    ERROR REPORTED BY IMS.

```

Once the final DL/I call has been issued CACTDRA must be called a final time to un-schedule the PSB. This is done using a CICS-like TERM call. The following shows sample code in a COBOL program to un-schedule the DFSSAM09 PSB:

```

MOVE "TERM" TO DLI-FUNC-CODE.
CALL "CACTDRA" USING ARG-DATA,
                   DLI-FUNC-CODE.
IF RETURN-CODE NOT EQUAL ZEROS
  DISPLAY "TERM CALL RC " RETURN-CODE.
GOBACK.

```

## Compiling and Linking Applications That Use CACTDRA

eXadas does not supply sample JCL to compile and link a stored procedure application program that uses the CACTDRA interface. The following instructions describe how to modify the CICS CACSPCCC sample compile and link JCL to include the CACTDRA interface instead of CACSPBR.

Begin by following the instructions in [“Compiling and Linking Applications that Use CACSPBR,” on page 304](#), Before submitting the job, modify the statement:

```
INCLUDE LOAD(CACSPBR)
```

to:

```
INCLUDE LOAD(CACTDRA)
```

If your stored procedure application program needs to both access/update IMS data and invoke a CICS application program, then include both of the above statements in the link step.

## CACTDRA Return Codes

CACTDRA returns a zero return code in normal situations. If CACTDRA detects an error, or an error is reported by DRA, a non-zero return code is returned. There are 11 different error return codes. The following table identifies the possible return code values and what, if any, information can be found in the Server's output log when one of these errors is detected.

**NOTE:** Return code values between 1 and 100 are reserved for use by the CACTDRA interface. The application return codes must be higher than the reserved error return codes.

**Table 63: CACTDRA Return Code Information**

Value	Description	Log Information
1	An invalid number of parameters were passed to CACTDRA. Additional calls can be issued to CACTDRA. This error should only be received during development of the stored procedure application program.	Message 5701825 (0x005700C1) is generated.
2	CACTDRA was unable to allocate memory for internal control blocks. Additional calls to CACTDRA should not be issued. This error should only be received when the Server is not properly configured.	Message 5701793 (0x005700A1) is generated.
3	IMS DRA Initialization service not active. Additional calls to CACTDRA should not be issued. This error should only be received when the Server is not properly configured.	Message 5701717 (0x00570055) is generated.
4	CACTDRA environment not properly initialized. Additional calls to CACTDRA should not be issued. This is an internal error that should never occur.	None. The information necessary to issue log calls is not available.
5	The requested PSB could not be scheduled. The stored procedure application program can try to schedule another PSB. This error can occur during development of the stored procedure application program. It can also be encountered in production situations when the Server has not been properly configured (for example, not enough DRA threads). This error can also be reported when the IMS Stage 1 gen. was not set up properly for the PSB scheduled.	Message 5701708 (0x0057004C) is generated.

**Table 63: CACTDRA Return Code Information**

Value	Description	Log Information
6	DL/I call failed. Additional calls can be issued to CACTDRA. This error should only be received during development of the stored procedure application program.	Message 5701715 (0x00570053) is generated.
7	An error occurred during TERM processing. Additional calls to CACTDRA should not be issued. This error should generally only be received if something has happened to IMS.	Message 5701711 (0x0057004F) is generated.
8	The first parameter passed to CACTDRA is not the SQLDA or the SQLDA has been corrupted. Additional calls to CACTDRA should not be issued. This error should only be received during development of the stored procedure application program.	None. The information necessary to issue log calls is not available.
9	The SQLDA has been corrupted. Additional calls to CACTDRA should not be issued. This is an internal error that should never be encountered.	None. The information necessary to issue log calls is not available.
10	The application issued a SCHED call, but a PSB had already been scheduled. This error should only be received during development of the stored procedure application program.	Message 5701826 (0x005700C2) is generated.
11	The application issued a standard DL/I or a TERM call, but no PSB was scheduled. This error should only be received during development of the stored procedure application program.	Message 5701827 (0x005700C3) is generated.

## Invoking Existing IMS Transactions

To execute existing IMS transactions from a stored procedure, the stored procedure application uses an APPC/MVS interface instead of the CACTDRA interface. APPC/MVS is used to communicate with APPC/IMS, which in turn is responsible for scheduling the requested IMS transaction and returning the output message(s) generated by the IMS transaction to the calling APPC/MVS application program (the stored procedure application program).

The remainder of this section discusses the following topics:

- [“APPC/IMS Overview,” on page 326,](#)
- [“APPC/MVS Overview,” on page 327,](#)

- [“Configuring APPC/IMS and APPC/MVS,”](#) on page 327,
- [“Application Design Requirements,”](#) on page 327,
- [“Stored Procedure Limitations,”](#) on page 328,
- [“Testing APPC/MVS Stored Procedures,”](#) on page 329, and
- [“Sample Stored Procedures,”](#) on page 330.

## APPC/IMS Overview

In IMS Version 4, IBM added the APPC/IMS interface into the IMS Transaction Manager product. The IMS Transaction Manager is normally implemented as an IMS DB/DC sub-system, though it can also be implemented as an IMS DC sub-system.

APPC/IMS supports interfacing with three types of IMS application programs:

- **Standard DL/I Applications**—these are existing IMS applications that are unaware that they are not communicating with a LU 2 terminal. APPC/IMS converts the APPC data streams into the appropriate input messages, sends them to the IMS transaction, waits for an output message from the IMS application, and then sends the output message to the invoking application as an LU 6.2 data stream. In most instances, the existing IMS transaction requires no modifications.
- **Modified Standard DL/I Applications**—these are existing IMS application that have been modified to issue CPI Communications calls, as well as normal DL/I calls that the application initially issued.
- **CPI Communication Driven Applications**—these are new IMS applications that use CPI Communications calls to communicate with the partner program. They participate in the two-phase commit process by issuing SSRCMIT or SSRBACK CPI calls. These types of applications can issue database-related DL/I calls to access and update full-function, DEBD, MSDB, and DB2 databases.

The *IMS Vx Application Programming: Design Guide, Chapter 7: Application Design for APPC* contains a good overview of how these types of application are designed. The section titled “LU 6.2 Partner Program Design” contains diagrams of the LU 6.2 message flows for the different types of IMS transactions that can be interfaced with, and additional message flows when these applications are under sync-point control. Also included is an integrity table that shows the differences between using and not using sync-point conversations.

---

## APPC/MVS Overview

APPC/MVS is an extension to APPC/VTAM that allows MVS/ESA applications to use the full capabilities of LU 6.2. APPC/MVS provides a set of high-level language callable services that allows applications using these APIs to communicate with other applications through the communications protocols provided by the SNA network.

The APPC/MVS API has combined several CPI calls into single APPC/MVS API calls, and allows for state transitions that normally require individual CPI calls. APPC/MVS allows applications to be abstracted from the network definitions by using such things as symbolic destination names and selection of default out-bound LU's.

The *OS/390 Vx MVS Writing TPs for APPC/MVS* manual describes how to write APPC/MVS applications and provides reference information on the APPC/MVS APIs that are available. Depending on the version of APPC/MVS installed, some of these APIs have different names and parameter lists.

## Configuring APPC/IMS and APPC/MVS

Setting up an environment for a stored procedure (or any application program for that matter) requires configuration changes to IMS, additional VTAM definitions, and the installation and configuration of APPC/MVS.

## Application Design Requirements

To design a stored procedure that invokes an existing IMS transaction, you do not need to know the business logic that is implemented by the IMS transaction, but you do need to know the message flow and input/output message formats that the IMS transaction is expecting to receive.

Typically, the IMS transaction will use MFS, so you should use the MID and MOD definitions to determine the input and output message formats. If the IMS transaction is implemented in COBOL, the COBOL program contains copy books or data structures that the IMS transaction uses.

If you use MFS MID/MOD definitions to identify the input and output messages formats that the transaction is using, you need to define a parameter for each unique input/output field defined in the MID and the MOD. The parameter length is for the length of the MID/MOD field, and the data type is usually CHAR. Use the following guidelines to determine what type of parameter should be used in the CREATE PROCEDURE definitions:

- Identify as INPUT parameters fields that only appear in the MID,
- Identify as OUTPUT parameters fields that only appear in the MOD, and

- Identify as INOUT parameters fields that occur in both the MID and the MOD.

While developing stored procedures, you need to take into account that the input and output message formats that are sent to APPC/IMS do not exactly match the formats that the IMS transaction is expecting. The standard format for an IMS input message is:

- LL—length of the message, including LL and ZZ
- ZZ—zeros
- Data—the input fields the transactions is expecting.

APPC communications use unmapped conversations, so the send message format is:

- APPC LL
- IMS LL
- Data

APPC/IMS automatically inserts the IMS transaction code at the beginning of the data portion of the message. This may make the invocation of IMS transactions that do not expect the transaction code at the beginning of the input message impossible without modifications to the existing IMS transaction. This should not occur if the transaction was designed using the IBM recommendation that all input messages begin with the 8-byte transaction code.

The output messages generated by an IMS transaction have the same format as the input messages. However, APPC/IMS strips off the LL and ZZ fields, so these do not need to be defined in the received (output message) definition—just the APPC LL field.

## Stored Procedure Limitations

Since stored procedures can only return a limited amount of information, it is recommended that stored procedures invoking IMS transactions be created to perform updates or to return a single screen of data. A stored procedure can be developed that invokes an IMS conversational transaction that returns multiple screens of data.

When a conversation is allocated, an 8-byte conversation ID is returned. This must be used in subsequent APPC/MVS calls. According to the APPC/MVS documentation, the conversation ID is associated with the address space and can be used by any TCB in the address space until the conversation is de-allocated.

Therefore, if an additional input/output parameter is added to the stored procedure definition for the conversation ID, it is possible to write a stored procedure that could be called multiple times, each time returning one or more screens of data. If this approach is taken, the input/output conversation ID parameter should be



defined as CHAR(16), and the conversation ID should be converted to hexadecimal format for transmission back and forth between the client application.

## Testing APPC/MVS Stored Procedures

Testing an APPC/MVS stored procedure is not difficult, due to the useful information returned from the APPC/MVS Error\_Extract API. It includes the message text that is usually displayed on the console, including error message numbers. This helps with the diagnosing and debugging of APPC/MVS-related errors.

Figuring out whether the input messages are being formatted correctly is another matter. APPC/MVS supports very good tracing facilities. According to IBM's documentation, you can get message flows and contents using the APPC/MVS tracing and debugging facilities.

Tracing the actual APPC/MVS message flows and content is generally not necessary, unless the stored procedure application program is attempting to interface with a very complicated IMS transaction. An alternate approach is to use the IMS trace facilities. With this approach, IMS logs the DL/I calls generated by the transaction. By inspecting the SSAs that were generated and the data returned, you can determine whether the input messages are formatted correctly. The only drawback to this approach is that you need some knowledge about the IMS transaction to figure out if the call patterns and returned information is correct.

To activate IMS tracing and get trace output:

1. Activate IMS tracing by issuing the command:

```
/TRACE SET ON PSB PSB-Name
```

where *PSB-Name* is the name of the PSB associated with the IMS transaction being tested.

Then, for each test run, perform the following actions:

- a. Run a client application that invokes the stored procedure.
- b. Once the stored procedure completes, issue the IMS `/CHECKPOINT` command to have the IMS log buffers flushed to disk.
- c. Run the IMS trace log print utility DFSERA10, referencing the online log(s) for your IMS system.
- d. Review the output from DFSERA10 to see if the DL/I calls look appropriate.

## Sample Stored Procedures

Two sample stored procedures are supplied that interface with two of the IMS IVP sample transactions. They are:

- CACSPTNO—Interfaces with the IMS IVTNO non-conversational transaction.
- CACSPTCV—Interfaces with the IMS IVTCV conversational transaction.

These samples are located in the SCACSAMP library. Also included is a member, CACSPMPP, which contains the CREATE PROCEDURE grammar for these two sample stored procedures.

The sample IMS transactions are very simple and provide QUERY, INSERT, UPDATE and DELETE functions to a sample employee database. IVTNO expects a single message with a command code, and the database is immediately updated and a single message is returned. IVTCV performs the same operation, but the changes are not committed until an END message is sent to it.

The sample stored procedures are simplified because they do not use security or sync-point control. The processing flow for CACSPTNO is:

1. Establish address ability to the input/output parameters it was passed.
2. Allocate the conversation using hard-coded values.
3. Format the input message buffer.
4. Send the message to IMS.
5. Wait for the output message from IMS.
6. Updates any output or input/output parameters.
7. De-allocates the conversation.

The CACSPTCV is similar to CACSPTNO, and includes additional send and receive steps to send the “END” message and wait for the response.

Both samples contain limited error detection and reporting. If an error is reported, the APPC/MVS Error\_Extract service (API) is called to obtain detailed information about the error condition. The error information is displayed on the console and a general error return code is returned to the calling application. This approach was taken because once the stored procedure has been successfully tested, errors should not occur in production operations. Another reason is that end-users probably will not know what to do with the detailed information that is provided when an APPC/MVS error is reported.

### Adding Transaction Security

As mentioned above, the sample stored procedures are unsecured transactions—they do not pass a user ID or password on the allocate request. To use secured transactions, the stored procedure application can use the CACSPGUI (Get User ID) and CACSPGPW (Get Password) subroutines to obtain the current user’s user ID and password for inclusion in the allocate call.

## Sync-Point Conversations

The sample stored procedures do not use the LU 6.2 sync-point protocols to communicate with APPC/IMS.

The *IMS Design Guide* contains diagrams that show the sync-point control flows for the different types of IMS transactions. The APPC/MVS documentation describes the formats of the calls that need to be issued. These calls are less complex than the ones that the sample stored procedures contain to allocate the conversation, send/receive messages, de-allocate the conversation, and obtain error information.

# Support Routine Descriptions

eXadas passes more information to your stored procedure application program than the SQLDA structure that is documented. There is a hidden area that precedes the SQLDA that eXadas uses to establish and maintain communications with other eXadas components.

At this time, eXadas provides three support routines that lets you get a copy of the connected user ID, their password and a copy of the RUN OPTIONS parameter that was supplied on the CREATE PROCEDURE statement. Unlike the CICS and IMS DRA interface these support routines are supplied in object form for direct inclusion in your stored procedure application program. These support routines are written in Assembler Language and are passed two parameters. They are:

- the SQLDA structure and
- the address of an output field where the requested information is moved.

Like the CICS and IMS DRA interface these support routines are written to accept 31-bit addresses. However, unlike the CICS and IMS DRA interfaces these support routines do not perform validation checks to confirm that the first parameter is in fact a pointer to the SQLDA. Failure to pass the SQLDA as the first parameter generally results in some form of addressing exception abend.

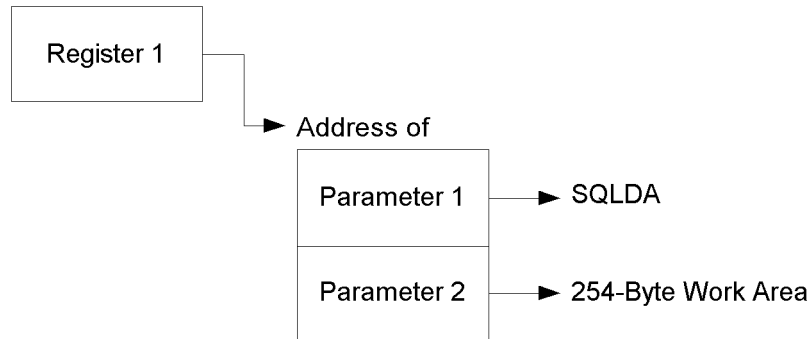
[Table 45, “Support Routines,” on page 272](#), identifies the support functions that are available. The calling conventions for each support routine are described and minimal linkage instructions in discussed in the following sections:

- [“Get RUN OPTIONS \(CACSPGRO\) Calling Conventions,” on page 332](#),
- [“Get User ID \(CACSPGUI\) Calling Conventions,” on page 332](#), and
- [“Get User Password \(CACSPGPW\) Calling Conventions,” on page 333](#).

## Get RUN OPTIONS (CACSPGRO) Calling Conventions

Support routine CACSPGRO copies the RUN OPTIONS parameter specified on the CREATE PROCEDURE statement into a 254-byte work area supplied by your stored procedure application program. The following figure shows the calling conventions used to invoke CACSPGRO.

**Figure 46: CACSPGRO Calling Conventions**



CACSPGRO always issues a return code of zeros. Upon return the area referenced by Parameter 2 contains a copy of the RUN OPTIONS parameter padded with blanks.

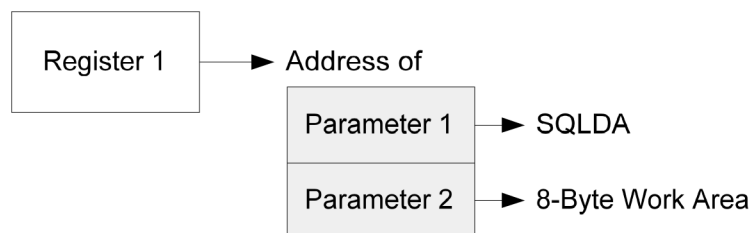
To link-edit CACSPGRO into your stored procedure application, concatenate library SCACSAMP into the SYSLIB DD statement on the link step and include the following in the SYSIN DD statement on the link step:

```
INCLUDE SYSLIB(CACSPGRO)
```

## Get User ID (CACSPGUI) Calling Conventions

Support routine CACSPGUI copies the connected user, user ID into an 8-byte work area supplied by your stored procedure application program. The following figure shows the calling conventions used to invoke CACSPGUI.

**Figure 47: CACSPGUI Calling Conventions**



CACSPGUI always issues a return code of zeros. Upon return the area referenced by parameter 2 contains a copy of the connected user's user ID padded with blanks. If no user ID was supplied, the output area contains spaces.

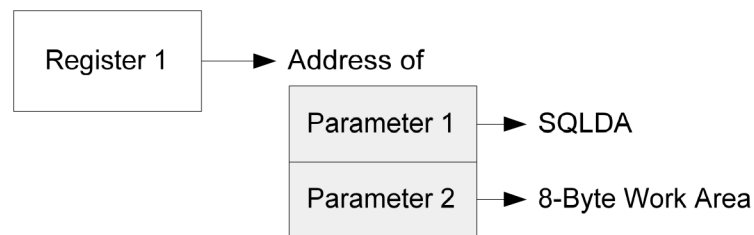
To link-edit CACSPGUI into your stored procedure application, concatenate library SCACSAMP into the SYSLIB DD statement on the link step and include the following in the SYSIN DD statement on the link step:

```
INCLUDE SYSLIB(CACSPGUI)
```

## Get User Password (CACSPGPW) Calling Conventions

Support routine CACSPGPW copies the connected user password into an 8-byte work area supplied by your stored procedure application program. [Figure 48: "CACSPGPW Calling Conventions," on page 333](#), shows the calling conventions used to invoke CACSPGPW.

**Figure 48: CACSPGPW Calling Conventions**



CACSPGPW always issues a return code of zeros. Upon return the area referenced by Parameter 2 contains a copy of the connected users password padded with blanks. If no password was supplied, the output area contains spaces.

To link-edit CACSPGPW into your stored procedure application, concatenate library SCACSAMP into the SYSLIB DD statement on the link step and include the following in the SYSIN DD statement on the link step:

```
INCLUDE SYSLIB(CACSPGPW)
```



# Enterprise Server

## Introduction to Enterprise Server Installation and Configuration

The Enterprise Server can be used to manage a large number of concurrent users across multiple data sources. An Enterprise Server sits between your application and the eXadas Server. It appears as the server to the client and as the client to the server. The Enterprise Server is responsible for starting additional servers as the number of concurrent users increases.

An Enterprise Server contains the same tasks that a server uses, with the exception of the Query Processor and the Initialization Services. Like a server, the Enterprise Server's Connection Handler is responsible for listening for client connection requests. However, when a connection request is received, the Enterprise Server does not forward the request to a Query Processor task for processing. Instead, the connection request is forwarded to a Data Source Handler (DSH) and then to a server for processing. The Enterprise Server maintains the end-to-end connection between the client application and the target server. It is responsible for sending and receiving messages between the client application and the server.

The Enterprise Server is also used to perform load balancing. Using configuration parameters, the Enterprise Server determines the locations of the servers that it

communicates with and whether those servers are running on the same platform as the Enterprise Server.

The Enterprise Server can automatically start a local server if there are no instances active. It can also start additional instances of a local server when the currently-active instances have reached the maximum number of concurrent users they can service, or the currently active instances are all busy.

This chapter contains the following sections:

- [“Deployment of the Enterprise Server,” on page 336](#), which describes how to install and deploy the sample for the Enterprise Server.
- [“Operations,” on page 339](#), which describes starting and stopping the Enterprise Server.

## Deployment of the Enterprise Server

This section details the steps necessary to bring up an eXadas Enterprise Server and access data using standard SQL.

The Enterprise Server invokes the eXadas base product server so that you can readily verify the deployment and configuration of the Enterprise Server (having already verified the base product). At the end of this process, you will have dynamically started a server (CACDS) using an Enterprise Server (CACES); have a Client Application (CACCLNT) issue a connection request to the Enterprise Server; and have it retrieve data back from the server. The connection from the Enterprise Server to the eXadas server will use the Cross Memory Communication Protocol.

For information on specific Enterprise Server configuration parameters, see [Appendix A, “Configuration Parameters.”](#)

### **To deploy the Enterprise Server:**

1. Ensure that CACDS can be started successfully.
2. Edit the base product server configuration to prepare for CACDS initialization by the Enterprise Server (CACES).
3. Apply the necessary customizations to the Enterprise Server JCL and examine the configuration.
4. Submit the Enterprise Server JCL.
5. Submit the batch Client Application JCL and verify the output.



## Deployment Steps

1. Ensure that CACDS has access to the necessary data sets.  
See the Installation chapter in the *eXadas OS/390 Getting Started Guide* for instructions on how to start the eXadas server as a started task.
  - a. Apply the necessary JCL changes to prepare the eXadas server to be started by an Enterprise Server.
  - b. Along with all the changes made to the eXadas server JCL during the product installation process, uncomment the PARM1 symbolic in the PROC statement and the PARM card in the EXEC statement.
2. Edit the server master configuration file to prepare the server to be started by an Enterprise Server (SCACCONF member CACDSCF).
  - a. Comment out the Service Info Entry for the Connection Handler used when configuring the eXadas server for communications. A Connection Handler Service is dynamically started within the server when it is started by the Enterprise Server, CACES. The communication parameters for this Connection Handler Service are specified on the Service Info Entry of the CACSAMP DSH in the SCACCONF member CACESCF. If a server is already listening where these parameters specify, and if peer support has been specified on the Enterprise Server Service Info Entry, the Enterprise Server will connect to that existing server.
  - b. Copy the CACES member from the SCACSAMP data set to your PROCLIB.
3. Apply the necessary customizations to the Enterprise Server JCL.  
Edit the eXadas data set high-level qualifier (SCACSAMP member CACES), and valid SYSOUT class.
4. Save this member.
5. Edit the Enterprise Server configuration file (SCACCONF member CACESCF).

Uncomment the SERVICE INFO ENTRY for the Cross Memory Data Source Session Handler Service. Also uncomment the SERVICE INFO ENTRY for the Connection Handler used when configuring the eXadas Server for communications, using the same references in the last field.

**NOTE:** The Cross Memory CACINIT Service Info Entry specifies the same data space name/queue name as in the Server's configuration. The CACSAMP Cross Memory DSH Service Info Entry has a different data space name/queue name specified (CACD/CACQ). These two Service Info Entry parameters must specify different values if using the same Transport Layer protocol between the client and the Enterprise Server and between the Enterprise Server and the eXadas Server. [Table 64, "Communication Parameter Correlation," on page 338](#), shows the Communication Parameter Correlation between a Client, an Enterprise Server, and a Server.

6. Issue an operator command to start the Enterprise Server.
  - a. From the OS/390 console, issue a standard start command:
 

```
S CACES
```
  - b. Examine the job output while it is executing, you will see the following message near the top of the listing:
 

```
CAC00103I EXADAS SERVER: V2.2.4 READY
```

**NOTE:** You should also see that the server has been started and initialized.
7. Submit the base product batch Client Application: the SCACSAMP member CACCLNT.
  - a. Use the same queries used for bringing your data source on line or follow the steps in the *eXadas OS/390 Getting Started Guide* for bringing Sequential data on line and use the Sequential queries.
  - b. View the output. The output should contain the SQL statements being used and the corresponding result set.

**Table 64: Communication Parameter Correlation**

Client	Enterprise Server		Server	
Data Source (Field 1 & 2)	Connection Handler Service  Information Entry (Field 10)	Data Source Session Handler Service  Information Entry (Field 2 & 10)	Connection Handler Service Information Entry (Field 10)	Query Processor Service Information Entry (Field 2)
CACSAMP (field 1)		CACSAMP (field 2)		CACSAMP
XM1/CAC/CAC (field 2)	XM1/CAC/CAC			
		XM1/CACD/CACQ (field 10)	*	

\* Connection Handler Service uses parameters passed from the DSH that requested the server to be started.

---

# Operations

This chapter provides a brief overview of running an Enterprise Server in a test and production environment. The following topics are discussed in this section:

- [“Starting OS/390 Enterprise Servers,” on page 339,](#)
- [“Monitoring and Controlling Enterprise Servers,” on page 340,](#)
- [“Starting and Stopping Individual Services,” on page 343, and](#)
- [“Stopping the Enterprise Server,” on page 344.](#)

For a list of MTO commands, see [Appendix C, “MTO Command Reference.”](#)

## Starting OS/390 Enterprise Servers

An eXadas Enterprise Server is a special type of server that loads one or more Data Server Handler (DSH) services. Typically, an Enterprise Server does not run any Query Processor (QP) services.

The main purpose of an Enterprise Server is to act as a proxy for sessions between eXadas client/ODBC applications and servers. The Enterprise Server is also responsible for dynamically starting and stopping servers based upon user load. In cases where multiple server instances are using the same service name, an Enterprise Server will balance new connections across all instances. A properly configured Enterprise Server offers scalability, dynamic scheduling, load leveling, and/or protocol translation (Gateway) to an eXadas client/server environment.

A DSH service instance is responsible for starting, stopping, and routing sessions to a corresponding server instance. There is a one-to-one relationship between a DSH instance and a server instance. If a DSH instance is started, then its corresponding server is started. If a DSH instance is stopped, then its corresponding server instance is stopped. If a server instance that has been started by a DSH is stopped independently of the Enterprise Server, then the Enterprise Server is notified and the corresponding DSH instance is stopped. An OS/390 DSH can control two types of servers: started tasks and/or batch jobs.

OS/390 Enterprise Servers can be run as either an OS/390 batch job or a started task. Like any batch job or started task, the user ID of the Enterprise Server itself is based on the submitter of the batch job or the user ID associated with a started task. Enterprise Servers are typically not configured to access data locally (using a Query Processor service), but if so configured they are subject to the same security restrictions as a server. Sessions routing through an Enterprise Server are subject to the security restrictions of the target server.

OS/390 servers started by an Enterprise Server can either be started tasks or batch jobs. When a DSH is configured to start multiple instances of a started task, the procedure name in field 11 of the DSH Service Info Entry is suffixed by a unique

identifier field generated by the DSH. The procedure name should not exceed 8 characters and the identifier is in the following format:

Tyyyyyyy

where yyyyyy is the EBCDIC representation of the TASKID of the DSH, such as DSPROC.T3589342. When a DSH is configured to start multiple instances of a batch job, the batch job name, which must not exceed 6 characters, is suffixed by a unique 2 character hexadecimal field, such as CACDS00.

A batch server is determined by setting the Task Type field of the corresponding DSH Service Info Entry to 3. A started task is determined by setting the same field to 2. In either case, field 11 of the DSH Service Info Entry is used to specify the JOB/PROC name, which follows the communications compound address field of the server that will be started by this DSH.

The OS/390 server configuration should not specify a Connection Handler Service, CACINIT, as the passed compound address field will trigger the server's region controller to start one up for the specified address.

For example:

```
SERVICE INFO ENTRY = CACDSH CACSAMP 3 2 4 1 4 5M 5M  
XM1/CACD/CACQ CACDS
```

This SIE specifies a server to be started as a batch job (field 3 = 3) using the JCL from the member name CACDS and that it will listen for connections on the address XM1/CACD/CACQ.

**NOTE:** The member CACDS must reside in the PDS data set specified by the JOBDEF DD name.

Once an Enterprise Server is started, it is ready to accept client connections and requests for data.

## Monitoring and Controlling Enterprise Servers

The server and the Enterprise Server are designed to run continuously. eXadas supplies an OS/390 MTO (Master Terminal Operator) interface that can be used to monitor and control server/Enterprise Server operations. Using the MTO interface, you can issue the following commands:

- Display active services in an Enterprise Server,
- Display configurations,
- Modify configuration parameters,
- Display memory utilization,

- Display users,
- **START** and **STOP** individual services, and
- **STOP** the Enterprise Server.

The basic format of an OS/390 MTO command is:

```
F servername , command , [ subcommand1 , subcommand2 , . . . ]
```

When executing these commands under IBM's SDSF, you must precede the command with the slash (/) character, as shown in the following example:

```
/F CACDS001 , DISPLAY , SERVICES
```

### Displaying Active Services in an Enterprise Server

At start-up, the Enterprise Server processes the master configuration file and starts services defined with SERVICE INFO ENTRY parameter values. The number of services started depends on the minimum tasks entry (field 4) in each SERVICE INFO ENTRY.

**NOTE:** This field can be set to 0 when you do not want the service started automatically at Enterprise Server initialization. For more information on the SERVICE INFO ENTRY parameter settings, see [Appendix A, "Configuration Parameters."](#)

To display the active services, issue the **DISPLAY,SERVICES** command from the OS/390 Master Console or SDSF. Sample output from the **DISPLAY,SERVICES** command follows.

```
F servername , DISPLAY , SERVICES
Service Type TASKID TASKNAME Status User
LOGGER CACLOG 9392624 CACLOG READY
CACSAMP CACDSH 9270608 CACDSH READY
CACSAMP CACDSH 9269664 CACDSH RECEIVING
XMQ CACINIT 9215624 CACINIT READY
TCPIP CACINIT 9214680 CACINIT READY
```

**NOTE:** The active service display only shows services that are currently running in the Enterprise Server. To view all services defined to a Enterprise Server, you must display the Master Configuration with the command **DISPLAY,CONFIG=MASTER**. For more information on displaying configurations, see ["Displaying Configurations," on page 342.](#)

### Displaying Servers Connected to an Enterprise Server

Once a server is started by an Enterprise Server, it is represented locally as a DSH service instance. For each DSH Service instance displayed by the **DISPLAY,SERVICES** command there is a corresponding server instance running. Individual users assigned to a particular server can be displayed only on that server.

## Displaying Configurations

Configuration information is loaded from members of the VHSCONF data set at Enterprise Server initialization time. The initial member identified in the VHSCONF DD is known as the MASTER configuration member. Minimally, there is a MASTER configuration active in all running Enterprise Servers.

To display the MASTER configuration, issue the following MTO command:

```
F servername,DISPLAY,CONFIG=MASTER
```

The output from this command follows:

```
Configuration: MASTER
*(1)MESSAGE POOL SIZE = 16777216
  (2)TASK PARAMETERS = NULL
  (3)USER CONFIG = 0
  (4)STATIC CATALOGS = 0
*(5)NL = US ENGLISH
*(6)NL CAT = DD:ENGCAT
  (7)BTREE BUFFERS = 4
  (8)LD TEMP SPACE = ALCUNIT=TRK,SPACE=15,EXTEND=5
  (9)SAF EXIT = NULL
  (10)SMF EXIT = NULL
  (11)MAX ROWS EXAMINED = 0
  (12)MAX ROWS RETURNED = 0
  (13)MAX ROWS EXCEEDED ACTION = ABORT
  (14)JOIN MAX TABLES ANALYZED = 4
  (15)CPU GOVERNOR = NULL
  (16)LOCALE = NULL
  (17)WLM UOW = NULL
  (18)PDQ = 0
  (19)INTERLEAVE INTERVAL = 100
  (20)VSAM AMPARMS = NULL
*(101)SERVICE INFO ENTRY = CACCNTL CNTL 0 1 1 100 4 5M 5M
NO_DATA
*(102)SERVICE INFO ENTRY = CACLOG LOGGER 1 1 1 100 1 5M 5M
NO_DATA
*(103)SERVICE INFO ENTRY = CACDSH CACSAMP 3 1 10 50 4 5M 5M
XM1/CACD/CACQ CACEDS
*(104)SERVICE INFO ENTRY = CACINIT TCPIP 2 1 10 100 4 5M 5M
TCP/199.222.141.33/7070
```

Lines in the output display prefixed by an asterisk (\*) denote configuration values that were either read in at initialization time or updated by an MTO operator SET command. All other configuration items are set to their default values as defined in [Appendix A, “Configuration Parameters.”](#)

Displaying the MASTER configuration is the only way to view all the SERVICE INFO ENTRYs defined to a running Enterprise Server. While many services can be viewed with a **DISPLAY,SERVICES** command, services that are not running cannot be viewed.

To display active configurations while an Enterprise Server is running, issue the **DISPLAY,CONFIGS** command.

The master configuration is always active while a server is running.

## Modifying Configuration Parameters

Any active configuration can be dynamically modified with the MTO **SET** command. The format of the **SET** command is as follows:

```
F servername,SET,NAME=configname,ORD=ordinalnumber,
VALUE='value'
```

**WARNING:** Dynamic modification of production servers is not recommended unless the full impact of configuration changes is known.

In general, configurations are generally static and modification at runtime is not necessary.

## Displaying Memory Utilization

Enterprise Servers run with a fixed pool of memory that must be shared by all services and users. The size of an Enterprise Server Memory Pool is defined by the MESSAGE POOL SIZE parameter in the master configuration file. To display the current memory utilization from the message pool, issue the command:

```
F servername,DISPLAY,MEMORY
```

The output of the command is as follows:

```
Total Memory 6835K, Available 5918K (86%)
```

The memory display includes:

- the total size of the pool,
- the amount of memory available for use by services and users, and
- the percentage of the pool that is still available.

While memory requirements vary based on the number of user connections, this command can be useful in estimating the maximum number of connections allowable for a given Enterprise Server.

**NOTE:** The MESSAGE POOL SIZE for an Enterprise Server must be at least 2 megabytes less than the actual region size for the server as memory must also be available for load modules, C runtime stacks, and OS/390 control blocks.

## Starting and Stopping Individual Services

Using the MTO interface, you can issue commands to start and stop individual services defined in the master configuration.

To start a new instance of a service, issue the command:

```
F servername , START , SERVICE= servicename
```

For example, to start a new instance of the service CACSAMP as shown in [“Displaying Configurations,” on page 342](#), issue the following command:

```
F servername , START , SERVICE=CACSAMP
```

When starting a service, the region controller first checks the number of active instances of the service and the maximum task count for the service before attempting to start a new instance. If the service has not reached its defined maximum task count, the region controller starts a new instance of the service.

Stopping a service can be performed either by its service name or its task ID. If a service is stopped by its service name, all active instances of the service are stopped. The formats of the **STOP** command for individual services are as follows:

```
F servername , STOP , SERVICE= servicename
```

```
F servername , STOP , TASKID= taskid
```

**WARNING:** The **STOP** command cancels any user activity in an active service and disconnects all active users from the stopped service.

Uses for the **START** and **STOP** commands include:

- starting additional instances of a DSH,
- stopping idle instances of a DSH,
- stopping a DSH that has been orphaned by a Server, and
- starting new Connection Handlers, for example, Cross Memory and TCP/IP.

## Stopping the Enterprise Server

To stop the Enterprise Server, enter the following command:

```
F servername , STOP , ALL
```

**WARNING:** The **STOP** command stops all DSH instances and, therefore, all connected servers corresponding to those DSH instances. All user connections to those servers will be terminated.



---

# Integration and Configuration

The increased degree of scalability and the communication gateway functionality that are inherent to an Enterprise Server are the primary reasons for implementing it into an eXadas eData Engine. These functions, as well as the ability to implement the Enterprise Server as needed, are accomplished by using an uncomplicated communication configuration scheme. This scheme allows the introduction of the Enterprise Server into an existing system to be transparent to all client applications and to have a minimal impact on existing server configurations.

This chapter discusses the Enterprise Server configuration parameters, and the available functionality and features implemented by them. A reference of detailed parameter syntax is located in [Appendix A, “Configuration Parameters.”](#)

## Base System Configuration Review

The configuration of an Enterprise Server does not involve a large number of parameters. There are no query optimization nor data-specific parameters available. This query-related tuning is accomplished mainly through data mapping and index definition in the Meta Data Grammar. There are also capacity-related parameters available within the client application and server configurations.

To properly implement an Enterprise Server into an existing system, you must understand the base configuration (with no Enterprise Server). The base configuration of an eXadas system is mainly concerned with communication configuration. That is, specifying the proper values on the correct parameters so that client applications connect to the proper server in order to retrieve the desired data. This has become a greatly simplified process with this version of eXadas.

### Parameter Correlation

There are only three parameters involved in establishing a client-to-server connection: the DATASOURCE parameter in the client configuration, and two SERVICE INFO ENTRY parameters in the server configuration. The DATA SOURCE parameter consists of two values: the data source name and the listener address of the target server. A request from the client application results in a connection request being sent to the address specified in the DATA SOURCE parameter. Once this client-to-server Connection Handler Service connection is established, a session is established between the client and the Query Processor that services the specified Data Source. A server Connection Handler Service listens on the address specified in the Service Information field (field 10) of its SERVICE INFO ENTRY parameter. The SERVICE NAME field (field 2) of a Query Processor’s SERVICE INFO ENTRY specifies the Data Source that it will service. The coordination of these three values is all that is involved in establishing a connection that a Client Application can use to retrieve data.

## Supported Protocols

The communication protocols that this version of eXadas supports include:

- TCP/IP,
- IBM MQ Series, and
- Cross Memory for the OS/390 platform.

The Cross Memory method uses queues maintained within OS/390 data spaces to transport data between eXadas components that exist on the same OS/390 image. The use of this method greatly reduces the overhead involved in translating the data to and from a communication protocol while eliminating the overhead of any communications subsystem and avoiding network bandwidth constraints. Use this method whenever possible.

**NOTE:** Even though it is not technically a protocol, for documentation simplification purposes, the Cross Memory communication method will be addressed as a protocol with the other supported protocols in this guide. Assume that when communication protocols are being discussed, the Cross Memory method is included in that discussion, unless otherwise noted.

## Communication Value Formats

The format for specifying the Connection Handler of a listen address is protocol-dependent. The first value is always a keyword denoting which protocol is to be used, followed by the values necessary for that protocol.

For TCP/IP, the keyword is `TCP`, and the necessary values are an IP address and a port number (`TCP/111.111.111.111/2222`). If your TCP/IP subsystem supports it, the IP address and port number can be specified as host and service names.

For Cross Memory, the keyword is `XM1`, followed by a data space name and a queue name (`XM1/CACD/CACQ`). These data space and queue names do not need to be predefined to any communication subsystems.

The following restrictions apply when using the Cross Memory protocol:

- Data Space name consists of 1 to 4 alphanumeric characters.
- Queue name consists of 1 to 4 alphanumeric characters.
- All Data Space names must be unique across an OS/390 image.
- Multiple servers can utilize the same Data Space/Queue name combination, although a unique combination for each server is recommended. Unique combinations can speed display output interpretation during problem diagnosis.
- A Data Space is initialized by the first server that references it.
- Data Space size is 8 MB, which is not configurable, and contains message buffers and related control blocks.
- Maximum 400 users per Data Space.
- Maximum 100 Services per Data Space.

The TCP/IP protocol implementation within eXadas allows remote addresses to be specified where applicable (a Server Connection Handler Service cannot listen on a remote address). For TCP/IP, a remote IP address/HOST name is specified in the same format as a local address/HOST name. By its nature, there are no remote capabilities with Cross Memory.

## Enterprise Server Integration

This section explains the steps necessary to integrate an Enterprise Server into an existing eXadas network, and details the configuration parameter values required to implement the functionality it provides.

### The Mechanics of a Transparent Integration

You can transparently integrate an Enterprise Server into an existing eXadas network because neither the client nor the server needs to know about the Enterprise Server's presence. This is possible because the Enterprise Server interacts with client applications using the same interface as a server, and appears to be a client application to servers. To make the integration of the Enterprise Server transparent to the client, the Connection Handler Service within the Enterprise Server must be configured to listen on the same communication service name as the eXadas Server Connection Handler Service. This necessitates changing the server configuration so that it is not listening on the same communication address.

## Data Source Handler Service Configuration

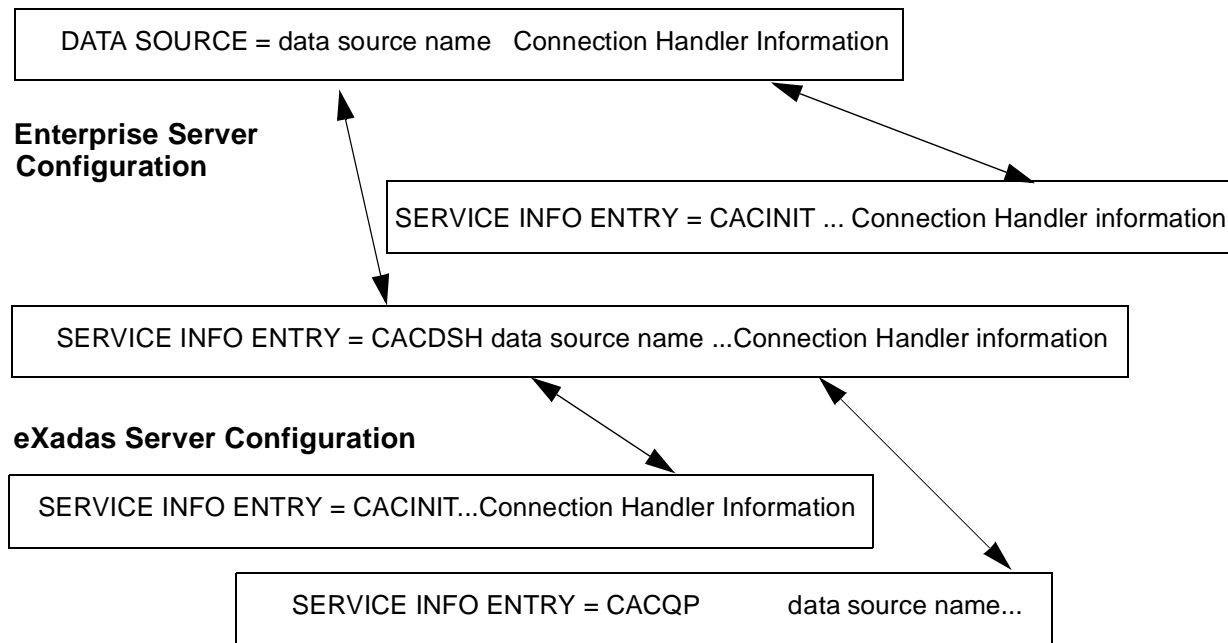
The Enterprise Server contains a new service that it uses to communicate with servers. This service is called the Data Source Handler (DSH). As the name implies, there is a one-to-one correlation between an instance of this service and an eXadas server. The parameter that is used to configure a DSH is the SERVICE INFO ENTRY (SIE). In addition to the information supplied in the common SIE subparameters, the SERVICE INFORMATION (field 10) that is passed to the

DSH when it is initialized is the listen address and optionally the procedure name/JCL of the eXadas server that it will connect to.

The following diagram may assist you in configuration. The arrows represent fields that must match in order for your configuration to be successful.

**Figure 49: Client-to-eXadas Server Connection Handler Service**

### Client Configuration



## Dynamic eXadas Server Scheduling

The DSH attempts to connect to the specified eXadas server at the communication address it was passed. If the connection is established the DSH waits for a client connection request from one of the Enterprise Server Connection Handler Services. A connection request is routed to a particular DSH by matching the DSH Service Name (field 2 of its SIE) with the Data Source name passed in the request.

If a procedure name or job name is specified in field 11 of the DSH SIE, the DSH starts a server. A server is started by either procedure (using internal start command) or batch (the DSH submits the job into an internal reader). These functions are referred to as Dynamic Server Scheduling and the DSH then waits for a client's connection request. This only happens if you specify a procedure name or job name.

The DSH passes the communication address value from its SERVICE INFORMATION (field 10 of its SIE) to the started eXadas server. The eXadas Server Region Controller will then start a Connection Handler Service using the communication address value that was passed. This eliminates the need to specify SERVICE INFO ENTRYs within the eXadas server configuration for Connection Handler Services. The administrative task of maintaining their coordination with those in the Enterprise Server configuration is eliminated.

The information to start a Connection Handler Service is passed as an execution parameter within the internal console (MTO) **START** command or when the job is submitted to the internal reader. This is the reason all server sample JCL members include a PARM1 field on the PROC and EXEC statements. If the eXadas server was started from the console, or the eXadas server is running as a JOB submitted from a TSO/E session, the eXadas server initialization routines ignore the NULL parameter.

## Expanded Scheduling and Connection Balancing With Cross Memory

There are restrictions within the TCP/IP protocol that prohibit more than one Connection Handler Service from listening on the same communication address. The dynamically started Connection Handler Service within server occurrences that are dynamically scheduled by the same DSH will attempt to listen on the same address (or set of addresses). This will result in a communication error if the protocol being used by the services is TCP/IP. This situation occurs whether the listen address was read from the configuration file, or was passed from the Enterprise Server. To prevent this, the maximum value allowed for the MAX TASKS field of a DSH Service Info Entry (field 5) is 1 when TCP/IP is specified as the communication protocol. This effectively limits the number of Servers by data source name to 1 for TCP/IP.

The restrictions noted do not apply when the Cross Memory protocol is used to manage server sessions. The Enterprise Server can manage multiple occurrences of servers that were dynamically scheduled from the same DSH service within that Enterprise Server. These occurrences can be initialized either at Enterprise Server start-up, or on an as-needed basis. There is a one-to-one relationship between a DSH thread and server address space. The number of servers started either initially or dynamically is a function of the DSH SIE Field 5 (Max Tasks). A subsequent occurrence of a server is scheduled when a new connection request is refused by a previously scheduled server because it was already at its MAX USERS (field 6) limit or is currently busy. After the connection is established with the new server, the new session is routed to it.

An Enterprise Server will also perform connection balancing of client sessions across these servers. When a request for a new connection is received from a

Connection Handler Service, it is routed to a currently active DSH service for the specified data source name, if possible.

A LIFO queue of active services is kept for each DSH Service Info Entry. When a new connection request is routed to a DSH service, its entry in the queue is moved to the bottom of the queue. The entries in this queue are checked from the top down. The connection request is routed to an active instance of a service if the number of connections that instance is already serving is less than the value specified in field 6 (Max Connections/Task), and the instance is not busy actively connecting or disconnecting a connection. If the connection request cannot be routed to a given instance, then the next entry in the queue is checked.

If none of the services are available, field 5 (Max Tasks) of the DSH Service Info Entry is checked to see if another service can be activated. If so, that service (and its associated server) is started and the connection request routed to it. If max tasks for this service has been reached, then an error is reflected back to the client indicating that there are no services available to service the connection request.

## Using IMS DBB Access in a DBRC FORCE=YES Environment

Some sites require that all batch IMS application systems access the IMS data using a DBB interface. At these sites, data integrity between these systems and any on-line systems is usually enforced by establishing a DBRC FORCE=YES environment. This requires that all DBB JOBS or started tasks have a unique name.

eXadas provides an extension to the Enterprise Server's dynamic scheduling of servers in order to support this type of environment. This is accomplished by allowing the Enterprise Server to submit server JCL streams as batch JOBS with unique names. This is invoked by specifying the Task Start Class of the desired DSH Service Info Entry as a value of three (3). The eXadas Server Name portion of that Service Info Entry's field 11 then signifies the name of a member in the PDS pointed at by the JOBDEF DD in the Enterprise Server's JCL instead of a procedure name. A symbolic parameter (&CACJSFX) must be appended to the JOB name within the JOBDEF member's JOBCARD. The value of this parameter is incremented each time a new server is scheduled for a given DSH, thus ensuring unique JOB names for every server started for this DSH. There is another symbolic parameter (&CACPRM1) that is used to place the communication address information for the dynamically started Connection Handler Service into the member's JCL. Both of these symbolic parameters are replaced by the Enterprise Server prior to the JCL being submitted through an internal reader.

This feature is intended for use with the Cross Memory communication protocol only. This is due to the restrictions of the TCP/IP protocol being limited to a single instance of a server listening on the same communications address.

The following requirements and restrictions apply to this feature:

- The Task Start Class field of the DSH SIE must be set to 3.
- The Enterprise Server JCL must contain a JOBDEF DD statement pointing at a valid PDS or PDSE.
- The member name specified in field 11 of the DSH SIE must exist in the JOBDEF PDS.
- The JCL within the member must contain a JOBCARD.
- The JOB name on the JOBCARD must not be longer than 6 characters and must be suffixed by the &CACJSFX symbolic parameter. (See Example 1.)
- If the JCL within the member invokes the server by executing a procedure, then PARM1 on the EXEC statement must be equated to the symbolic parameter &CACPRM1. (See Example 2.)
- If the JCL within the member invokes the server by executing a program, then the parameter string on the EXEC card must start with the symbolic parameter &CACPRM1 enclosed in quotes. (See Example 3.)

The following examples show the use of the symbolic parameters.

#### **Example 1: A Job Statement**

```
//JOBNAM&CACJSFX JOB(0),ABCDEF,CLASS=A.....
```

#### **Example 2: An Execute Procedure Statement**

```
//E1 EXEC PROC01,PARM1=('&CACPRM1',.....)
.
.
.
.
//STEP1 EXEC CACCNTL,PARM=('&PARM1',.....)
```

#### **Example 3: An Execute Program Statement**

```
//STEP1 EXEC CACCNTL,PARM=('&CACPRM1',.....)
```





## SQL Update

### Introduction to SQL Update

eXadas, like other database systems, supports updating application data in a secure transaction environment. Changes to data can be aggregated and committed as a single logical unit of work and can also be backed out in the event of unexpected application errors. eXadas supports IDMS, Adabas, IMS, DB2, and CICS VSAM data sources.

This chapter describes the SQL statements that allow updating application data as well as the statements necessary to manage SQL transactions. It includes the following topics:

- [“Transactions,” on page 354,](#)
- [“SQL Update Statements,” on page 355,](#)
- [“SQL Update and Mapped Tables,” on page 357,](#)
- [“Adabas Update Considerations,” on page 360,](#)
- [“DB2 Update Considerations,” on page 362,](#)
- [“IDMS Update Considerations,” on page 362,](#)
- [“IMS Update Considerations,” on page 364,](#) and
- [“CICS VSAM Update Considerations,” on page 366.](#)

# Transactions

A transaction is a sequence of one or more SQL statements that together form a logical unit of work. A transaction automatically begins with the first SQL statement executed within a client connection to the eXadas Server.

Transactions continue through subsequent SQL statements until one of the following conditions occur:

- A client COMMIT statement ends the transaction making any database changes permanent.
- A client ROLLBACK statement aborts the transaction, backing out any database changes.
- A normal client disconnect occurs, resulting in a transaction COMMIT.
- An unexpected client disconnect occurs, resulting in a transaction ROLLBACK. In addition to committing or rolling back changes, all open cursors associated with the transaction are closed and any prepared statements are freed.

Once a transaction is completed by either a commit or rollback, a new transaction will start with the next SQL statement issued by the SQL client.

**NOTE:** Some clients support an auto-commit mode of processing that makes each SQL statement its own transaction by automatically issuing an explicit COMMIT after each SQL statement. In applications that require bundling multiple updates in the same transaction, this feature must be turned off.

eXadas is not a DBMS and therefore relies on the transaction capabilities of the underlying Data Savants. In some cases, SQL update is not allowed by the Data Savant because a method of committing or rolling back database changes is not currently supported. See the DBMS-specific sections of this chapter for details on each Data Savant's capabilities.

Currently, two-phase COMMIT is not supported, so update statements within a single transaction are limited to a single Data Savant. Queries to all Data Savants may still be issued while updating through a single Data Savant.

Attempts to update multiple database or update a single database across multiple database threads will result in a -817 SQLCODE. This SQLCODE is also returned if a Stored Procedure call or any DDL statement (for example, DROP, GRANT, or REVOKE) is issued while updates are pending against a Data Savant.

In most cases, an error SQLCODE leaves the transaction intact and pending database updates can still be committed or rolled back. However, in cases where an error is detected in the middle of updating multiple rows in a Data Savant, you must perform an explicit ROLLBACK to back out of the partial update. In this case, an SQLCODE -4910 is returned to the client. Note that this only applies when you are updating multiple rows between COMMITS. If you are updating a single record when an error occurs, eXadas performs the ROLLBACK automatically.

In general, it is good practice to issue an SQL ROLLBACK whenever a negative SQLCODE is returned from the Server.

**WARNING:** Database transactions automatically lock database resources and prevent concurrent access to other database users. Applications issuing updates should keep update transactions as short as possible to avoid contention for database resources with other users.

## SQL Update Statements

This topic describes the SQL statements used to update application data and manage transactions in the Server. Each of these statements can be dynamically prepared and executed or executed immediately by client applications. With the exception of COMMIT and ROLLBACK, prepared update statements may be executed multiple times within the same transaction without re-preparing the statement.

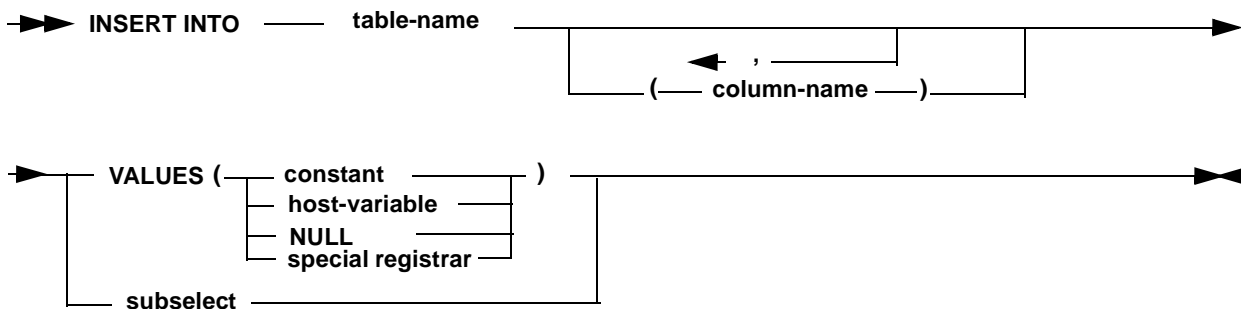
The Statements described in this section are:

- INSERT,
- UPDATE,
- DELETE,
- COMMIT, and
- ROLLBACK.

### INSERT

The INSERT statement inserts one or more rows in an application database. The format of the insert statement is:

**Figure 50: Insert Statement Format**

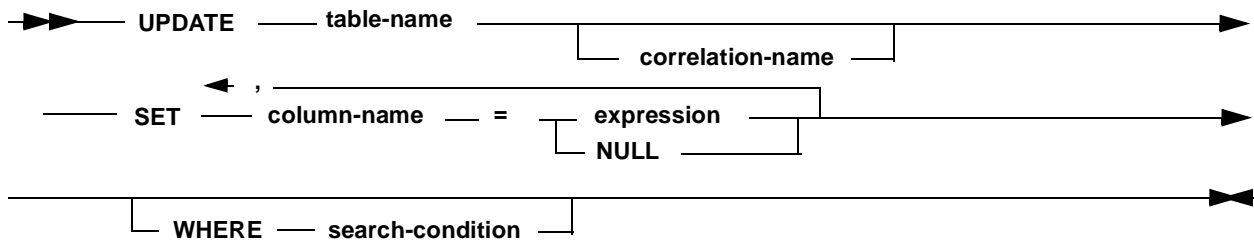


If the column names are omitted from the statement, the values or subselect list must include data for all columns of the inserted table.

## UPDATE

The UPDATE statement updates one or more rows in an application database. The format of the update statement is:

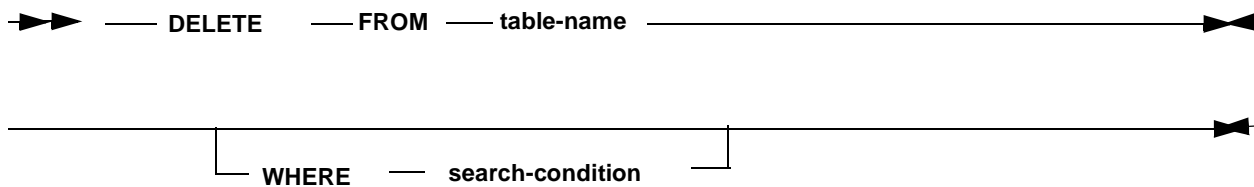
**Figure 51: Update Statement Format**



## DELETE

The DELETE statement deletes one or more rows in an application database. The format of the DELETE statement is:

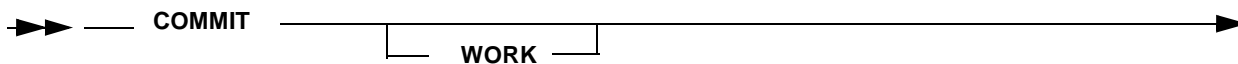
**Figure 52: Delete Statement Format**



## COMMIT

The COMMIT statement commits a transaction and makes any pending database changes permanent. In addition, all open cursors associated with the transaction are closed and all prepared statements are freed. The format of the COMMIT statement is:

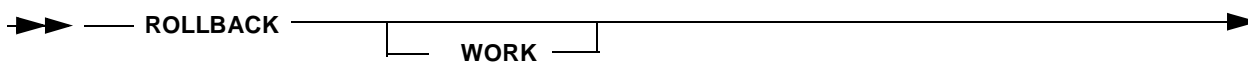
Figure 53: COMMIT Statement Format



## ROLLBACK

The ROLLBACK statement rolls back a transaction and backs out any pending database changes. In addition, all open cursors associated with the transaction are closed and all prepared statements are freed. The format of the ROLLBACK statement is:

Figure 54: ROLLBACK Statement Format



## SQL Update and Mapped Tables

The eXadas product maps both non-relational and relational databases into a DB2-like system catalog. In mapping these tables, some mapping constructs are created that have special meaning when updating mapped tables.

### Mappings Containing Multiple Records

Some Data Savants, such as IMS and IDMS, allow mapping multiple records/segments in a database path to a single logical table in the eXadas system catalog. When updating these mappings with SQL update commands, updates are only applied to the last record mapped in the path. The following example is a mapping that involves two different record types: EMPLOYEE and PAYCHECK.

```
EMPLOYEE RECORD:  SSN          CHAR(9)
                  LAST_NAME   CHAR(20)
                  FIRST_NAME  CHAR(20)
PAYCHECK RECORD:  PAY_DATE    DECIMAL(8,0)
                  GROSS_PAY   DECIMAL(15,2)
```

NET_PAY	DECIMAL(15,2)
FED_TAX	DECIMAL(7,2)
STATE_TAX	DECIMAL(7,2)
FICA	DECIMAL(7,2)

The table named EMPLOYEE\_PAY, which maps the database path EMPLOYEE to PAYCHECK, returns each employee record combined with each of the employee's paycheck records in a standard SQL query. In the case of an update call, updates apply only to the PAYCHECK record.

To update any information in the EMPLOYEE record, a separate table mapping must be created that contains only the employee record.

Considering the example above, the following is sample SQL that inserts a new paycheck record:

```
INSERT INTO EMPLOYEE_PAY (SSN, PAY_DATE, GROSS_PAY, NET_PAY,
    FED_TAX, STATE_TAX, FICA)
VALUES( '012339920', 10311999, 4200.00, 3300.00,
    800.00, 75.00, 25.00 );
```

**NOTE:** In this case, a value is provided for the EMPLOYEE SSN even though the EMPLOYEE record itself is not updated. For INSERT statements, values given for any records other than the last record in the mapping are used for qualification of which parent or owner record to insert under.

In relational terms, these values are treated like foreign keys in an SQL database. If for example, the employee with SSN "012339920" does not exist in a record in the database, and SQLCODE -530 is returned indicating that an invalid foreign (positioning) key was supplied.

The same concepts hold true for UPDATE and DELETE statements. However, since both UPDATE and DELETE support WHERE clauses, foreign key qualification is placed in the WHERE clause itself.

In the case of update, SET statements must not include columns from any record other than the last record in the mapping. Otherwise, a -4903 SQLCODE is returned indicating that the statement is not supported.

## Insert Positioning

Positioning of inserted records is determined by any values supplied for records other than the last record in a mapped database path. In single record mappings and multiple record mappings where no path information is supplied, the positioning of new records is determined by the underlying database system. Before allowing INSERT access on a mapped table, consider the underlying insert behavior to ensure that unqualified insert positioning by the database will produce the desired results.

**NOTE:** Insert with subselects are not supported for multiple-record mapping.

---

## Data Defaulting in Database Records on INSERT

Database record mappings can include columns for all or only part of the underlying database record. When inserting records containing partial mappings, all areas of the database record not mapped are initialized to binary zeros.

In cases where mapped columns from a target database record are omitted from an insert statement, the underlying data in the record is initialized as follows:

- NULL IS specification (if supplied), or one of the following:
  - SPACES for underlying data type C,
  - ZERO for numeric datatypes,
  - Binary zeroes for VARCHAR data types.

## Update and Delete Behavior

The SQL UPDATE and DELETE statement applies changes to all database records that meet the WHERE clause criteria specified in the statement. Care must be taken to correctly specify WHERE criteria because unqualified UPDATES and DELETES change ALL instances of the target record in the database.

## Update and NULL Records

SQL UPDATE statements only update existing records in the database. When retrieving rows for tables mapped to multiple database records, SQL rows are returned even when the last record mapped in the table does not exist in the database within the mapped path. UPDATE statements will not be applied to SQL rows where the last record is returned as all NULLs due to the lack of a database record in the database.

## Mappings Containing Record Arrays

SQL Update is not supported on tables containing record array mappings of OCCURS clauses. If you need to update records containing multiple occurrences of data items, each occurrence must be mapped as a separate column in order for updates to be allowed.

SQL INSERT on tables that map items or overlapping fields can produce unpredictable results due to initialization of unspecified INSERT columns in the underlying database record. This is particularly true when overlapped columns are

of different SQL data types. It is recommended that you do not use SQL INSERT on table mappings containing group items or overlapping fields.

## Group Items and Overlapping Fields

SQL INSERT on tables that map group items or overlapping fields can produce unpredictable results due to initialization of unspecified INSERT columns in the underlying database record. This is particularly true when overlapped columns are of different SQL data types. CrossAccess strongly recommends against using SQL INSERT on table mappings containing group items or overlapping fields.

## General Recommendations

Usually, separate table mappings geared to update processing and securing access rights are better than non-update mapped tables using catalog security. Make sure update mappings are complete and that all access behavior produces the desired result.

Unless you have specific reason to do otherwise, for any tables being configured for UPDATE processing, use the NULL IS parameter on a column to define the string with the same length as the column definition.

For more information on securing access to mapped tables, see [Chapter 7, “SQL Security.”](#)

# Adabas Update Considerations

If a single query will be updating a large number of records, you must ensure that the Adabase parameter NISNHQ is large enough to handle the number of records.

## CA-DATACOM/DB Update Considerations

CA-DATACOM/DB tables may be modified (updated) using SQL statements specifying INSERT, DELETE, or UPDATE. To modify a table, the User



Requirements Table in use at the time of the modification must allow modifications to the specified table. This is done by including the parameter UPDATE=YES on the DBURTBL macro for the table being modified. For more information about the DBURTBL macro, see the *CA-DATACOM/DB 9.0 Database and System Administrator Guide*, topic “34.6.3 Entry Macro (DBURTBL).”

SQL modification subsequently requires either a COMMIT or ROLLBACK to delineate a unit of recovery. COMMIT and ROLLBACK may be either explicit or implicit. An explicit COMMIT or ROLLBACK can be issued by the program or script being executed. An implicit COMMIT is generated by the database at the normal end of the process. An implicit ROLLBACK is generated by the database at the abnormal end of the process.

To enable the ROLLBACK (Transaction Backout) action, several additional steps are required. These steps are CA-DATACOM/DB-specific and are outlined below.

**NOTE:** The following discussion pertains only to Transaction Backout (ROLLBACK). It does not encompass issues of Recovery within CA-DATACOM/DB. To implement full CA-DATACOM/DB recovery, reference the appropriate documentation. To better understand Transaction Backout, see the *CA-DATACOM/DB 9.0 Database and System Administrator Guide*, topic “41 Using Transaction Backout.”

**To enable a Transaction Backout in CA-DATACOM/DB, follow these steps:**

1. Enable the CA-DATACOM/DB logging system. For information about using logging, see the *CA-DATACOM/DB 9.0 Database and System Administrator Guide*, topics “39 Using Logging” and “39.1 Implementing Logging.”
2. Identify which tables may be Transaction Backout candidates. Transaction Backout requires logging of all update (INSERT, DELETE, UPDATE) transactions affecting a specific table.
3. Specify LOGGING=YES in the CA-DATACOM/DB Datadictionary for all tables identified by step 2, and catalog the definitions to the CA-DATACOM/DB Directory (CXX). For additional information, see the *CA-DATACOM/DB 9.0 Database and System Administrator Guide*, topic “23.2 Logging and Recovery” and the *CA-DATACOM/DB 9.0 Datadictionary User Guide*.
4. Specify TXNUNDO=YES in the DBURSTR macro for all User Requirements Tables used by programs that require transaction backout. For information about the DBURSTR macro, see the *CA-DATACOM/DB 9.0 Database and System Administrator Guide*, topic “34.6.2 Start Macro (DBURSTR),” and the *CA-DATACOM/DB 9.0 Programmer Guide*, topic “7.1.2 DBURSTR - Start User Requirements Table.”
5. Assemble and link edit all affected User Requirements Tables.

To verify the correct function of ROLLBACK, you must execute an SQL statement to modify (INSERT, DELETE or UPDATE) a CA-DATACOM/DB table, and follow it with an explicit ROLLBACK statement. The ROLLBACK process issues message DB00103I, containing a return code (RC=) value on the

system log. Return code RC=Y indicates the ROLLBACK completed successfully after backing out modification data. You can find complete information about message DB00103I in the *CA-DATACOM/DB 9.0 Message Guide*, topic “2.1 CA-DATACOM/DB Processing Messages (DB0xxxxc).”

Another way to verify is to query (SELECT) the row before modifications and after the ROLLBACK and compare the results. They should be identical.

## DB2 Update Considerations

Since eXadas SQL is essentially the same as DB2 SQL, there are no special considerations in updating DB2 data other than ensuring the eXadas Server has the appropriate access authority to the DB2 tables to be updated.

If you are running the DB2 thread management exit CACSX07, then DB2 table update authority must be granted to each user ID connecting to the eXadas Server. Otherwise, update authority must be granted to the eXadas Server task itself.

DB2 access errors due to authorization problems will return a -9999 SQLCODE and a message that the DB2 specific error is in the SQLEXT field. The SQLEXT field itself will contain the value E551 indicating a -551 error.

## IDMS Update Considerations

The IDMS Data Savant supports updating IDMS data using standard SQL update statements to the eXadas data server. All updates issued to IDMS mapped tables apply *only* to the last record mapped in the database path. See “[SQL Update and Mapped Tables](#),” on page 357, for more information.

IDMS insert and delete statements are restricted to records that are members of no more than one set. An attempt to insert or delete a record belonging to more than one set will result in a -9999 SQLCODE and the message that the IDMS specific error is in the SQLEXT field. The SQLEXT field itself will contain the value 1225, indicating a set occurrence has not been established for each set in which the record belongs.

---

## General IDMS Update Considerations

Navigating IDMS records requires issuing an @READY of database areas for access purposes. The areas readied in processing a mapped table are determined by the IDMS records mapped in the table as well as the system indexes defined for the first and last IDMS records in the mapping.

In general, all areas associated with the mapped path and indexes on the first and last record are readied in preparation for retrieving data. In update requests, these areas are readied with update intent. Depending on the complexity of the schema and the number of mandatory and automatic sets associated with the last record in a mapping, SQL INSERT and DELETE operations may not be possible in some cases.

## SQL INSERT Considerations

An SQL INSERT on an IDMS table results in an @STORE of the last record type mapped in the table. When IDMS processes a store operation, it implicitly connects the stored record to all sets in which the record is defined as an AUTOMATIC member. In order to automatically connect stored records, a record position for each OWNER record type must be established for each AUTOMATIC set to which the stored record belongs.

If more than one record is mapped in the path, and if the last record is a member of multiple AUTOMATIC set relationships (as is common in junction records), then every OWNER record of the last record must be mapped in the defined path. In addition, the set names between all records in the path should be defined as `_NONE_` so positioning can be done on each OWNER record independent of whether the record belongs to any particular set.

When mapping OWNER records, elements making up the CALC key of each OWNER record should be mapped for positioning purposes. Specifying CALC KEY mapped elements in the INSERT clause prevents area scans from occurring when positioning on OWNER records. Other elements may also be mapped for positioning qualification purposes if necessary.

The order of OWNER records in a table mapped for INSERT purposes does not matter. However, if any OWNER record lacks a CALC KEY and has a usable system index for positioning purposes, that OWNER should be the first record in the mapping as the use of system indexes is supported in the first record of a mapping only.

See [“Mappings Containing Multiple Records,” on page 357](#), for more information on INSERT positioning. Attempts to INSERT records without establishing required automatic OWNER positions results in a -9999 SQLCODE with an SQLEXT value of 0x005700F7.

Before attempting to use SQL to insert IDMS records, review the IDMS Schema definition of the candidate record to determine the sets for which the record is an AUTOMATIC member.

## SQL DELETE Considerations

An SQL DELETE on an IDMS table results in an @ERASE PERMANENT of the last record type mapped in the table. IDMS automatically cascades the ERASE statement to members of mandatory sets and disconnects members of optional sets when issuing this type of ERASE. Some records in the schema may not be deletable due to the number of set relationships they participate in and the amount of cascading necessary to accomplish the delete.

# IMS Update Considerations

The IMS Data Savant supports updating IMS data using standard SQL update statements to the eXadas server. All updates issued to IMS-mapped tables apply ONLY to the leaf segment mapped in the database path. See [“SQL Update and Mapped Tables,” on page 357](#), for more information.

**WARNING:** CrossAccess does not recommend updating an IMS database that is accessing the DBD through a secondary index, especially inverted hierarchies.

IMS updates are supported in both the DRA (DBCTL) and DBB/BMP access modes. However, in DBB/BMP mode, you must manage updates as client connections attempting an update enqueue the whole PSB for update purposes. This will lock out IMS access for all other users. Users locked out of the PSB will receive a -9999 SQLCODE and the error message: “ALL PCBS are in use.” For this reason, CrossAccess recommends using DRA mode for updating IMS data.

If you plan to update IMS databases in DBB mode, the eXadas server requires an IMS database log (IEFRDER) allocated to disk storage and the IMS BKO parameter set to Y in the server JCL. This configuration allows database changes to be backed out in the event of a client rollback request. If a DBB is started without database backout support, attempts to update the database are failed with a -5701659 SQLCODE and the message: “Update not supported by Data Savant.”

**NOTE:** IMS may change the JOBSTEP name for an eXadas Server when running IMS access with an IMS log file. In these cases, operator console commands to the Server must be issued to the JOBSTEP name created by IMS. Because the databases are being allocated by the DBB Server, the user ID associated with the Server requires CONTROL authority in RACF.

---

## IMS PSB Considerations

Creating PSBs for use with eXadas servers requires planning much like defining PSB's for any IMS batch or online applications. Unlike batch or online programs in which database and segment sensitivity and access options are based on a specific application program, PSB requirements for an eXadas data server are determined by:

- DBB versus DRA access.
- Table mappings (segment sensitivity).
- JOIN requirements against one or more databases.
- Query and Update requirements within an eXadas data server transaction (PCB PROCOPT).

If you plan on accessing IMS data using DBB or BMP access, then the access PSB must have enough PCBs to support all users of the server accessing the database at a single point in time. In general, the number of PCBs required for each database should minimally equal the maximum number of Query Processor services running in the Server at any point in time.

For more information on PSB definition guidelines, see [Chapter 2, “Deploying Applications.”](#)

## Update and Non-Update SQL Requests in a Single Transaction

If your application needs to issue both SQL query and update requests to IMS databases in a single transaction, all mapped tables must be accessible through a single PSB. SQL queries that take place after an update request automatically use the scheduled update PSB for accessing the IMS data. If a PCB necessary to satisfy the query request is not found, a -9999 SQLCODE is returned with the message: “Cannot access a PCB for the database requested.” For more information about transactions, see [“Transactions,” on page 354.](#)

## PCB Processing Options

When mapping tables for IMS update, verify all PSBs defined in the mapping grammar have the correct PCB processing options to insert, update, and delete IMS segments. Failure to do so will result in -9999 errors and the message: “Unexpected IMS status code received.”

---

# CICS VSAM Update Considerations

When eXadas accesses and modifies CICS VSAM data, it can perform all of the same database functions as it can when accessing and modifying native VSAM data. As when directly accessing VSAM data sources, you can SELECT, INSERT, UPDATE, and DELETE from a CICS VSAM data source. Supported VSAM file types are ESDS, KSDS, RRDS.

The main difference between performing database functions on native VSAM and CICS VSAM data is that CICS provides logging, which in turn provides support for transactional capabilities. The CICS VSAM files have to be set as recoverable to participate. With this support for transactions, you can perform COMMIT and ROLLBACK operations.

VSAM limitations are applicable to CICS VSAM. For example, due to the flat file format of ESDS, there is no DELETE functionality through VSAM access. This would also include no support with the ROLLBACK function on an ESDS file because it performs a delete on INSERT. Also, there is no support for INSERT on an RRDS file, as we do not yet provide an exit to fabricate a Relative Record Number.

CICS also supports the use of alternate indexes defined against a VSAM ESDS or KSDS data set. SELECT statements can be issued against the alternate index for both ESDS and KSDS data sets. UPDATE and DELETE statements can be issued against the alternate index when it is defined for a KSDS data set. UPDATE and DELETE statements referencing an alternate index on an ESDS data set are not supported.

The access and update considerations discussed in this topic also apply to IAM files. IAM (Innovation Access Method) is supplied by Innovation Data Processing and is a reliable, high-performance disk file manager that can be used in place of VSAM KSDS and ESDS data sets. Innovation Data Processing also provides an optional Alternate Index feature that is supported by eXadas. An exception is any reference to VSAM RRDS support, which currently are not supported by IAM.

## Transport Protocol

eXadas uses LU6.2 to transmit and return data between the Data Savant and the provided CICS transaction, “EXV1.”

Likewise, the Meta Data Utility uses LU6.2 to extract VSAM file information like record length, operation authorizations, and so on from the CICS VSAM data sources, using the provided CICS transaction, “EXV2.”

---

## Flow of Interactions

The Meta Data Utility starts a CICS EXV2 transaction and sends it a filename. The EXV2 CICS transaction issues an EXEC CICS INQUIRE FILE. If the file is not opened, it issues an EXEC CICS SET FILE OPEN and reissues the EXEC CICS INQUIRE FILE. Information such as MaxRecLen, KeyLen, KeyOff, and file type is sent back.

The eXadas Data Server starts a EXV1 CICS transaction for each table open. It sends an “Open” command to the EXV1 transaction, which gets information about the file. Depending on the query, commands like Seek, Search, Read Next, Insert, Update, Delete, Commit, and Rollback are sent to the EXV1 transaction, which issues the EXEC CICS native calls.





# Using Field Procedures

## Introduction to Using Field Procedures

This chapter describes how to use field procedures to transform data values from one format to another of your specification. The following topics are covered:

- [“Specifying a Field Procedure,” on page 370,](#)
- [“When Exits are Taken,” on page 370,](#)
- [“Execution Environment,” on page 371,](#)
- [“Field-Encoding \(Function Code 0\),” on page 374,](#)
- [“Field-Decoding \(Function Code 4\),” on page 376,](#) and
- [“Sample Field Procedures,” on page 378.](#)

A sample field procedure is included at the end of this chapter. A table containing the names of a sample macro that maps field procedure values is included in the DB2 macro library SDSNMACS and is named DSNDFPPB.

Field procedures are assigned to columns as Conversion Exits in the DataMapper and Meta Data Utility. The specified field procedure is invoked every time that column is referenced.

The transformation the field procedure performs on a value specified in a WHERE clause is called **field-encoding**. The same routine is used to undo the transformation when values are retrieved; that operation is called **field-decoding**. Values in columns with a field procedure are described by the SQL data types as defined by the DataMapper.

**WARNING:** The field-decoding function must be the exact inverse of the field-encoding function. For example, if a routine encodes ALABAMA to 01, it must decode 01 to ALABAMA.

## Specifying a Field Procedure

To create a field procedure for a column, use either of the following methods:

- Use the DataMapper to define a conversion exit for a column. The conversion exit will be included in the Meta Data Grammar used to create the Meta Data Catalog.
- Manually specify a field procedure in a column definition by using the WITH CONVERSION parameter. For more information on the WITH CONVERSION parameter, see [“Column Definitions,” on page 161](#).

Once a field procedure has been defined, the load module must also be placed in a library referenced by the Server STEPLIB DD statement. The field procedure routine must be written and linked as AMODE 31, RMODE ANY. Any storage or other resources that the field procedure allocates must be freed each time the field procedure is called.

## When Exits are Taken

Field Procedures are conversion routines that translate between data items in a database record and their corresponding SQL data types. These procedures are used when the underlying data type in a record does not match the SQL datatype

to be returned in SQL requests. Some examples of situations in which field procedures can be used include:

- Database fields transformed to reverse or change sorting order. For example, decimal date fields are often stored in 9s compliment format in order to reverse the sorting order of dates such that the most recent date comes first.
- Abbreviations or coded tables. Abbreviations or codes for application data items may be used to save space in database records. For example, the value “01” may be stored in place of the state name “Alabama.”
- Encryption. For security purposes, password fields may be stored in a database record in encrypted format.

Field procedures specified for a column are either encoded or decoded. Field encoding takes place when:

- the field is going to be compared to a value specified in the WHERE clause using the equal or not equal comparison operators.
- It can also take place where a column is assigned a value with an SQL INSERT or UPDATE statement.

Field-decoding takes place when:

- a stored value is to be compared to any operator other than equal or not equal.
- A value is also decoded during retrieval of the field in a select list. field-decoded back into its original string value.

A field procedure is never invoked to determine if the field is NULL or NOT NULL.

**WARNING:** Use of a key column with a field procedure in a WHERE clause may cause the Data Savant to scan the database. This can happen if the column is the beginning of the key and the comparison operator is not the equal comparison operator.

## Execution Environment

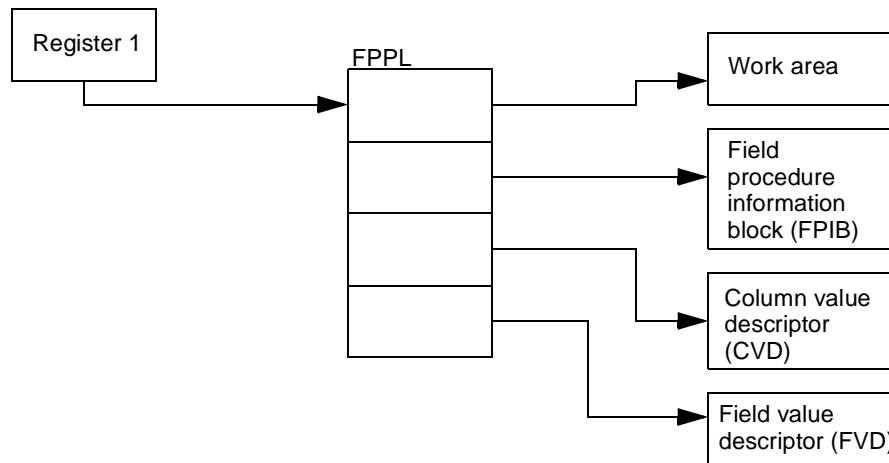
This section describes the control blocks used to communicate with a field procedure.

### The Field Procedure Parameter List (FPPL)

The field procedure parameter list is pointed to by register 1 on entry to a field procedure. It, in turn, contains the addresses of four other areas, as shown in the

following figure. Those areas are described on the following pages. The FPPL and the areas it points to are all described by the mapping macro DSNDFPFB.

**Figure 55: Field Procedure Parameter List**



## The Work Area

The work area is 512 bytes of contiguous uninitialized storage that can be used by the field procedure as working storage.

If 512 bytes is sufficient for your operations, your field-definition operation does not need to change the value supplied by the Query Processor. If less than 512 bytes is required, the field-definition can return a smaller value. If your program requires more than 512 bytes then the field procedure will have to acquire the necessary storage.

## The Field Procedure Information Block (FPIB)

The field procedure information block communicates general information to a field procedure. For example, it tells what operation is to be done, allows the field procedure to signal errors, and gives the size of the work area. The format is

shown in the following table.

**Table 65: Format of FPIB**

Value Description	Hex Offset	Integer Type	Data Type
FPBFCODE	0	Signed 2-byte integer	Function code: <ul style="list-style-type: none"> <li>• 0 (field-encoding).</li> <li>• 4 (field-decoding).</li> </ul>
FPBWKLN	2	Signed 2-byte integer	Length of work area is 512 bytes.
FPBSORC	4	Signed 2-byte integer	Reserved.
FPBRTNC	6	Character, 2 bytes	Return code set by field procedure.
FPBRSNC	8	Character, 4 bytes	Reason code set by field procedure.
FPBTOKP	C	Address	Address of a 40-byte area in which to return an error message.

**NOTE:** Error messages placed in the FPBTOKP are not returned to the client application when a field procedure reports an error. It is, however, logged in the Server log for debugging purposes.

## Value Descriptors

A value descriptor describes the data type and other attributes of a value. During field-encoding and field-decoding, the decoded (column) value and the encoded (field) value are described by the column value descriptor (CVD) and the field value descriptor (FVD).

The **column value descriptor** (CVD) contains a description of a column value and, if appropriate, the value itself. During field-encoding, the CVD describes the value to be encoded. During field-decoding, it describes the decoded value to be supplied by the field procedure.

The **field value descriptor** (FVD) contains a description of a field value and, if appropriate, the value itself. During field-encoding, the FVD describes the encoded value to be supplied by the field procedure. During field-decoding, it describes the value to be decoded.

Both value descriptors have the format shown in the following table.

**Table 66: Format of Value Descriptors**

Value Descriptor	Hex Offset	Integer Type	Data Type
FPVDTYP	0	Signed 2-byte integer	Data type of the value: <ul style="list-style-type: none"> <li>• 0 (INTEGER)</li> <li>• 4 (SMALLINT)</li> <li>• 8 (FLOAT)</li> <li>• 12 (DECIMAL)</li> <li>• 16 (CHAR)</li> <li>• 20 (VARCHAR)</li> <li>• 24 (GRAPHIC)</li> <li>• 28 (VARGRAPHIC)</li> </ul>
FPVDVLEN	2	Signed 2-byte integer	<ul style="list-style-type: none"> <li>• For a varying-length string value, its maximum length</li> <li>• For a decimal number value, its precision (byte 1) and scale (byte 2)</li> <li>• For any other value, its length</li> </ul> <p><b>NOTE:</b> For GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC data types, the length is specified in bytes.</p>
FPVDVALE	4	None	The value of the column or field. If the value is a varying-length string, the first Halfword is the value's actual length in bytes. This field is not present in a CVD, or on an FVD used as input to the field-definition operation. An empty varying-length string has a length of zero with no data following.

## Field-Encoding (Function Code 0)

This section describes input provided to the field-encoding operation and the output required.

On ENTRY, the registers have the following information:

**Table 67: Field Encoding on Entry**

Register	Contents
1	Address of the field procedure parameter list (FPPL); see <a href="#">Figure 55: “Field Procedure Parameter List,”</a> on page 372, for a schematic diagram.
0, 2 through 12	Values must be restored on exit.
13	Address of the calling program’s save area. Must be restored on exit.
14	Return address.
15	Address of entry point of exit routine.

The **work area** is uninitialized and is 512 bytes.

The FPIB has the following information.

**Table 68: PFIB Fields and Contents**

Field	Contents
FPBFCODE	0, the function code
FPBWKLN	the length of the work area is 512.

The CVD has the following information.

**Table 69: CVD Fields and Contents**

Field	Contents
FPVDTYPE	The numeric code for the data type of the column value, as shown in <a href="#">Table 66, “Format of Value Descriptors,”</a> on page 374.
FPVDVLE N	The length of the column value.
FPVDVAL E	The column value; if the value is a varying-length string, the first Halfword contains its length.

The FVD has the following information.

**Table 70: FVD Fields and Contents**

Field	Contents
FPVDTYPE	The numeric code for the data type of the field value.
FPVDVLEN	The length of the field value.
FPVDVALE	Uninitialized area with a value of FPVDVLEN.

On EXIT the registers have the following information.

**Table 71: EXIT Registers and Contents**

Register	Contents
1	Address of the field procedure parameter list (FPPL); see <a href="#">Figure 55: “Field Procedure Parameter List,”</a> on page 372, for a schematic diagram.
0, 2 through 14	The values they contained on entry.
15	The integer zero. An error in processing should be indicated by the value in FPBRTNC (see <a href="#">Table 78, “FPIB Fields and Contents,”</a> on page 378).

The FVD must contain the encoded (field) value in field FPVDVALE. If the value is a varying-length string, the first halfword must contain its length.

The FPIB can have the following information.

**Table 72: FPIB Fields and Contents**

Field	Contents
FPBRTNC	Return code. The character “0” followed by a space indicates success. Anything other than “0 ” indicates an error.
FPBRSNC	An optional 4-byte character reason code, defined by the field procedure; blanks if no reason code is provided.
FPBTOKP	Address of a 40-byte area containing an error message when an error message is detected.

Errors signaled by a field procedure result in SQLCODE -681, and are written to the error log, which is set in the SQL Communications Area (SQLCA). FPBRTNC is the return code, FPBRSNCD is the reason code, and a 40 byte error field for the specific error message is FPBTOKP.

## Field-Decoding (Function Code 4)

The input provided to the field-encoded operation, and the output required, are described in the tables that follow.



On ENTRY, the registers have the following information:

**Table 73: ENTRY Registers and Fields**

Register	Contents
1	Address of the field procedure parameter list (FPPL); see <a href="#">Figure 55: “Field Procedure Parameter List,”</a> on page 372, for a schematic diagram.
0, 2 through 12	Values must be restored on exit.
13	Address of the calling program’s save area. Must be restored on exit.
14	Return address.
15	Address of entry point of the exit routine.

The **work area** is an uninitialized 512 byte area.

The FPIB has the following information.

**Table 74: FPIB Fields and Contents**

Field	Contents
FPBFCODE	4, the function code.
FPBWKLN	The length of the work area is 512 bytes.

The CVD has the following information.

**Table 75: CVD Fields and Contents**

Field	Contents
FPVDTYPE	The numeric code for the data type of the column value, as shown in <a href="#">Table 66, “Format of Value Descriptors,”</a> on page 374.
FPVDVLEN	The length of the column value.
FPVDVALE	Uninitialized area with a value of FPVDVLEN.

The FVD has the following information.

**Table 76: FVD Fields and Contents**

Field	Contents
FPVDTYPE	The numeric code for the data type of the field value.
FPVDVLEN	The length of the field value.
FPVDVALE	The field value; if the value is a varying-length string the first Halfword contains its length.

On EXIT the registers have the following information.

**Table 77: EXIT Registers and Contents**

Register	Contents
1	Address of the field procedure parameter list (FPPL); see <a href="#">Figure 55: “Field Procedure Parameter List,”</a> on page 372, for a schematic diagram.
0, 2 through 14	The values they contained on entry.
15	The integer zero. An error in processing should be indicated by the value in FPBRTNC (see <a href="#">Table 78, “FPIB Fields and Contents,”</a> on page 378).

The CVD must contain the decoded (column) value in field FPVDVALE. If the value is a varying-length string, the first Halfword must contain its length.

The FPIB can have the following information.

**Table 78: FPIB Fields and Contents**

Field	Contents
FPBRTNC	Return code. The character “0” followed by a space indicates success. Anything other than “0 ” indicates an error.
FPBRSNC	An optional 4-byte character reason code, defined by the field procedure; blanks if no reason code is provided.
FPBTOKP	FPBTOKPAddress of a 40-byte area in which to return an error message.

Errors signaled by a field procedure result in SQLCODE -681, and are written to the error log, which is set in the SQL Communications Area (SQLCA). FPBRTNC is the return code, FPBRSNCD is the reason code, and a 40 byte error field for the specific error message is FPBTOKP.

## Sample Field Procedures

This section describes the sample field procedures shipped with eXadas.

Sample field procedures are provided in SCACSAMP for various data types. The following table lists the member and the associated data type.

**Table 79: SCACSAMP Members and Associated Data Types**

SCACSAMP Member Name	Description
CACFP001	A 9s complement field procedure used to convert decimal or zoned decimal data.
CACFP999	Used to create site-specific field procedures.

Both types are described in the sections that follow. See SCACSAMP for sample coding for these field procedures.

## Sample Field Procedure CACFP001

This sample field procedure is a 9s complement field procedure. It is designed for use on decimal or zoned decimal data. The following three mapping combinations are supported:

- DATATYPE P|UP USE AS DECIMAL(p,s) in which decimal fields are mapped as SQL DECIMAL.
- DATATYPE C|UC USE AS CHAR(n) in which zoned decimal fields are mapped as SQL CHAR.
- DATATYPE C|UC USE AS DECIMAL(p,s) in which zoned decimal fields are mapped as SQL DECIMAL.

This example is a common way to force decimal date fields to have descending sort order in databases.

## Sample Field Procedure CACFP999

This field procedure is generic and can be used as a model for creating your own site-specific field procedures. It copies field data directly to column data as-is and supports the following data types:

- SMALLINIT,
- INTEGER
- DECIMAL,
- FLOAT, and
- CHARACTER.

**NOTE:** This field procedure does not do any data conversion.



# Appendix A

## Configuration Parameters

### Introduction to Configuration Parameters

This appendix contains the format, relationships, and descriptions of the eXadas configuration parameters. For specific information regarding the use of these parameters to configure the Server, see the Concepts chapter in the *eXadas OS/390 Getting Started Guide*.

The following sections are discussed:

- [“Configuration Parameter Format,”](#) on page 382,
- [“Configuration Parameter Relationships,”](#) on page 383, and
- [“Configuration Parameter Descriptions,”](#) on page 385.

# Configuration Parameter Format

Configuration parameters consist of fixed-length 80-byte records containing either a parameter starting in column 1 or a comment, represented as an asterisk (\*), in column 1. The parameter syntax follows.

Example:

```
parameter name = value
```

In this example,

- Parameter name is one or more keywords beginning in the first column of the record.
- A blank must exist on both sides of the equal sign.
- Value is any number of characters up to the end of the record.
- String values are not surrounded by delimiters.
- Comments after the value are not allowed.

The maximum parameter length is 255 characters, but parameters may be continued across 80-byte records by using the backslash (\) as a continuation character. The continuation character may not be used until after the equal sign, and must be the last non-blank character of the record. The backslash character will be discarded, as will leading blanks on the continued record. Comment lines may be inserted between the continued records.

When editing configuration data sets, do not insert sequence numbers at the end of the records because they become part of the value assigned to the corresponding keyword.

**NOTE:** At any level, if a configuration file contains invalid parameters then that entire configuration image is not loaded into memory. If the Master Configuration File cannot be loaded, the Server (or eXadas Enterprise Server) will terminate. The Server will not terminate on failing to load either the Query Processor level configuration or the User level configuration. In these cases the configuration in effect is the next higher configuration that was previously loaded successfully. Worst case is that the Master Configuration would be used. For example, if user "TOM" connects but his client configuration file is missing or invalid, then his connection will revert to using the Query Processor configuration (if one has been loaded successfully) or the Master Configuration.

If ISPF is used, make sure that NUM OFF is set in the edit profile when editing configuration members.

# Configuration Parameter Relationships

Configuration parameters are classified into the following general categories:

- Client configuration parameters,
- Server configuration parameters,
- Client Configuration Member Override parameters, and
- Query Processor Configuration parameters.

The parameters for each of the categories may be related to, and dependent upon, one another. The parameters are shown in the table that follows and described in detail in [“Configuration Parameter Descriptions,” on page 385](#).

The following table lists the configuration parameters, the configuration member type in which the parameter is specified, and whether the parameter is required (R), optional (O), or not applicable (left blank).

**Table 80: Configuration Parameter Classifications**

Parameter	Server Master	Query Processor	User Config. Member Override	Client
BTREE BUFFERS	O	O	O	
CPU GOVERNOR	O	O	O	
DATASOURCE				R
DECODE BUFFER SIZE				O
DEFLOC				O
FETCH BUFFER SIZE				O
INTERLEAVE INTERVAL	O	O	O	
JOIN MAX TABLES ANALYZED	O	O	O	
LD TEMP SPACE	O	O	O	O
LOCALE	O	O	O	O
MAX ROWS EXAMINED	O	O	O	
MAX ROWS EXCEEDED ACTION	O	O	O	
MAX ROWS RETURNED	O	O	O	
MESSAGE POOL SIZE	O			O

**Table 80: Configuration Parameter Classifications**

<b>Parameter</b>	<b>Server Master</b>	<b>Query Processor</b>	<b>User Config. Member Override</b>	<b>Client</b>
NL	R			R
NL CAT	R			R
OME	O	O	O	
PDQ		O	O	O
RESPONSE TIME OUT				O
SAF EXIT	O			
SERVICE INFO ENTRY	R			
SMF EXIT	O		O	
STATIC CATALOGS	O			
STATEMENT RETENTION	O	O		
TASK PARAMETERS	O			O
TRACE LEVEL				O
USER CONFIG	O	O		
USERID				O
USERPASSWORD				O
VSAM AMPARMS	O	O	O	
WLM UOW	O	O	O	



---

# Configuration Parameter Descriptions

The following is an alphabetical listing of configuration parameters and their descriptions.

Parameter defaults are used when the parameter is not specified in the relevant configuration file.

Invalid parameters will cause the file to be ignored. eXadas does not issue a message indicating that the default value was applied or that a value in the configuration file is invalid.

## BTREE BUFFERS

Description: **Optional parameter** used to determine the number of in-memory B-tree buffer caches that will be used before staged result set output is spooled to data spaces, or physical files. By default, four in-memory buffers are used before the data is written out to a dataspace or physical file.

The BTREE BUFFERS parameter is used to override the default value of four. If sufficient memory is available in the MESSAGE POOL SIZE memory pool, this parameter can be increased and performance may improve depending on the size of the result set and if the result set is ordered or grouped.

Example:

```
BTREE BUFFERS = 4
```

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 214730

Minimum permitted value: 4

Default: 4

Use: Server, Query Processor, Client Config. Member Override, Client

## CPU Governor

Description: **Optional parameter** used to specify the name and time limit of the exit used to implement a CPU resource governor. If this parameter is omitted, there is no limit to the amount of CPU time that can be used to process a query. For more information, see [Chapter 15, “System Exits.”](#)

Example:

```
CPU GOVERNOR = CACSX03 5M
```

Maximum permitted value: 255M

Minimum permitted value: 1S

Default: none

Use: Server, Query Processor, Client Config Member Override

## DATASOURCE

Description: **Required parameter** that is used to specify the name of the data source a client is attempting to connect to. Field 1 is the name of the remote data source that matches the service name (field 2) of the SERVICE INFO ENTRY parameter for the corresponding Server's Query Processor task. Field 2 is the address field by which the client connects to the named data source. This field consists of three parts separated by the backslash (/) character and must match the task data field (field 10) of the Server's Connection Handler.

### Sample Address Field for TCP/IP Protocol with data source name, CACSAMP

- The first part of the field must be set to TCP.
- The second part of the field is the hostname (string) of the Server or the IP address of the Server. If an IP address is specified, it must be defined in dot notation (123.456.789.10).
- The third part of the field is the port number (decimal value), or service name on which the Server is listening for connection requests.

Example:

```
DATASOURCE = CACSAMP tcp/111.111.111.11/2222
```

### Sample Address field for Cross Memory Protocol with data source name, CACSAMP

- The first part of the field must be set to XM1
- The second part of the field is the data space name and has a maximum length of four characters. This must be the same as the data space name specified on the SERVICE INFO ENTRY of the Server.
- The third part of the field is the queue name and has a maximum length of four characters. This must be the same as the data space name specified on the SERVICE INFO ENTRY of the Server.

---

```
DATASOURCE = CACSAMP XML/CAC/CAC
```

### Sample Address Field for MQ Series Protocol with data source name, CACSAMP

- The first part of the field specifies that the Controller invoke the IBM MQ Series Transport Module.
- The second part of the field identifies the name of the MQ Series Queue Manager to connect to.
- The third part of the field identifies the name of the Model Queue that the client receives SQL responses on from the Server.
- The fourth part of the field identifies the name of the MQ Series Queue Manager. This must be the same as the first Queue Manager Name.
- The fifth part of the field identifies the name of the Local Queue that the Server is listening on for connection requests, or the name of a Remote Queue definition that is associated with the Local Queue that the Server is listening on for connection requests.

Example:

```
DATASOURCE = CACSAMP MQI/SCQ1/CAC.CLIENT/SCQ1/CAC.SERVER
```

Allowable value type: string

Representation: string

Maximum Permitted Value: 18 characters for data source name; 64 characters for address field

Minimum Permitted Value: 1 character for data source name; address field depends on the protocol

Default: None

Use: Client

## DECODE BUFFER SIZE

Description: **Optional parameter** that defines the size of the eXadas DECODE buffer, a staging area used to decode data from the network format into the host local data format. Data is taken from the FETCH buffer in pieces of the size specified for the DECODE buffer and converted until a single row of data is completely processed and returned to the application. For optimum usage, set the DECODE buffer to a size at least equivalent to a single row of data. The DECODE BUFFER SIZE and FETCH BUFFER SIZE parameters work together. If the DECODE BUFFER SIZE is omitted, its value is set to the value of FETCH BUFFER SIZE. If a value higher than the FETCH BUFFER SIZE is used, the value of DECODE BUFFER SIZE is set to the FETCH BUFFER SIZE.

**NOTE:** This parameter works closely with the `FETCH BUFFER SIZE` parameter. It is recommended that you coordinate the settings between the two parameters. See the discussion of the `FETCH BUFFER SIZE` parameter on [page 388](#).

Example:

```
DECODE BUFFER SIZE = 8096
```

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 64000

Minimum permitted value: 4096

Default: 8192

Use: client configuration

## DEFLOC

Description: **Optional parameter** that is the default `DATASOURCE` used if a `SELECT` or `CONNECT` statement does not specify where the data resides.

Example:

```
DEFLOC = CACSAMP
```

Allowable value type: string

Representation: string

Maximum permitted value: 18 characters

Minimum permitted value: 1 character

Default: none

Use: Client

## FETCH BUFFER SIZE

Description: **Optional parameter** that specifies the size of the result set buffer that is returned to a client application. This is specified in the client application's configuration file.

Regardless of the size of the fetch buffer specified, eXadas always returns a complete row of data in this buffer. Setting the fetch buffer size to 1 causes eXadas to return single rows of data to the client application.

Setting an appropriate `FETCH BUFFER SIZE` depends upon the average size of the result set rows that are sent to the client application and the optimum communication packet size. From a performance standpoint, you will want to pack as many rows as possible into a fetch buffer. The default fetch buffer size is generally adequate for most queries.

If the `FETCH BUFFER SIZE` is set smaller than a single result set row, then the size of the actual fetch buffer that is transmitted is based on the result set row size. The size of a single result set row in the fetch buffer depends on the number of columns in the result set and the size of the data returned for each column.

The following calculations can be used to determine the size of a result set row in the buffer:

```
fetch buffer row size = (number of data bytes returned) x
(number of columns * 6)
```

There is also a fixed overhead for each fetch buffer. This can be computed as:

```
fetch buffer overhead = 100 + (number of columns * 8)
```

If your applications are routinely retrieving large result sets you will want to contact your network administrator in order to determine the optimum communication packet size and set the `FETCH BUFFER SIZE` to a size that takes this into account. For more information on the `FETCH BUFFER SIZE` parameter, see [Chapter 9, "Optimization."](#)

Example:

```
FETCH BUFFER SIZE = 64000
```

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 64000

Minimum permitted value: 1

Default: 64000

Use: Client

## INTERLEAVE INTERVAL

**Description: Optional parameter.** This value sets the interleaving interval from the Query Processor. The unit of measurement for this interval is a result set row. When multiple results sets are being processed on the same Query Processor instance, the interleaving interval controls context switching between users and result sets. For example, if `INTERLEAVE INTERVAL` is set to 100 then the

Query Processor will context switch between active users on that instance for every 100 rows produced.

Maximum permitted value: 4294967295

Minimum permitted value: 0 (Disables context switching and has the same effect as setting PDQ to zero.)

Default: 100

Use: Server, Query Processor, Client Override

## JOIN MAX TABLES ANALYZED

Description: **Optional parameter** that determines the JOIN optimization method used in queries containing joins.

- Specifying a value of zero bypasses JOIN optimization logic completely and the query is processed as-is.
- Specifying a value of one indicates that the simple JOIN optimization algorithm will be applied to all SQL JOINS.
- Specifying a value greater than one identifies the maximum number of tables in the JOIN that are analyzed using the estimated number of reads algorithm.

The estimated read analysis evaluates all possible combinations of table processing order and therefore a value greater than four is not recommended (24 combinations). JOINS including more than the specified number of tables will automatically use the simple optimization method. See [Chapter 9, “Optimization,”](#) for additional information.

Example:

```
JOIN MAX TABLES ANALYZED = 4
```

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 15

Minimum permitted value: 0

Default: 4

Use: Server, Query Processor, Client Config. Member Override

## LD TEMP SPACE

Description: **Optional parameter** that defines a temporary data set dynamically allocated by a Server to store the intermediate result set. Temporary data set information is a set of parameters separated by commas. Parameters not specified are set to the defaults.

Set this parameter so the resulting file is large enough to hold any intermediate result sets that are generated from a typical query running under a particular Server. If your site has a storage unit name set up for VIO storage, specify VIO.

Example:

```
LD TEMP SPACE = ALCUNIT=TRK , SPACE=15 , VOL=CACVOL
LD TEMP SPACE = ALCUNIT=CYL , SPACE=2
LD TEMP SPACE = ALCUNIT=CYL , SPACE=2 , EXTEND=1 , UNIT=VIO
```

- ALCUNIT = BLOCK|TRK|CYL unit of space allocation specifying block, track, or cylinder. The default value is TRK.
- SPACE = nnn primary amount of space to allocate. The default value is 15.
- EXTEND = nnn secondary amount of space to allocate. The default value is 5.
- VOL = VOLSER (volume serial number). The default is the OS/390 default for this site.
- UNIT = unit name. The value may specify a DASD allocation group name, or the VIO group name, if it exists. The default unit name is the OS/390 default for this site.
- RECFM = F|V|U record format to allocate corresponds to OS/390 RECFM of FB|VB|U. Default is V.
- RECLEN = nnn record length. If variable format record, OS/390 LRECL will be set to RECLEN +4. Default is 255.
- BLKSIZE = nnn block size. Default is 6144.

Hiperspace is a feature that allows the placement of temporary data files, such as temporary files, spill files, and so on, in expanded storage. This results in improved performance. This performance improvement mostly affects complex queries, for example, ORDER BY. To specify Hiperspace, set the LD TEMP SPACE as follows:

```
LD TEMP SPACE = HIPERSPACE , INIT=16M , MAX=24M , EXTEND=8M
```

Where:

- INIT= initial region size for the hiperspace.
- MAX= maximum region size for the hiperspace.
- EXTEND= unit of growth when INIT is exceeded.

The estimate for determining these values is related to system installation limits and expected query types. Roughly, the maximum size should be equivalent to that of the regular temp space file as described for the non-hiperspace LD TEMP SPACE setting.

**NOTE:** Using hiperspace requires APF authorization.

Use: Server, Query Processor, Client Config. Member Override, Client

## LOCALE

Description: **Optional parameter** that is used to activate DBCS processing within the Server. The LOCALE parameter is used to activate a specific SAS/C processing locale and to activate special DBCS and mixed-mode processing code within the Server. When a LOCALE parameter is not supplied, the Server uses the standard SAS/C “C” locale and the Server uses standard memcmp comparison routines on operators and sorting operations, for example, an ORDER BY clause. If a LOCALE of DBCS is specified, it activates the SAS/C DBCS locale and special locale specific comparison functions are used for comparison purposes. When a LOCALE of C is specified it activates the standard SAS/C “C” locale functions and memcmp functions are used for comparison purposes.

**NOTE:** Activation of the DBCS locale affects Server performance and should only be turned on when the data being retrieved contains GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC data types, or when there is a mixture of DBCS and SBCS characters sets contained within CHAR, VARCHAR, or LONG VARCHAR data types. Even if the above conditions are true, then the DBCS locale only needs to be activated when the query result set is sorted, as in an ORDER BY clause, or when less/than greater/than operators are contained in the query’s WHERE clause.

Example:

```
LOCALE = DBCS
```

Allowable value type: string

Representation: string

Maximum permitted value: 4 characters

Minimum permitted value: 1 character

Allowed values and results:

- DBCS: Activate the SAS/C Double-Byte Character Set locale and perform comparisons using SAS/C locale specific comparison routines.
- C: Activate standard SAS/C Single-Byte Character Set locale and perform comparisons using SAS/C locale specific comparison routines based on “memcmp.”



---

Default: C - Single-Byte Character Set

Use: Server, Query Processor, Client Config. Member Override, Client

## MAX ROWS EXAMINED

Description: **Optional parameter** is one of the governor parameters used to implement the eXadas governor features. It is designed to protect against excessive resource use caused by inefficient or erroneous queries. The governor is implemented by placing a restriction that limits the number of rows examined. In the EXAMINATION phase, the restriction on the number of rows examined is put into effect after native retrieval is performed and before additional filtering takes place to satisfy any SQL WHERE clause specifications.

Example:

```
MAX ROWS EXAMINED = 0
```

Allowable value type: numeric

Representation: decimal

Max permitted value: 2147483647

Minimum permitted value: 0

Recommended value: 10000

Default: 0 (unlimited rows)

Use: Server, Query Processor, Client Config. Member Override

## MAX ROWS EXCEEDED ACTION

Description: **Optional parameter** used to determine the behavior of the eXadas governor feature when either the MAX ROWS EXAMINED or MAX ROWS RETURNED governor limits are reached. The MAX ROWS EXCEEDED ACTION parameter is used to test a query to ensure that the query is returning the correct result set and is not expending large amounts of resources, without running the full query.

Specification of the value of ABORT causes query execution to be terminated when either the MAX ROWS EXAMINED or MAX ROWS RETURNED governor limits are reached. In this situation, the end user will receive a -9999 return code, and no result set is returned. Specification of a value of RETURN causes the result sets retrieved to be returned to the client with a truncated result set.

**WARNING:** If you use RETURN, any result you get may not be complete and there is no indication that the governor limit was reached.

Example:

```
MAX ROWS EXCEEDED ACTION = ABORT
```

Allowable value type: string

Representation: string

Maximum permitted value: 6 characters

Minimum permitted value: 5 characters

Allowed values and results:

- ABORT: Stop the query when a governor limit is reached
- RETURN: Return a normal result set when a governor limit is reached

Default: ABORT

Use: Server, Query Processor, Client Config. Member Override

## MAX ROWS RETURNED

Description: **Optional parameter** is one of the governor parameters used to implement the eXadas governor. It is designed to protect against excessive resource use caused by inefficient or erroneous queries. The governor is implemented by placing a restriction that limits the number of rows returned. In the RETURN phase, a restriction limiting the number of rows returned is put into effect after any WHERE clause is fully processed, but before the result table is returned to the application.

Example:

```
MAX ROWS RETURNED = 0
```

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 2147483647

Minimum permitted value: 0

Recommended value: 10000

Default: 0 (unlimited rows)

Use: Server, Query Processor, Client Config. Member Override

---

## MESSAGE POOL SIZE

Description: **Required parameter** that specifies the size of the memory region used for all memory allocation. The number is specified in bytes. The actual workable maximum value should be set to 2MB less than the region size. If the value specified is less than 1MB, 1MB is used. If the amount of storage that can be obtained is less than the value specified, the maximum amount available is obtained.

Example:

```
MESSAGE POOL SIZE = 16777216
```

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 2147483648 (2GB)

Minimum permitted value: 1048576 (1MB)

Default: 1048575 (1MB)

Use: Server, Client

## NL

Description: **Required parameter** that specifies the language used for text messages produced by eXadas. The only valid value in this release for this parameter is US English, which corresponds to standard English as used in the United States.

Example:

```
NL = US ENGLISH
```

Allowable value type: string

Default: US ENGLISH

Use: Server, Client

## NL CAT

Description: **Required parameter** that points to the language catalog that contains eXadas messages in a specified language. It is defined by a DD statement in the start-up procedure.

Example:

```
NL CAT = DD:ENGCAT
```

**NOTE:** Different catalog file names may be specified, but the US English language catalog is the only one available in this release; therefore, the catalog **MUST** be defined as DD:ENGCAT.

Allowable value type: `string DD:`, followed by a character string or `string DSN`, followed by a data set name.

Representation: string

Default value: DD:ENGCAT

Use: Server, Client

## PDQ

Description: **Optional parameter** used to activate immediate return of data processing. When PDQ processing is activated, the Server inspects each query and determines whether the result set output requires additional post-processing. If post-processing is required, the result set is handled as if PDQ processing was not activated. If no post-processing is required, the result set is immediately returned to the client.

Example:

```
PDQ = 0
```

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 1

Minimum permitted value: 0

Allowed values and results:

- 0: do not analyze queries; the result set is staged in the B-trees for all queries processed by the Server.
- 1: analyze each query to determine whether data can be returned immediately.

**NOTE:** Setting this parameter to 1 will have no affect unless the OME parameter is set to something other than 0.

Default: 0

Use: Query Processor, Client Config. Member Override

---

## RESPONSE TIME OUT

**Description:** **Optional parameter** that specifies the response time-out. This value specifies the maximum amount of time that this service waits for an expected response before terminating a connection.

Valid formats include:

- nMS = number of milliseconds
- nS = number of seconds
- nM = number of minutes

Example:

```
RESPONSE TIME OUT = 10M
```

Allowable value type: numeric with alpha modifier

Representation: decimal

Maximum permitted value: 0 (never time out), 1000MS, 60S, and 60M respectively

Minimum permitted value: 0MS

Default: 6M

Use: Client

## SAF EXIT

**Description:** **Optional parameter** used to specify the SAF system exit that will perform authorization checks for the Data Savant associated with a Server or execute a stored procedure application program. The default is NONE. For more information, see [Chapter 15, “System Exits.”](#)

Example:

```
SAF EXIT = CACSX04 IMS CLASS=IMSP,PSB PREFIX=IMSP
```

Default: none

Use: Server

## SERVICE INFO ENTRY

Description: **Required parameter** used in Server configuration files to inform the Region Controller task that a service should be activated and how that service should be controlled. It is valid only in Server and Enterprise Server configuration files.

Multiple Service Info Entry parameters are required to activate multiple instances of a given service if different subparameter values are needed. A single Service Info Entry parameter is used if only a single instance is needed (or multiple instances using the same subparameter values). A given service's restrictions and allowable combinations of multiple instances are discussed in that service's specific descriptions. Mutually-exclusive services are also noted in these descriptions.

The Service Info Entry parameter consists of ten subparameters, each delimited by at least one space. The format of the first nine of these subfields is consistent across all services and is not service dependent. The format for the tenth subfield is service dependent, as well as all valid values of all ten fields.

A description of the meaning and format of the first nine subfields is presented first, followed by a separate description of each available service. These separate descriptions will give the valid values for all ten subfields and the valid format(s) for the tenth subfield. Related parameters, references to detailed discussions elsewhere in this guide, and any special circumstances will also be noted in these service specific descriptions. A table at the end of the description of the SERVICE INFO ENTRY parameter contains an entry for each valid service, and is provided for you to use as a quick reference chart.

The following table lists all of the subparameters, in order, along with a descriptive label of their contents.

**Table 80: SERVICE INFO ENTRY Fields**

Field	Label
1	Task Name
2	Service Name
3	Service Start Class
4	Minimum Tasks
5	Maximum Tasks
6	Max Connections/Task
7	Tracing Output Level
8	Response Time Out

**Table 80: SERVICE INFO ENTRY Fields**

Field	Label
9	Idle Time Out
10	Service Specific Information

Example:

```
SERVICE INFO ENTRY = CACQP CACSAMP 2 3 5 100 4 /
5M 5M NO_DATA
```

Where:

- Field 1, Task Name, is the name of the executable module to be invoked for this service. Valid values include:
  - CACCNTL - Region Controller,
  - CACLOG - Logger Service,
  - CACQP - Query Processor Service,
  - CACINIT - Connection Handler Service,
  - CACIMSIF - IMS BMP/DBB Interface,
  - CACDRA - IMS DRA Interface,
  - CACCAF - DB2 CAF Service,
  - CACWLM - Work Load Manager System Exit Initialization Service,
  - CACDSH - Data Source Handler Service (eXadas Enterprise Server only),
  - CACLE - Language Environment Initialization Service,
  - CACDCI - Datacom Initialization Service, and
  - CACVSMS - VSAM Service.

Allowable value type: string

Representation: maximum 8 characters, no spaces

- Field 2, Service Name, represents the name that identifies a given service. The value specified must be unique within a given Server's configuration file. Some services require a specific value, those that do not require that this value be referenced in another parameter. Required values and/or required correlations of this field are noted in the service specific descriptions. This value is also used in MTO commands to identify a specific service, such as in a START command. See [Appendix C, "MTO Command Reference,"](#) for more information.

Valid values: Noted under given service specific descriptions.

Allowable value type: string

Representation: maximum 18 characters, no spaces (although the field name may be longer than 18 characters, only the first 8 will display when the MTO DISPLAY command is issued).

- Field 3, **Service Start Class**, indicates to the Region Controller which server initialization phase a given service should be started in. The value must be 0 for the CACCNTL service and 1 for the CACLOG service. All other services must have a value of 2 for this field except the CACDSH service, which can have a value of 2 or 3. See [Chapter 17, “Enterprise Server,”](#) for information regarding the Enterprise Server and the CACDSH service.

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 3

Minimum permitted value: 0

- Field 4, **Minimum Tasks**, is a required field that specifies the minimum tasks. This value specifies the number of instances of this server that the Region Controller should start during Server initialization. If a service must be limited to a single occurrence, this field and field 5 must be set to 1. Setting this field to 0 indicates to the Region Controller that occurrences of this service should not be started at Server initialization. The MTO **START** command can be used to start an occurrence when needed.

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 65535

Minimum permitted value: 0

- Field 5, **Maximum Tasks**, specifies the maximum number of instances of this service that the Region Controller is allowed to start. All of these instances will use the same subparameter values. If a service must be limited to a single instance, this field must be set to 1.

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 65535

Minimum permitted value: 1

- Field 6, **Max Connections/Task**, specifies the maximum number of user connections that are allowed per instance of this service activated as a result of this SERVICE INFO ENTRY parameter. Setting this field to 1 disables multi-tasking for all instances of this service. Some services require this field be set to 1.

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 65535



Minimum permitted value: 0

- Field 7, **Tracing Output Level**, specifies the level of tracing messages a service will pass to the Logger service. This value is a minimum filtering level, meaning that setting this value to 4 causes messages of category 4 or higher to be passed.

For the logger service, this field specifies the trigger value. This means that when a tracing message of this level or higher is passed to the logger service, the trace buffer and all subsequent trace messages are written to the log file. For more logger service information, see [Chapter 12, “Server Logging.”](#)

Valid values include:

- 1 = Full Tracing
- 2 = Message Buffers
- 3 = Procedural/Query Processor Instructions
- 4 = Informational/Warning Messages
- 8 = Fatal Error Messages
- 20 = No Tracing

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 20

Minimum permitted value: 1

- Field 8, **Response Time Out**, represents the maximum amount of time that a service will wait for an expected response before terminating a connection.

**Reserved Field.**

- Field 9, **Idle Time Out**, specifies the amount of time that a service will remain idle before polling their respective local message queue for messages waiting to be processed. For an Enterprise Server’s DSH, this specifies the amount of time the service remains idle before terminating. See [Chapter 17, “Enterprise Server,”](#) for more information.

Valid formats include:

- nMS = n milliseconds
- nS = n seconds
- nM = n minutes
- nH = n hours

Allowable value type: string consisting of numeric suffixed by alpha unit modifier

Representation: alphanumeric

Maximum permitted value: 1000MS, 60S, 60M, 24H

Minimum permitted value: 0MS, 0S, 0M, 0H indicating no time out

- Field 10, **Service Specific Information**, encompasses all remaining information contained in a Service Info Entry parameter after Field 9. The values specified in this field are passed to an instance of this service when it is activated. If a service does not require or accept values from this subparameter when it is activated then the string “NO\_DATA” must be specified. The format and valid values of this field are service-dependent and are discussed in the remainder of the SIE subparameter description.

For more information, see [Appendix D, “Sample SERVICE INFO ENTRY Definitions.”](#)

Allowable value type: numeric or string

Default: NONE

Use: Server

## SMF EXIT

Description: Optional parameter used to report wall clock time and the CPU time for an individual user session with a Query Processor Task. For more information, see [Chapter 15, “System Exits.”](#)

Example:

```
SMF EXIT = CACSX02 RECTYPE=255 ,SYSID=JES2
```

Default: none

Use: Server, Client Config, Member Override

## STATEMENT RETENTION

Description: **Optional Parameter** defines the prepared statement behavior when a commit or rollback operation occurs. The acceptable values for this parameter are:

- SYNCPOINT - Release the statement whenever a COMMIT or ROLLBACK is issued. This is consistent with DB2 Version 4 and earlier.
- ROLLBACK - Release the statement only when a ROLLBACK syncpoint is issued. This is equivalent with the DYNAMICKEEP option introduced in DB2 version 5.
- DISCONNECT - Release the statement only when the user disconnects from the data server. All prepared statements are retained across both COMMIT and ROLLBACK calls.

Example:

```
STATEMENT RETENTION = SYNCPOINT
```

Default: SYNCPOINT

Recommended value: SYNCPOINT is recommended unless there are specific application requirements for re-executing a prepared statement after a syncpoint is issued.

Use: Server, Query Processor

## STATIC CATALOGS

Description: **Optional parameter** used to activate static catalog processing for the system catalog data sets referenced by the Server. Static catalog close should be used when the system catalogs referenced by the Data Savant will not be updated while the Data Savant is executing (essentially, when the Server is operating in production mode and the system catalogs are static).

When static catalog processing is activated, the System Catalog files are opened once for a Query Processor task. The System Catalog files remain open until that Server is shut down. In normal operating mode, the System Catalogs are closed after the required table and column information has been retrieved in order to process a query, for each query processed by the Query Processor.

Activation of static catalog processing can substantially improve query performance in outer/inner cursor situations when a large number of queries are being issued serially.

Example:

```
STATIC CATALOGS = 1
```

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 1

Minimum permitted value: 0

Allowed value and results:

- 0 (close system catalog files and establish read locks for each query)
- 1 (close system catalog files when the Server is shut down)

Default: 0

Use: Server

## TASK PARAMETERS

Description: **Optional parameter** that specifies SAS/C runtime options passed to system daughter tasks through the OS/390 ATTACH macro.

One common use of this parameter is to pass TCP/IP information to the Communications Interface task.

The TCPIP\_PREFIX variable sets the high-level qualifier (hlq) for finding the TCP/IP system data sets. It can be set to use the installation defined data sets, or a user-defined data set.

The TCPIP\_MACH variable sets the address space name/subsystem name of the TCPIP stack for Interlink. For IBM's TCP/IP system utilizing the Berkeley Socket interface, this parameter can also be specified in the hlq.TCPIP.DATA file under the parameter TCPIPUSERID.

The default for both variables is TCPIP.

```
TASK PARAMETERS= =TCPIP_PREFIX=TCPIP =TCPIP_MACH=TCPIP
```

The Time Zone environment variable (TZ) must be set for each job on OS/390. The variable sets the time zone in which the task will start, for example Pacific Standard Time (PST).

Example:

```
TASK PARAMETERS = =MI =TZ=PST8PDT
```

This sets the time zone to PST plus 8 hours from Greenwich mean time (8) and Pacific Daylight Time (PDT).

Using the same example for Eastern Standard Time (EST), enter the following information:

```
TASK PARAMETERS = =MI =TZ=EST5EDT
```

For additional information about other valid TZ settings see the *SAS/C Compiler and Library User's Guide*.

Representation: string

Maximum permitted value: any valid parameter preceded by the equal (=) sign, as documented in the *SAS/C Compiler and Library User's Guide*.

Default: none

Use: Server, Client

---

## TRACE LEVEL

Description: **Optional parameter** that regulates the amount of information placed into trace log by Server tasks.

Example:

```
TRACE LEVEL = 4
```

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 20

Minimum permitted value: 0

Allowed values and results:

- 20 (no trace information generated)
- 16 (identify fatal error conditions)
- 8 (identify all recoverable error conditions)
- 4 (generate warning messages)
- 3 (enable message logging)
- 1 (generate tracing information)
- 0 (trace all)

Default: 0

Use: Client

**WARNING:** This parameter should only be changed at the request of CrossAccess Technical Support. Settings lower than 4 will cause response time degradation.

## USER CONFIG

Description: **Optional parameter** used to allow individual users to override the standard Server configuration parameter settings or a Query Processor task. Client configuration overrides are done by user ID. A member should exist in the VHSCONF data set for each user ID that will be accessing the Server. The USERID must be passed to eXadas from the client application.

If an individual user ID's member is not found in the VHSCONF data set or the USERID is not passed through the client application, no error is issued, and the Server level configuration parameters are used. If a member name is found under the user ID, the parameters become active once the user connects to the Server.

The following parameters can be overridden in a user level configuration file:

- BTREE BUFFERS,
- CPU GOVERNOR,
- LD TEMP SPACE,
- JOIN MAX TABLES ANALYZED,
- LOCALE,
- MAX ROWS EXAMINED,
- MAX ROWS EXCEEDED ACTION,
- MAX ROWS RETURNED,
- OME,
- PDQ,
- SMF EXIT
- VSAM AMPARMS, and
- WLM UOW.

Example:

```
USER CONFIG = 1
```

Allowable value type: numeric

Representation: decimal

Maximum permitted value: 1

Minimum permitted value: 0

Allowed values and results:

- 0 (do not activate user level configuration overrides)
- 1 (activate user level configuration overrides)

Default: 0

Use: Server, Query Processor

## USERID

Description: **Optional parameter** that is a default dynamic SQL ID if no ID is present on a CONNECT statement, or dynamic CONNECT is issued due to the client application not issuing a CONNECT statement. USERID is used when the first line in the SQL input file is blank.

Example:

```
USERID = CACUSER
```

Allowable value type: string

Representation: maximum of 7 characters with no spaces. If more than 7 characters are specified, only the first 7 are used.

Default: NONE

Use: Client

## USERPASSWORD

Description: **Optional parameter** that is the default dynamic SQL ID password if no ID is present on a CONNECT statement, or if a dynamic CONNECT is issued due to the client application not issuing a CONNECT statement.

USERPASSWORD is used when the first line in the SQL input file is blank.

Example:

```
USERPASSWORD = CACPWD
```

Allowable value type: string

Representation: maximum of 8 characters with no spaces

Default: NONE

Use: Client

## VSAM AMPARMS

Description: **Optional parameter** string used to supply VSAM buffer and memory-tuning parameters when a VSAM file is opened. The VSAM AMPARMS parameter specifies tuning parameters that are applied to all VSAM files opened when a single cursor is open.

The VSAM AMPARMS parameter takes the form of a string of comma delimited parameters that are passed to the SAS/C afopen call that is used to access VSAM files. The following parameters can be supplied:

- **BUFND=n**: Specifies the number of data I/O buffers VSAM is to use. Specification of this parameter is equivalent to coding the BUFND value on a DD statement. A data buffer is the size of a control interval in the data component of a VSAM cluster. The default number of data buffers is the

number of strings plus one. If you are using the VSAM service, the default number of buffers would be 11. If you are not using the VSAM service, the default number of buffers is two.

Generally, with sequential access the optimum value for the data buffers is six buffers or the size of the control area, whichever is less. When skip-sequential processing (random keyed read access) is being performed, specifying two buffers is optimum. Specifying a larger BUFND value when the VSAM file is being scanned during query processing generally yields performance improvements. In keyed access situations specifying a larger BUFND may show no performance improvements, or may actually degrade query performance by tying up large amounts of virtual storage and causing excessive paging.

- **BUFNI=n:** Specifies the number of index I/O buffers VSAM is to use. Specification of this parameter is equivalent to coding the BUFNI value on a DD statement. An index buffer is the size of a control interval in the index component of a keyed VSAM cluster. If you are using the VSAM service, the default number of index buffers is 10. If you are not using the VSAM service, the default number of index buffers is 1.

For keyed access, the optimum BUFNI specification is the number of high-level (non-sequence set) index buffers + 1. This number can be determined by subtracting the number of data control areas from the total number of index control intervals within the data set. An upper bound BUFNI specification of 32 can be used, which accommodates most VSAM files with reasonable index control interval and data control area sizes (cylinder allocated data component) up to the 4GB maximum allowed data component size. Specification of a large BUFNI value incurs little or no performance penalty, unless they are excessive.

- **BUFSP=n:** Specifies the maximum number of bytes of storage to be used by VSAM for file data and index I/O buffers. Specification of this parameter is equivalent to coding the BUFSP value on a DD statement. A data or index buffer is the size of a control interval in the data or index component.

A valid BUFSP specification generally overrides any BUFND or BUFNI specification. However, the VSAM rules for specifying an optimum BUFSP value are fairly complex. The appropriate IBM-supplied information on the ACB macro should be consulted to determine the rules for specifying a BUFSP value.

Example:

```
VSAM AMPARMS = BUFND=20, BUFNI=15
```

Allowable value type: string

Representation: string

Maximum permitted value: 255 characters

Minimum permitted value: 7 characters



Default: None

Use: Server, Query Processor, Client Config, Member Override

## WLM UOW

Description: **Optional parameter** that specifies the Workload Manager unit-of-work activities and manages queries in WLM goal mode. The WLM Exit is activated using a SERVICE INFO ENTRY parameter to initialize the exit once when the address space is initialized. The WLM Exit must be able to handle all of the users accessing the Server concurrently.

Example:

```
SERVICE INFO ENTRY = CACWLM WLM 1 1 1 20 0M 0M 500000/
                    CACSX06 SUBSYS=xxx SUBSYSID=xxxx
```

For more information on the WLM parameter settings, see [Chapter 15, “System Exits.”](#)

You can also supply WLM unit-of-work information from the master configuration member, however, CrossAccess recommends that you supply unit-of-work information from the service configuration member(s). Supplying this information at the service level allows different services (data sources) to run queries with different performance profiles. For example, if you are running short queries you can give them more resources. For longer running queries you can use period switching to reduce the rate that these types of queries uses resources.

An example of the WLM UOW in the Master Configuration file follows.

Example:

```
WLM UOW = TRXNAME=XXXXXXX
```

- Field 1: TRXNAME=XXXXXXXX
- Field 2: TRXCLASS=XXXXXXXX

These parameters can be one to eight characters long. Both of these parameters are optional. If supplied they are passed to the WLM to classify the work into a service or report class otherwise it takes the default class for this subsystem type. It is recommended that you define a subsystem type and classification rules for the workload processed by the eXadas Query Processor. If an existing subsystem type is used then select which of these parameters fits that subtype. For example the STC type supplied with OS/390 supports user ID, and TRANNAME. JES supports user ID, TRANNAME, and TRANCLASS.

See IBM's *OS/390 Planning: Workload Management* for information on how to define service classes and classification rules. This assumes you are going to operate the system in WLM goal mode. The priority for units of work should be

less than VTAM and IMS. The discretionary goal may result in very slow response times. Performance periods allow you to define a high number of service units for short transactions and a smaller number for long running ones.

Allowable value type: string

Representation: string

Default: none

Use: Server, Query Processor, Client Config. Member Override

# Appendix B

## Sample Stored Procedure VTAM and CICS Definitions

### Introduction to Sample Stored Procedure VTAM and CICS Definitions

There are two sets of definitions required to activate a CICS-enabled stored procedure implementation. There are VTAM definitions required to support the communications between the Server and the CICS system. There are CICS definitions required to identify files, programs, and transactions. Additional CICS definitions are required to support the communications between the Server and the CICS system.

**NOTE:** The program fixes HVT4301 UW25918 and UW29177 are required for VTAM 4.4 when implementing the CICS stored procedures. Contact IBM to receive these fixes.

# VTAM Resource Definitions

An APPL definition is required on the server side to provide a local LU name for the communication session. Multiple local LU names (APPL definitions) may be required depending upon the number of server-stored procedure users that you allow to be active concurrently. Allowing one local LU name per active server stored procedure user eliminates any possibility of communication failures because the local LU name is busy. However, a one-to-one relationship between local LU names and active Server's stored procedure users is not usually required. As a general rule, a relatively small number of local LU names should be adequate for most sites. The number of local LU names required is the number of expected concurrent requests the server will handle. The maximum tasks specification (field 5) of the SERVICE INFO ENTRY for the Query Processor should be set to the number of local LU names that are defined.

The user site assigns the ACB name in the APPL definition. The ACB name is specified on the OPEN request issued by the user-written stored procedure program that executes in the Server address space. In the eXadas sample communication programs CACSPCOM or CACSPVTM, a pool of APPL definitions must be created by assigning sequentially ascending ACB names like CACAPPC0, CACAPPC1, CACAPPC2, and so on. To use this pool of ACB names, the OPEN request would then specify the local LU name as CACAPPC\*. The communications processor attempts to open the specified ACB name after replacing the asterisk position with a sequentially ascending value beginning at zero.

If a pool of 10 ACB names is found to be insufficient, the APPL definitions can use names that end in two digits, for example, CACPPC00, CACPPC01, CACPPC02, and so on. To use this pool of ACB names, the OPEN request would then specify the local LU name as CACPPC\*\*. Up to seven suffix characters (\*) can be specified.

The size of your APPL definition pool should be carefully controlled. This is important because if an OPEN request fails, the next ACB name is generated and the OPEN is re-attempted. If each subsequent OPEN request fails, the entire pool of ACB names will be attempted before the communications processor reports an OPEN failure. To use only a specific ACB name the OPEN request specifies the exact name without an asterisk suffix.

An example of VTAM APPL definitions and a VTAM Mode Table Entry definition follows:

## VTAM APPL Definition

```

CACCAPPL VBUILD TYPE=APPL
CACCICS1 APPL ACBNAME=CACCICS1, X
                APPC=YES, X
                AUTOSES=1, X
                MODETAB=CACCMODE, X
                DLOGMOD=MTLU62, X
                AUTH=(ACQ), X
                EAS=100, PARSESS=YES, X

```

```

                                SONSCIP=YES,           X
                                DMINWNL=0,             X
                                DMINWNR=1,           X
                                DSESLIM=100
CACCIICS2 APPL ACBNAME=CACCIICS2,           X
                                APPC=YES,           X
                                AUTOSES=1,         X
                                MODETAB=CACCMODE,  X
                                DLOGMOD=MTLU62,    X
                                AUTH=(ACQ),        X
                                EAS=1, PARSESS=YES, X
                                SONSCIP=YES,       X
                                DMINWNL=0,         X
                                DMINWNR=1,         X
                                DSESLIM=1

```

The example shows APPL definitions to be used by the example program CACSPCOM or CACSPVTM. Your actual APPL definitions may vary based upon site standards. The OPEN request issued by CACSPCOM specifies local LU name CACPPC0\*, thereby using the APPL definitions as a pool of up to 10 entries. Assuming the definitions in the example are complete, the pool actually consists of only two entries. CACCMODE is defined as the Logon Mode Table name and MTLU62 is defined as the Logon Mode Table entry name. The Logon Mode Table entry must be in either the specified Logon Mode Table or in ISTINCLM, an IBM-supplied Logon Mode Table. The OPEN request issued by CACSPCOM also specifies the Logon Mode Table entry (DLOGMOD) name. If you plan to execute the eXadas example program CACSPCOM and changes were made to ACB name or DLOGMOD, you must correct program CACSPCOM to specify the modified values, recompile, and link-edit the program before attempting to execute the example. You can make changes to MODETAB to identify your correct Mode Table Name without affecting the example program CACSPCOM.

#### VTAM Mode Table Entry Definition

```

CACCMODE  MODEENT LOGMODE=CACCMODE,        *
                                TYPE=0,        *
                                FMPROF=X'13',  *
                                TSPROF=X'07',  *
                                PRIPROT=X'B0', *
                                SECPROT=X'B0', *
                                COMPROT=X'D0B1', *
                                RUSIZES=X'8585', *
                                PSERVIC=X'0602000000000000000000000300'

```

The example shows a Mode Table entry definition used by the example program CACSPCOM or CACSPVTM. Your actual Mode Table entry definition may vary based upon site standards. As stated earlier, the OPEN request issued by CACSPCOM specifies Logon Mode Table entry MTLU62. If you plan to execute the eXadas example program CACSPCOM and changes were made to LOGMODE, you must correct program CACSPCOM to specify the new name, recompile, and link-edit the program before attempting to execute the example. The CICS system must have access to an identically named Logon Mode Table entry.

# CICS Resource Definitions

The CICS APPLID becomes the Remote LU Name for the communication session. The CICS APPLID can handle multiple sessions from Server stored procedure processors. Before completing specific resource definitions in CICS, several definitions within your CICS or VTAM system need to be reviewed. Incorrect specification of the following parameters will result in your system failing to operate correctly.

- The CICS system initialization table (DFHSIT) definition or initialization overrides must include ISC=YES to enable intercommunication programs.
- The ACF/VTAM application definition for your CICS system must include the following options on the VTAM APPL statement:
  - AUTH=(ACQ,VPACE,.....) to allow CICS to acquire LUTYPE6 sessions and to allow pacing of intersystem flows.
  - VPACING=n specifies the pacing rate.
  - EAS=n specifies the maximum number of network addressable units with which CICS can establish sessions.
  - PARSESS=YES to enable LUTYPE6 parallel session support.
  - SONSCIP=YES to enable session outage notification support.
- APPC=YES must not be coded on the APPL statement.

For a complete discussion of these topics and others related to intersystem connectivity, see the *IBM CICS/ESA Intercommunications Guide* (IBM Document Number SC33-1181-01). Related information is listed in the index under the general topics as follows:

- Installation:
  - ACF/VTAM definition for CICS,
  - LOGMODE entries, and
  - Intersystem communication.
- Intersystem communication (ISC):
  - connections between systems,
  - defining APPC links,
  - defining APPC mode sets, and
  - installation considerations.

The resource definitions to install the required LU6.2 Connection Handler for the eXadas Stored Procedure implementation include:

- The program definition. This program has been identified earlier in this document as load member CACSP62. The name of this program is used only in the transaction definition. This load module must be in a library included in the DFHRPL concatenation for your CICS system.
- The transaction definition. The user site assigns the transaction name. This transaction is referenced as XASP. The OPEN request issued by the sample program CACSPCOM specifies transaction name XASP. If you plan to execute the supplied sample program CACSPCOM and you have defined a different transaction name, you must correct program CACSPCOM to specify the new transaction, recompile, and link-edit the program before attempting to execute the example.

The following entries show the online CEDA panels used to define the required CICS resources. The fields that must be described by the user for the define to be successful are shown in underlined bold type, for example, XXXX. Fields that contain only lower-case characters must be defined by the user site. Fields that are mixed case must be entered exactly as shown. Any other modifiable fields on these panels may be specified to conform to site standards. The CEDA panels were generated from an OS/390 CICS V4.1 system.

#### PROGRAM DEFINITION

```

OVERTYPE TO MODIFY                                CICS RELEASE
= 0410
CEDA DEFINE PROGram( pppppppp )
  PROGram      : CACSP62
  Group        : gggg
  DDescription ==>
  Language     ==> Assembler CObol | Assembler | Le370 | C
|Pli
                                     | Rpg
  REload       ==> No                    No | Yes
  RESident     ==> No                    No | Yes
  USAge        ==> Normal                 Normal | Transient
  USElpacopy   ==> No                    No | Yes
  Status       ==> Enabled                Enabled | Disabled
  RSl          : 00                      0-24 | Public
  Cedf         ==> Yes                    Yes | No
  Datalocation ==> Below                  Below | Any
  EXECKey      ==> User                   User | Cics
REMOTE ATTRIBUTES
  REMOTESystem ==>
  REMOTENAME   ==>
  Transid      ==>
  EXECUTIONset ==> Fullapi                Fullapi | Dplsubset

```

#### TRANSACTION DEFINITION

```

OVERTYPE TO MODIFY                                CICS RELEASE
= 0410
CEDA DEFINE TRANSAction( tttt )
  TRANSAction  : XASP
  Group        : gggg
  DDescription ==>

```

```

PROGram      ==> CACSP62
TWAsize     ==> 00000          0-32767
PROFile     ==> DFHCICST
PARTitionset ==>
STAtus      ==> Enabled       Enabled | Disabled
PRIMedsize  : 00000          0-65520
TASKDATAloc ==> Below        Below | Any
TASKDATAKey ==> User         User | Cics
STOrageclear ==> No          No | Yes
RUnaway     ==> System       System | 0-2700000
SHutdown    ==> Disabled     Disabled | Enabled
ISolate     ==> Yes          Yes | No
REMOTE ATTRIBUTES
DYNAMIC     ==> No           No | Yes
REMOTESystem ==>
REMOTEName  ==>
TRProf      ==>
Localq      ==>             No | Yes
SCHEDULING
PRIOrity    ==> 001          0-255
TClass     : No             No | 1-10
TRANClass   ==>
ALIASES
Alias       ==>
TASKReq     ==>
XTRanid     ==>
TPName      ==>
            ==>
            ==>
            ==>
XTPname     ==>
            ==>
            ==>
RECOVERY
DTImout     ==> No           No | 1-6800
INDoubt     ==> Backout     Backout | Commit | Wait
REStart     ==> No           No | Yes
SPurge      ==> No           No | Yes
TPUrge      ==> No           No | Yes
DUmp        ==> Yes         Yes | No
TRAcE       ==> Yes         Yes | No
CONfdata    ==> No           No | Yes
SECURITY
RESec       ==> No           No | Yes
CMdsec      ==> No           No | Yes
Extsec      : No
TRANSec     : 01             1-64
RS1         : 00             0-24 | Public

```

The resource definitions to install the optional eXadas Stored Procedure example program environments include:

- the program definition for the example program CACSPREM,
- the file definition for the sample VSAM Employee file CACIVP,
- one CONNECTION/SESSION definition is required for each LU Name that may be used to communicate with CICS from a Server. For purposes of this example, two LU NAMES are defined to VTAM (CACPPC00 and CACPPC01). The sample program CACSPCOM, executing in the Server address space, uses one of these Local LU Names when it requests an **OPEN** for the LU6.2 conversation.



The following entries show the on-line CEDA panels that are used to define the optional CICS resources. The fields that must be described by the user for the define to be successful are shown in underlined bold type, for example, **xxxx**. Fields that contain only lower case characters must be defined by the user site. Fields that are mixed case (upper- and lower-case) must be entered exactly as shown. Fields that are completely upper case characters and/or numeric values may be changed by the user site. If changes are made, you must compensate for those changes in the related program(s), or VTAM APPL definition(s), or Mode Table entry definition(s). Any other modifiable fields on these panels may be specified to conform to site standards. The CEDA panels shown in the following example were generated from an OS/390 CICS V4.1 system.

## PROGRAM Definition

```

OVERTYPE TO MODIFY                                CICS RELEASE
= 0410
CEDA DEFine PROGram( CACSPREM )
  PROGram      : CACSPREM
  Group        : gggg
  Description   ==>
  Language      ==> Le370  CObol | Assembler | Le370 | C | Pli
                  | Rpg
  REload        ==> No      No | Yes
  RESident      ==> No      No | Yes
  USAGE         ==> Normal  Normal | Transient
  USElpacopy    ==> No      No | Yes
  Status        ==> Enabled  Enabled | Disabled
  RSl           : 00       0-24 | Public
  Cedf          ==> Yes     Yes | No
  Datalocation  ==> Below   Below | Any
  EXECKey       ==> User    User | Cics
REMOTE ATTRIBUTES
  REMOTESystem ==>
  REMOTENAME    ==>
  Transid       ==>
  EXECUTIONset ==> Fullapi   Fullapi | Dplsubset

```

The CACSPREM example program is a COBOL II program. The program, as supplied, does not exploit Language Environment/370 facilities. However, if the program is compiled using an SAA AD/Cycle COBOL/370 Version 1 Release 1, or later, the attribute LANGUAGE ==>Le370 must remain as shown in the previous example. If the program is not compiled by a Language Environment/370 enabled compiler, change the attribute to be LANGUAGE==>CObol.

If the LANGUAGE attribute specifies Le370, you must have Language Environment/370 support installed in your CICS system. If the LANGUAGE attribute specifies CObol, you must have the VS COBOL II interface installed in your CICS system. If CICS is not presently enabled to support the LANGUAGE attribute specified for this program, see the topic “Adding CICS support for programming languages” in the *CICS System Definition Guide* (IBM Document Number SC33-1164-00). There are sub-topics explaining in detail how to install and utilize either the Language Environment/370 or the VS COBOL II support.

Specifically, review the following topics:

- Language Environment/370 support,
- Installing Language Environment/370 support,
- Language Environment/370 support for COBOL, or
- Installing CICS support for VS COBOL II.

## FILE DEFINITION

```

OVERTYPE TO MODIFY                                CICS RELEASE
= 0410
CEDA DEFine File( CACIVP )
  File      : CACIVP
  Group     : gggg
  DEscription ==>
VSAM PARAMETERS
  DSName    ==> your.vsam.cluster.name.here
  Password  ==>                                PASSWORD NOT SPECIFIED
  Lsrpoolid ==> 1                               1-8 | None
  DSNSharing ==> Allreqs                        Allreqs | Modifyreqs
  STRings   ==> 001                             1-255
  Nsrgroup  ==>
REMOTE ATTRIBUTES
  REMOTESystem ==>
  REMOTENAME ==>
  RECORDSize  ==>                               1-32767
  Keylength   ==>                               1-255
INITIAL STATUS
  STATUS      ==> Enabled                       Enabled | Disabled |
Unenabled
  Opertime    ==> Firstref                      Firstref | Startup
  DISposition ==> Share                        Share | Old
BUFFERS
  Databuffers ==> 00002                         2-32767
  Indexbuffers ==> 00001                       1-32767
DATATABLE PARAMETERS
  Table       ==> No                            No | Cics | User
  Maxnumrecs  ==>                               16-16777215
DATA FORMAT
  RECORDFormat ==> F                          V | F
OPERATIONS
  Add         ==> Yes                          No | Yes
  BRowse      ==> No                            No | Yes
  DElete      ==> Yes                          No | Yes
  REAd        ==> Yes                          Yes | No
  UpDate      ==> Yes                          No | Yes
AUTO JOURNALLING
  JOurnal     ==> No                            No | 1-99
  JNLRead     ==> None                          None | Updateonly |
Readonly | All
  JNLSYNCRead ==> No                            No | Yes
  JNLUpdate   ==> No                            No | Yes
  JNLAdd      ==> None                          None | Before | AFTER | ALL
  JNLSYNCRwrite ==> Yes                        Yes | No
RECOVERY PARAMETERS
  RECOvery    ==> None                          None | Backoutonly | All
  Fwdrecovlog ==> No                            No | 1-99
  Backuptype  ==> Static                        Static | Dynamic
SECURITY
  RESsecnum   : 00                              0-24 | Public

```

## CONNECTION DEFINITION (1 OF 2)

```

OVERTYPE TO MODIFY
= 0410
CEDA DEFINE Connection( ccc1 )
  Connection      : ccc1
  Group           : gggg
  Description     ==>
CONNECTION IDENTIFIERS
  Netname        ==> CACPPC00
  INDSys         ==>
REMOTE ATTRIBUTES
  REMOTESYSTEM ==>
  REMOTENAME   ==>
  REMOTESYSNET ==>
CONNECTION PROPERTIES
  ACCESSMETHOD ==> Vtam          Vtam | IRC | INdirect | Xm
  PROTOCOL     ==> Appc          Appc | Lu61 | Exci
  CONNTYPE     ==>              Generic | Specific
  SINGLENESS  ==> No            No | Yes
  DATASTREAM ==> User          User | 3270 | SCs |
STRfield | Lms
  RECORDFORMAT ==> U            U | Vb
  QUEUELIMIT   ==> No            No | 0-9999
  MAXQTIME     ==> No            No | 0-9999
OPERATIONAL PROPERTIES
  AUTOCONNECT ==> No            No | Yes | All
  INSERVICE  ==> Yes          Yes | No
SECURITY
  SECURITYNAME ==>
  ATTACHSEC   ==> Local        Local | Identify | Verify |
Persistent
  | Mixidpe
  BINDPASSWORD :                PASSWORD NOT SPECIFIED
  BINDSECURITY ==> No            No | Yes
  USEDFLTUSER  ==> No            No | Yes
RECOVERY
  PSRECOVERY   ==> Sysdefault   Sysdefault | None

```

## CONNECTION DEFINITION (2 OF 2)

```

OVERTYPE TO MODIFY
= 0410
CEDA DEFINE Connection( ccc2 )
  Connection      : ccc2
  Group           : gggg
  Description     ==>
CONNECTION IDENTIFIERS
  Netname        ==> CACPPC01
  INDSys         ==>
REMOTE ATTRIBUTES
  REMOTESYSTEM ==>
  REMOTENAME   ==>
  REMOTESYSNET ==>
CONNECTION PROPERTIES
  ACCESSMETHOD ==> Vtam          Vtam | IRC | INdirect
  | Xm
  PROTOCOL     ==> Appc          Appc | Lu61 | Exci
  CONNTYPE     ==>              Generic | Specific
  SINGLENESS  ==> No            No | Yes
  DATASTREAM ==> User          User | 3270 | SCs |
STRfield

```

RECORDformat ==> U	Lms
QueueLimit ==> No	U   Vb
Maxqtime ==> No	No   0-9999
OPERATIONAL PROPERTIES	No   0-9999
Autoconnect ==> No	No   Yes   All
INService ==> Yes	Yes   No
SECURITY	
Securityname ==>	
Attachsec ==> Local	Local   Identify
Verify	Persistent   Mixidpe
BINDPassword :	PASSWORD NOT SPECIFIED
BINDSecurity ==> No	No   Yes
Usedfltuser ==> No	No   Yes
RECOVERY	
PSrecovery ==> Sysdefault	Sysdefault   None

**NOTE:** If you want CICS to verify user IDs and passwords as valid CICS users, set the ATTACHSEC parameter to Verify.

SESSIONS DEFINITION (1 OF 2)

```

OVERTYPE TO MODIFY                                CICS RELEASE
= 0410
CEDA DEFINE Sessions( sssssss1 )
  Sessions      : ssssss1
  Group         : gggg
  Description   ==>
SESSION IDENTIFIERS
  Connection    ==> ccc1
  SESSName     ==>
  NETNameq     ==>
  MODename     ==> MTLU62
SESSION PROPERTIES
  Protocol     ==> Appc           Appc | Lu61 | Exci
  Maximum     ==> 001 , 000     0-999
  RECEIVEPfx  ==>
  RECEIVECount ==>              1-999
  SENDPfx     ==>
  SENDCount   ==>              1-999
  SENDSize    ==> 00256         1-30720
  RECEIVESize ==> 00256         1-30720
  SESSPriority ==> 000           0-255
  Transaction  :
OPERATOR DEFAULTS
  OPERId      :
  OPERPriority : 000           0-255
  OPERRsl    : 0
24,....
  OPERSecurity : 1
1-64,....
PRESET SECURITY
  USERId     ==>
OPERATIONAL PROPERTIES
  Autoconnect ==> No           No | Yes | All
  INService  :
  Buildchain ==> Yes          Yes | No
  USERArealen ==> 000         0-255
  IOarealen  ==> 00000 , 00000 0-32767
  RELreq     ==> No           No | Yes
  Discreq    ==> No           No | Yes
  
```

```

NEPclass      ==> 000                0-255
RECOVERY
RECOVOption   ==> Sysdefault          Sysdefault | Clearconv
                                           | Releasesess | Uncondrel
| None
RECOVNotify   : None                None | Message |
Transaction

```

**SESSIONS DEFINITION ( 2 OF 2 )**

```

OVERTYPE TO MODIFY                                CICS RELEASE
= 0410
CEDA DEFINE Sessions( ssssss2 )
  Sessions      : sssss2
  Group         : gggg
  Description    ==>
SESSION IDENTIFIERS
  Connection     ==> ccc2
  ESSName        ==>
  NETnameq      ==>
  MODename       ==> MTLU62
SESSION PROPERTIES
  Protocol       ==> Appc              Appc | Lu61 | Exci
  Maximum        ==> 001 , 000        0-999
  RECEIVEPfx     ==>
  RECEIVECount   ==>                  1-999
  SENDPfx        ==>
  SENDCount      ==>                  1-999
  SENDSize       ==> 00256           1-30720
  RECEIVESize    ==> 00256           1-30720
  SESSPriority    ==> 000              0-255
  Transaction    :
OPERATOR DEFAULTS
  OPERID        :
  OPERPriority   : 000                0-255
  OPERRsl       : 0
24,...
  OPERSecurity   : 1                  1-
64,...
PRESET SECURITY
  USERID        ==>
OPERATIONAL PROPERTIES
  Autoconnect    ==> No                No | Yes | All
  INservice      :
  Buildchain     ==> Yes                Yes | No
  USERArealen   ==> 000                0-255
  IOarealen      ==> 00000 , 00000     0-32767
  RELreq         ==> No                No | Yes
  DIScreq       ==> No                No | Yes
  NEPclass       ==> 000                0-255
RECOVERY
  RECOVOption    ==> Sysdefault          Sysdefault | Clearconv
                                           | Releasesess | Uncondrel
| None
RECOVNotify     : None                None | Message |
Transaction

```

Once the CEDA definitions are completed and verified correct, the group should be installed using the on-line command CEDA INSTALL GROUP(gggg)ALL. Installing the group makes the definitions immediately available for use. The

group should also be added to the start-up group list so the stored procedure group will be automatically installed during each subsequent start up of the CICS system. This is accomplished by locating the GRPLIST parameter on the system initialization table (DFHSIT) or in the SIT overrides used to start the CICS system. When the GRPLIST=xxxxxxx parameter has been determined, that name is used in the on-line command `CEDA ADD GROUP(gggg) LIST(xxxxxxx)` to permanently add the eXadas Stored Procedure processing group to the CICS system start-up.

# Appendix C

## MTO Command Reference

### Introduction to MTO Commands

The server is started and stopped with standard OS/390 console commands. This chapter describes the available operator commands. The operator commands allow you to dynamically configure OS/390-based servers. This chapter describes the MTO facility and how to monitor and control a server using the operator commands. The following topics are covered:

- [“MTO Facility,”](#) on page 424, and
- [“Commands,”](#) on page 424.

For additional information on server operations, see [Chapter 10, “Server Operations.”](#)

# MTO Facility

Dynamic configuration can be accomplished through a traditional MTO interface, or through a programmable operator API. This interface requires an operator task to be running. The operator interface will be the standard modify interface.

Commands are executed in the following format:

```
F program name,command
```

Where:

- `F` is the abbreviation for the OS/390 MODIFY command.
- `program name` is the name of the eXadas started-task to communicate with.
- `command` is the command passed to the server for execution.

The commands are described in detail in the section that follows.

**NOTE:** If the started task (program name) is a server that was started by an eXadas Enterprise Server, specify the fully-qualified task name in the form `server.stepname`. To send the modify to all servers, you may use an asterisk (\*).

For example:

```
/f cacds.t9396840,display,users  
/f cacds.*,display,config=master
```

## Commands

This section details all the available operator commands for the MTO facility.

### **SET,NAME=name, ORD=number, VALUE=value**

This command modifies a configuration parameter in an active configuration. In addition, this command can be used to create new **SERVICE INFO ENTRY** configuration parameters for the active server. Any changes made to an active configuration remain only for the duration of the active configuration unless the **FLUSH** command is issued to commit the change permanently.

Configuration values containing imbedded spaces and special characters must be enclosed in either quotation marks (“ ”) or apostrophes (‘ ’).



Timing values must be suffixed with either M (minutes), S (seconds), or MS (milliseconds), for example, 5M, 5S, or 500MS.

To reset a configuration parameter back to its system default value, issue the set command with VALUE=NULL. The associated parameter will be reset. If the **FLUSH** command is issued after resetting a configuration value, the reset parameter will not be written out to the associated initialization file. Resetting a SERVICE INFO ENTRY removes that entry from the active configuration.

**WARNING:** Do not reset SERVICE INFO ENTRY values that have active instances. Issue the STOP,SERVICE command to stop all active instances first.

To create a new SERVICE INFO ENTRY, issue the SET command with an ordinal value greater than the highest existing SERVICE INFO ENTRY in the configuration.

Examples:

```
F CACDS , SET , NAME=MASTER , ORD=105 , VALUE=10m
RESPONSE TIME OUT=10M
```

```
F CACDS , SET , NAME=MASTER , ORD=106 , VALUE='CACINIT P390 2 1 1
256 4 1M 30S P390'
SERVICE INFO ENTRY = CACINIT P390 2 1 1 256 4 1M 30S P390
```

## CANCEL,USER=userid

This command puts all users referenced by userid into a cancelled state. The users will be notified upon completion of any subsequent action. For maximum effectiveness, INTERLEAVE INTERVAL should be enabled. See [page 389](#) for more information on INTERLEAVE INTERVAL.

Example:

```
CANCEL , USER=PUBLIC
```

## CANCEL,SESSIONID=sessionid

This command puts a single user, referenced by sessionid, into a cancelled state. The user will be notified upon completion of any subsequent action. For maximum effectiveness, INTERLEAVE INTERVAL should be enabled. See [page 389](#) for more information on INTERLEAVE INTERVAL.

Example:

```
CANCEL , SESSIONID=212683400
```

**NOTE:** When displaying users (`DISPLAY, USERS`) after a cancel command has been issued, you can identify dormant users by their sessionid, which is set to -1 on cancel. These users will be removed, and will no longer display, once any new activity takes place for them. Active users will be removed immediately.

## DISPLAY,QUERIES

This command lists all of the query statements on the server by statement name and session id. With this information, you can issue the cancel for a particular query.

Example:

```
DISPLAY,QUERIES
```

Sample output:

```
CAC00200I DISPLAY,QUERIES
```

QUERY	USER	SESSIONID	SERVICE	TASKID	TYPE	STATE	MEMORY
SELECT_ST	wca027	351185720	CACSAMP	7139560	XQRY	CLOSED	29K/29K
S1	wca009	351514104	CACSAMP	7139560	XQRY	FETCHED	20K/22K
S1	WCA027	351184696	CACSAMP	7139560	unk	INITIAL	3K/3K

## CANCEL,QUERY=name,SESSIONID=sessionid

This command puts the query referenced by name and sessionid into a cancelled state. The user will be notified upon completion of any subsequent action. For maximum effectiveness, `INTERLEAVE INTERVAL` should be enabled. See [page 389](#) for more information on `INTERLEAVE INTERVAL`.

Example:

```
CANCEL,QUERY=S1,SESSIONID=212683400
```

## MODIFY,servicename,TRACELEVEL=number

This command changes the trace level for a specific service. This command is applicable to all services.

Example:

```
MODIFY ,SFCOMMON ,TRACELEVEL=1
```

## **MODIFY,servicename,TRIGGER START=number**

This command sets the message number trigger on which the logger will start dumping all received LOG messages to level 1, temporarily replacing the current trigger level. This command is applicable to type 1 services only (loggers).

Example:

```
MODIFY,LOG,TRIGGER START=0X00570012
```

## **MODIFY,servicename,TRIGGER STOP=number**

This command sets the message number trigger on which the logger will stop dumping (reverting to previous trigger level) all received LOG messages. This command is applicable to type 1 services only (loggers).

Example:

```
MODIFY,LOG,TRIGGER STOP=0x00570043
```

## **MODIFY,servicename,OUTPUT=DISPLAY**

This command causes the logger to mirror all output to SYSOUT (mirroring meaning output still goes to the logfile as well). This output is formatted to text, but does not include engcat explanations. This command is applicable to type 1 services only (loggers).

Example:

```
MODIFY,LOG,OUTPUT=DISPLAY
```

## **MODIFY,servicename,OUTPUT=DEFAULT**

This command turns off mirroring. This command is applicable to type 1 services only (loggers).

Example:

```
MODIFY, LOG, OUTPUT=DEFAULT
```

## **MODIFY,servicename,FLUSH**

This command causes the logger to flush all pending output to the log. This command is applicable to type 1 services only (loggers).

Example:

```
MODIFY, LOG, FLUSH
```

## **FLUSH,NAME=name**

This command writes an in-core configuration image, specified by name to its corresponding member in the VHSCONF referenced data set.

```
F CACDS, FLUSH, NAME=MASTER
```

**WARNING:** All comments in the initial configuration will be lost when the configuration is written back out. CrossAccess recommends backing up your initial configuration before issuing the FLUSH command.

## **DISPLAY, {SERVICES | USERS | CONFIG=name | CONFIGS | MEMORY | ALL }**

The display command outputs a formatted list of the selected server information. The format is described as follows:

- **SERVICES:** Displays all running services in the server.
- **USERS:** Displays all users connected to the server.
- **CONFIG=name:** Displays the contents of a specified configuration.
- **CONFIGS:** Lists all configurations currently active in the server.

- **MEMORY:** Displays the current memory pool utilization in the server.
- **ALL:** Lists **SERVICES**, **USERS**, **CONFIGS**, and **MEMORY**.

Sample command:

```
F CACDS , DISPLAY , ALL
```

Sample output:

```
16.14.17 JOB00441 CAC00200I DISPLAY , ALL
16.14.17 JOB00441
16.14.17 JOB00441 SERVICE TYPE TASKID TASKNAME STATUS USER
16.14.18 JOB00441 LOGGER CACLOG 9288536 CACLOG READY
16.14.18 JOB00441 CACSAMP CACQP 9287976 CACQP READY
16.14.18 JOB00441 TCPIP CACINIT 9287032 CACINIT READY
16.14.18 JOB00441 CACSAMP CACQP 9190800 CACQP READY
16.14.18 JOB00441 CACSAMP CACQP 9189888 CACQP READY
16.14.18 JOB00441
16.14.18 JOB00441 Total Number of TASKS = 5
16.14.18 JOB00441
16.14.18 JOB00441 USER SESSIONID HOSTNAME PROCESSID THREADID SERVICE
TASKID
16.14.18 JOB00441 CACUSER 103185312 unknown 185 224 CACSAMP
9287976
16.14.18 JOB00441 CACUSER 102817600 unknown 185 224 CACSAMP
9190800
16.14.18 JOB00441 102663264 unknown 97 136 CACSAMP
9189888
16.14.18 JOB00441
16.14.18 JOB00441 Total Number of USERS = 3
16.14.18 JOB00441
16.14.18 JOB00441 ACTIVE CONFIGURATIONS
16.14.18 JOB00441 MASTER
16.14.18 JOB00441 CACQPCF
16.14.18 JOB00441
16.14.18 JOB00441 Total Number of CONFIGURATIONS = 2
16.14.18 JOB00441 CAC00225I TOTAL MEMORY 16384K, USED 403K (2%), MAX USED 603K
(3%)
```

The following service states are displayed for either a **DISPLAY,ALL** or **DISPLAY,SERVICES** operator command:

- 01) **READY** waiting for requests
- 02) **RECEIVING** receiving a request
- 03) **RESPONDING** sending a response
- 04) **DYN EXEC** processing an SQL DYN EXEC request
- 05) **CLOSE CURSOR** processing an SQL CLOSE CURSOR request
- 06) **DESCRIBE** processing an SQL DESCRIBE request
- 07) **EXECUTE** processing an SQL EXECUTE request
- 08) **EXEC IMMED** processing an SQL DYN EXEC IMMEDIATE request
- 09) **FETCH** processing an SQL FETCH request
- 10) **OPEN CURSOR** processing an SQL OPEN request

- 11) **PREPARE** processing an SQL PREPARE request
- 12) **SLCT INTO** processing an SQL SELECT INTO request
- 13) **QUIESCE** unused
- 14) **STOP** processing a server STOP,ALL request

**NOTE:** The Query Processor can display all of the states. All services can display numbers 1,2,3 and 14.

The Connection Handler and the DSH can also display states 4 through 12 when ferrying the indicated request over to a Query Processor.

**NOTE:** The MTO DISPLAY SERVICES and MTO DISPLAY USERS commands will only display the first 8 characters of Field 10 of the SERVICE INFO ENTRY parameter (service name), even though the service name may be longer than 8 characters.

## **START,SERVICE=name**

This command starts an instance of a service based on the service definition in the master configuration file. To view existing service definitions, issue the **DISPLAY,CONFIG=MASTER** operator command.

Sample command:

```
F CACDS , START , SERVICE=CACSAMP
```

**NOTE:** Service definitions in the master configuration can be dynamically created and changed using the SET command.

## **STOP, {TASKID=tasknumber | SERVICE=name | ALL}**

This command tells the server controller to stop the specified task or service. **STOP,ALL** stops all running services and terminates the server itself. If the service or task does not exist, the command fails.

Sample command:

```
F CACDS , STOP , TASKID=9270608  
F CACDS , STOP , SERVICE=SFCOMMON
```

**WARNING:** The STOP command cancels any user activity in an active service and disconnects all active users from the stopped service.

# Appendix D

## Sample SERVICE INFO ENTRY Definitions

### Introduction

This appendix shows you some sample SERVICE INFO ENTRY definitions, on which you can base your own definitions:

- [“Region Controller and Logger,” on page 432,](#)
- [“Query Processor,” on page 432,](#)
- [“Connection Handler,” on page 432,](#)
- [“IMS Interface Initialization Services,” on page 433,](#)
- [“DB/2 Access,” on page 434,](#)
- [“Datacom Initialization Service,” on page 434,](#)
- [“VSAM Service,” on page 435,](#)
- [“Work Load Manager Initialization Service,” on page 435,](#)
- [“Language Environment Initialization Service,” on page 436, and](#)
- [“Multiple Catalog Support,” on page 436.](#)

## Region Controller and Logger

The following SERVICE INFO ENTRY examples are for the Region Controller and the Logger service. These two are required services, limited to one occurrence per instance. None of the TASK LOAD modules accept initialization parameters, so both specify NO\_DATA in field 10 (Service Information).

```
SERVICE INFO ENTRY = CACCNTL CNTL 0 1 1 100 4 5M 5M NO_DATA  
SERVICE INFO ENTRY = CACLOG LOGGER 1 1 1 100 1 5M 5M NO_DATA
```

## Query Processor

The following example is for a Query Processor service. There are no restrictions as to the number of instances nor the number of occurrences per instance. The only limiting factors are performance and storage availability. Field 2 (Service Name) in a Query Processor Service Info Entry references the data source name for which this Query Processor instance will process SQL commands. Field 10 (Service Information) for a Query Processor Service Info Entry is optional. It must either be NO\_DATA, signifying that no initialization data be passed to the Query Processor service, or it must be the name of a member of the SCACCONF library. If specified, any parameters contained within this member will override the default value, or the value specified in CACDSCF.

```
SERVICE INFO ENTRY = CACQP CACSAMP 2 1 10 100 0 5M 0M CACQPCF
```

## Connection Handler

The next Service Info Entry examples are for the Connection Handler, one for each of the transport layer modules. They are restricted to a single occurrence for each Service Info Entry instance, but multiple Service Info Entries may be specified, so long as this does not introduce any conflicts within or between their respective Service Information fields (field 10). It is also permissible to specify Connection Handler Service Info Entries for one, two, or all three of the different Transport Layer protocols within a single server configuration.



## Cross Memory Transport Layer

The Communication Initiator module (CACINIT) determines which transport layer the SIE is for by the first value in Field 10 (in this case, XM1). The remaining values are for the data space name followed by the queue name. This data space name/queue name pair must be unique for each client-server, client-Enterprise Server, and Enterprise Server-Server connection. The data space and the queue name fields each have a maximum length of 4 characters. See [Chapter 6, “Communication Configuration,”](#) for more information on data spaces and queues used by the Cross Memory Transport Layer.

```
SERVICE INFO ENTRY = CACINIT XMNT 2 1 1 100 4 5M 5M
XM1/CAC/CAC
```

## TCP/IP Transport Layer

In the following example, TCP is the first value of Field 10 of the SIE. The remaining two values represent the IP address of the machine the server is running on, and the port number that has been assigned to this server as a Listen port. If the TCP/IP subsystem provides support for it, the IP address is allowed to be specified as a HOST name, and the port number specified as a SERVICE name.

```
SERVICE INFO ENTRY = CACINIT TCPIP 2 1 1 100 4 5M 5M \
TCP/111.111.111.111/socket#
```

## MQ Series

For SIEs that define MQ Series, the value MQI is specified in Field 10. The remaining two values represent the name of the Queue Manager that the MQ Series Connection Handler connects to, and the name of the MQ Series Queue that the Connection Handler is listening on for connection requests.

```
SERVICE INFO ENTRY = CACINIT MQI 2 1 1 100 4 5M 10M \
MQI/SCQ1/CAC.SERVER
```

# IMS Interface Initialization Services

The next sections describe the two IMS Interface Initialization services. These services are both limited to a single SERVICE INFO ENTRY instance, and only one occurrence of that single instance is allowed.

**NOTE:** The IMS services are mutually-exclusive. Therefore, while both services can be defined, only one can be started by having its minimum task field set to 1. The non-started service must have the minimum service task field set to 0.

## CACIMSIF

The following example has the Task Name CACIMSIF. This indicates that it applies to the DBB/BMP interface. It accepts no initialization parameters from Field 10, as all of its parameters are passed in the server JCL.

```
SERVICE INFO ENTRY = CACIMSIF IMS 2 1 1 10 4 5M 5M NO_DATA
```

## CACDRA

This example has the Task Name CACDRA, indicating that it is for the DRA interface. The three values that are passed from Field 10 are the start-up table suffix, followed by the DRA user ID, and finally by the default PSB name. For more information see [Chapter 3, “Server Setup for IMS Access.”](#)

```
SERVICE INFO ENTRY = CACDRA IMS 2 1 1 10 4 5M 5M  
00 ,DRAUSER ,DEFPSB
```

## DB/2 Access

The DB2 CAF Initialization Service is used to connect to the DB2 subsystem in order to access or update DB2 data using the DB2 Call Attach Facility.

```
SERVICE INFO ENTRY = CACCAF DSN 2 1 5 1 4 5M 5M CACPLAN
```

## Datacom Initialization Service

An SIE like the following is included only if you want to access a CA-DATACOM/DB database. It is restricted to a single instance. Field 10 (Service Information) for the Datacom Initialization Service must be either NO\_DATA, signifying that no initialization data is passed to the Datacom Initialization Service, or must indicate the number of CA-DATACOM/DB task

areas to be acquired during initialization. If Field 10 is specified as 0, is specified as greater than 200, or if Field 10 is not specified, then it is set to 5. No warning messages are issued when an adjustment to this parameter is made.

```
SERVICE INFO ENTRY = CACDCI DCOM 2 1 1 50 4 5M 5M 4
```

## VSAM Service

The VSAM Service manages VSAM RPLs to allow multiple users to share access to an open file. The following default SERVICE INFO ENTRY for the VSAM Service opens VSAM datasets on first use and closes them on last use.

```
SERVICE INFO ENTRY = CACVSMS VSAMSRV 2 1 1 50 4 5M 5M NO_DATA
```

An optional setting for field10 is CLOSE\_ON\_IDLE. This enables the VSAM Service to not close the VSAM datasets when the count use goes to zero. The VSAM Service will poll all open VSAM datasets looking for a candidate which has been inactive for more than the specified idle time (Field 9). The close will occur if the VSAM dataset has exceeded the idle time.

## Work Load Manager Initialization Service

The following example is for the Work Load Manager Initialization Service. It is only included if you want to invoke the Work Load Manager System Exit, and is restricted to a single occurrence of a single instance. The service information that is passed consists of the WLM System Exit load module name, followed by a subsystem type and name parameters. These are then passed to the Work Load Manager subsystem, which uses them for classification of the work that will take place in this server. See [Chapter 13, “Utilities,”](#) and [Chapter 15, “System Exits,”](#) for additional information.

```
SERVICE INFO ENTRY = CACWLM WLM 2 1 1 10 4 5M 0M \ CACSX06
SUBSYS=STC SUBSYSNM=JES2
```

# Language Environment Initialization Service

The following example SERVICE INFO ENTRYs are for the Language Environment Initialization Service. The service is used to initialize IBM's Language Environment (CEEPIPI) or COBOL II (IGZERRE), which allows exits to be written in a high-level language. See ["Performance Considerations,"](#) on page 259.

```
SERVICE INFO ENTRY = CACLE LANGENV 2 1 1 50 5 5M 5M CEEPIPI
SERVICE INFO ENTRY = CACLE LANGENV 2 1 1 50 4 5M 5M IGZERRE
```

## Multiple Catalog Support

Multiple Catalog Support allows you to specify unique catalogs at the data source level. Using the data source name, you can specify additional catalogs, one for each data source. When a Query Processor initializes, it looks for a matching data source name (field2) and a corresponding prefix character outlined in the following naming convention.

To override the default catalog on a specific data source, add a DD statement with a label equal to the data source name, appended with a pound sign (#) for the catalog and an at sign (@) for the index file. If your data source is defined as:

```
SERVICE INFO ENTRY = CACQP SAMPL1 2 1 1 200 4 5M 5M NO_DATA
SERVICE INFO ENTRY = CACQP SAMPL2 2 1 1 200 4 5M 5M NO_DATA
```

then the following should be added to your JCL to specify unique catalogs.

```
CACCAT DD DSN=DEFAULT.CATALOG,DISP=SHR
CACINDX DD DSN=DEFAULT.INDEX,DISP=SHR
SAMPL1# DD DSN=SAMPL1.CATALOG,DISP=SHR
SAMPL1@ DD DSN=SAMPL1.INDEX,DISP=SHR
SAMPL2# DD DSN=SAMPL2.CATALOG,DISP=SHR
SAMPL2@ DD DSN=SAMPL2.INDEX,DISP=SHR
```

Each Query Processor service attempts to locate a DD name using this naming convention. If it cannot locate one, then it uses the default.

**NOTE:** OS/390 restricts DD names to eight characters. Data source names should be unique up to seven characters. For example, if your data source is defined as:

```
SERVICE INFO ENTRY = CACQP SAMPLEQUERYPROCS 2 1 1 200 4 5M
5M / NO_DATA
```

Then add the following JCL to specify unique catalogs:

```
CACCAT DD DSN=DEFAULT.CATALOG,DISP=SHR
CACINDX DD DSN=DEFAULT.INDEX,DISP=SHR
SAMPLEQ# DD DSN=SAMPL1.CATALOG,DISP=SHR
SAMPLEQ@ DD DSN=SAMPL1.INDEX,DISP=SHR
```



# Appendix E

## Meta Table Definitions

### Introduction

This appendix contains IBM DB2 Version 4 style descriptions for all of the meta data tables. For each table there is a short description of what the table contains and how many rows exist in the System Catalog. These descriptions are obtained from the REMARKS column for that table.

Following the table description are descriptions of each of the columns defined in the table. For each column, the following information is supplied:

- The column's name.
- The column's SQL data type and whether or not the column supports null values. Columns that do not support null values are identified by the clause "NOT NULL."
- A description of the column's content, and for columns that support null values, what the null value is. These descriptions were programmatically generated based on the columns REMARKS column and the NULL\_VALUE column.

The tables that follow describe each meta data table. They are presented in alphabetical order. For more information on each table's purpose, see [Table 33, "Meta Data Tables,"](#) on page 199.

**Table 81: SYSCAC.SYSADABASCOLUMNS**

Column Name	Data Type	Description
NAME	CHAR(30) NOT NULL	DB2 column. Name of the column.
TBNAME	CHAR(18) NOT NULL	DB2 column. Name of the table that contains the column.
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table that the column is in.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of column within the table. Numbers start at 1.
COLTYPE	CHAR(8) NOT NULL	DB2 column. Type of column, values are INTEGER, SMALLINT, FLOAT, CHAR, VARCHAR, LONGVAR, DECIMAL, GRAPHIC, VARG, and LONGVARG.
LENGTH	SMALLINT NOT NULL	DB2 column. For a DECIMAL data type the columns precision. for all other data types the columns length.
SCALE	SMALLINT NOT NULL	DB2 column. Scale for a DECIMAL data type, otherwise zero.
NULLS	CHAR(1) NOT NULL	DB2 column. Y/N flag identifying whether the column can contain null values.
COLCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
HIGH2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
LOW2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
UPDATES	CHAR(1)	DB2 column. For DB2 tables, the corresponding UPDATES value from the DB2 catalog. For system tables always 'N'. For all other types of tables always 'Y'. Null and default value is 'Y'.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the column. Null and default value is spaces.



**Table 81: SYSCAC.SYSADABASCOLUMNS**

Column Name	Data Type	Description
DEFAULT	CHAR(1)	DB2 column. For DB2 tables, the corresponding DEFAULT value from the DB2 catalog or for a meta data table whether the column has a NULL IS specification. For all other types of tables 'N'. Null and default value is spaces.
KEYSEQ	SMALLINT	DB2 column. The columns numeric position within the primary key, Zero if the column is not part of the primary key. Null and default value is zeros.
FOREIGNKEY	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
FLDPROC	CHAR(1)	DB2 column. Flag indicating whether the column has a field procedure associated with it. 'Y' - field procedure exists, otherwise 'N'. Null and default value is 'N'.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIELD_OFFSET	SMALLINT NOT NULL	eXadas column. Relative zero offset of the start of the column in the record or segment.
FIELD_LENGTH	SMALLINT NOT NULL	eXadas column. Physical length of the field.
NATIVE_DATA_TYPE	CHAR(1) NOT NULL	eXadas column. Native type of the the underlying column. For a description of the different data types refer to the documentation on the Meta Data Utility in the System Reference Manual.
SQL_COLUMN_LENGTH	SMALLINT NOT NULL	eXadas column. The SQL length of the column.
DB2_DATA_TYPE	SMALLINT NOT NULL	eXadas column. The DB2 SQLDA data type value for the column.
PRECISION	SMALLINT NOT NULL	eXadas column. For a DECIMAL data type the columns scale. for all other data types zero.
COLUMN_SEQUENCE_NUMBER	SMALLINT NOT NULL	eXadas column. Relative zero ordinal position of this column in the table.
USE_RECORD_LENGTH	SMALLINT	eXadas column. For variable length records, whether the entire contents of the record should be mapped to the column. A value of 1 indicates that the record length should be used. Null and default value is spaces.
FRAGMENT_ID	SMALLINT NOT NULL	eXadas column. The ID of the fragment that the column is located in.

**Table 81: SYSCAC.SYSADABASCOLUMNS**

Column Name	Data Type	Description
FRAGMENT_LEVEL	SMALLINT NOT NULL	eXadas column. The level number of the fragment that the column is located in.
OFFSET_IN_FRAGMENT	SMALLINT NOT NULL	eXadas column. Relative zero starting offset of the column within the fragment.
NULLABLE	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating whether the column supports null values.
SIGNED	CHAR(1) NOT NULL	eXadas column. Is the underlying column's data type signed or unsigned? 'Y' the data is signed, 'N' the data is unsigned.
NULL_DATA_LENGTH	SMALLINT	eXadas column. Length of the NULL IS specification for this column Null and default value is zeros.
NULL_VALUE	CHAR(16)	eXadas column. The value for this column that means the column contains a null value. Null and default value is spaces.
FIELD_PROCEDURE_NAME	CHAR(8)	eXadas column. Name of the field procedure associated with this column. Null and default value is spaces.
OCCURS_DEPENDING_ON	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag that identifies whether the column exists within an OCCURS DEPENDING ON record array. Null and default value is 'N'.
FRAGMENT_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the starting position of the fragment within the linear buffer. Null and default value is -1.
FRAGMENT_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. Length of the fragment. If fragment level is 0 the length is the length of the record or segment. For non-zero fragment levels the length of the RECORD ARRAY. Null and default value is -1.
FRAGMENT_MAXIMUM_OCCURRENCES	SMALLINT	eXadas column - obtained from fragment definition. Maximum times the fragment can occur. Null and default value is zeros.
NULL_FRAGMENT_RULE	CHAR(30)	eXadas column - obtained from fragment definition. Method used to determine a NULL occurrence in a RECORD ARRAY. Null and default value is spaces.

**Table 81: SYSCAC.SYSADABASCOLUMNS**

Column Name	Data Type	Description
OCCURS_DEPENDING_ON_COLUMN	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the name of the column that identifies the number of occurrences that exist. Null and default value is spaces.
OCCURS_DEPENDING_ON_COLNO	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the COLUMN_SEQUENCE_NUMBER of the column that identifies the number of occurrences that exist. Null and default value is zeros.
FRAGMENT_NULL_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the length of the NULL IS specification for the fragment. Null and default value is zeros.
FRAGMENT_NULL_VALUE	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the value that identifies an instance as being null. Null and default value is spaces.
FORMAT_BUFFER_FIELD_ID	CHAR(20) NOT NULL	eXadas column. ADABAS format buffer field id.
FDT_DEFINITION_TYPE	CHAR(1) NOT NULL	eXadas column. ADABAS FDT definition type code for this column.
FDT_FIELD_NAME	CHAR(2) NOT NULL	eXadas column. ADABAS FDT field name for this column.
FDT_OPTION	CHAR(24) NOT NULL	eXadas column. ADABAS FDT option list. One or more of the following values: DE, FI, MU, NU, PE, PP, PS and UQ. Refer to the ADABAS documentation for descriptions of these codes.
FDT_LEVEL_NUMBER	SMALLINT	eXadas column. ADABAS FDT level number for this column. Null and default value is zeros.
FDT_LENGTH	SMALLINT NOT NULL	eXadas column. ADABAS FDT length value for this column.
FDT_FORMAT_TYPE	CHAR(3)	eXadas column, ADABAS FDT format type code for this column. One of the following: A, B, F, G, P or U, or superdescriptor number for a subfield. Null and default value is spaces.
FDT_OPTION2	CHAR(8)	eXadas column. ADABAS FDT option2 list. One or more of the following values: LA, NN and NC. Refer to the ADABAS documentation for descriptions of these codes. Null and default value is spaces.

**Table 81: SYSCAC.SYSADABASCOLUMNS**

Column Name	Data Type	Description
DATE_FIELD	CHAR(1)	eXadas column. Y/N flag that identifies whether this column is defined as a date field to ADABAS. Null and default value is spaces.
TIME_FIELD	CHAR(1)	eXadas column. Y/N flag that identifies whether this column is defined as a time field to ADABAS. Null and default value is spaces.
DATE_FORMAT	CHAR(30)	eXadas column. Date format string if this column is mapped against an ADABAS date field. Null and default value is spaces.
TIME_FORMAT	CHAR(30)	eXadas column. Date format string if this column is mapped against an ADABAS time field. Null and default value is spaces.
MAIN_REDEFINTION	CHAR(1)	eXadas column. Y/N flag that identifies whether the ADABAS field referenced by this column is a main redefinition within Predict. Null and default value is spaces.
REDEFINTION	CHAR(1)	eXadas column. Y/N flag that identifies whether the ADABAS field referenced by this column is redefined within Predict. Null and default value is spaces.
REDEFINITION_OFFSET	SMALLINT	eXadas column. For a column that is identified as a redefinition, the relative zero starting offset of the column. Null and default value is zeros.
MAIN_REDEFINITION_OFFSET	SMALLINT	eXadas column. For a column that is identified as a main redefinition, the relative zero starting offset of the column. Null and default value is zeros.
TRANSFER_FIELD_LENGTH	SMALLINT	eXadas column. Length of the data transferred to the linear buffer (for use by the QP) for this column. Null and default value is zeros.
ISN	CHAR(1)	eXadas column. Y/N flag that identifies whether the columns represents the ISN. Null and default value is spaces.

**Table 82: SYSCAC.SYSADABASINDEXES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of index.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the index.
TBNAME	CHAR(18) NOT NULL	DB2 column. Name of the table that the index is referencing.
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
UNIQUERULE	CHAR(1) NOT NULL	DB2 column. Whether the index is unique. 'U' yes or 'D' no.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns that make up the key.
CLUSTERING	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
CLUSTERED	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
DBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
ISOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
DBNAME	CHAR(8) NOT NULL	DB2 column. Not used by eXadas.
INDEXSPACE	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIRSTKEYCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
FULLKEYCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
NLEAF	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
NLEVELS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.

**Table 82: SYSCAC.SYSADABASINDEXES**

Column Name	Data Type	Description
BPOOL	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
PGSIZE	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
ERASERULE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
DSETPASS	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLOSERULE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'Y'.
SPACE	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
CLUSTERRATIO	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the index.
SUPER_DESCRIPTOR_COLNO	SMALLINT	eXadas column. The COLNO of the super descriptor column definition for this table. Null and default value is zeros.

**Table 83: SYSCAC.SYSADABASKEYS**

Column Name	Data Type	Description
IXNAME	CHAR(18) NOT NULL	DB2 column. Name of the index.
IXCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the index.
COLNAME	CHAR(30) NOT NULL	DB2 column. Name of the column for this key.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of the column in the table.
COLSEQ	SMALLINT NOT NULL	DB2 column. Position of this column within the key.

**Table 83: SYSCAC.SYSADABASKEYS**

Column Name	Data Type	Description
ORDERING	CHAR(1) NOT NULL	DB2 column. Sort order of this column within the key. 'A' ascending or 'D' decending.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
FIELD_NAME	CHAR(2) NOT NULL	eXadas column. ADABAS field name associated with the key column.
START_POSITION	SMALLINT NOT NULL	eXadas column. Relative zero starting position of the key column within the ADABAS field.
END_POSITION	SMALLINT NOT NULL	eXadas column. Relative zero ending position of the key column within the ADABAS field.

**Table 84: SYSCAC.SYSADABASTABLES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of the table.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
TYPE	CHAR(1) NOT NULL	DB2 column. Type of object. 'T' for a table or 'V' for a view.
DBNAME	CHAR(8) NOT NULL	DB2 column. Identifies the type of DBMS that the table is referencing. Values are \$ADABAS, \$CFI (System Table), \$DATACOM, \$DB2, \$IAM, \$IMS, \$IDMS, \$MDS (Meta Data Table) \$SEQUENT and \$VSAM.
TSNAME	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
DBID	SMALLINT	DB2 column. Internal identifier of the database. Zero if the the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Internal identifier of the table. Zero if the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns defined for this table.

**Table 84: SYSCAC.SYSADABASTABLES**

Column Name	Data Type	Description
EDPROC	CHAR(8)	DB2 column. Name of the edit procedure. Blank if the row describes a view or the table does not use an edit procedure. Only populated for IDMS tables that use compression routines. Null and default value is spaces.
VALPROC	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERTYPE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERRID	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
CARD	INTEGER	DB2 column. Number of rows in the table. Null and default value is -1.
NPAGES	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
PCTPAGES	SMALLINT	DB2 column. Not used by eXadas. Null and default value is -1.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the table. Null and default value is spaces.
PARENTS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CHILDREN	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
KEYCOLUMNS	SMALLINT	DB2 column. Number of columns that make up the table's primary key. Only maintained for system tables. For all other tables 0. Null and default value is zeros.
RECLENGTH	SMALLINT NOT NULL	DB2 column. Number of bytes in a row. This is also the size of the linear buffer allocated for this table.
STATUS	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
KEYOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.



**Table 84: SYSCAC.SYSADABASTABLES**

Column Name	Data Type	Description
CHECKFLAG	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKRID	CHAR(4)	DB2 column. Not used by eXadas. Null and default value is spaces.
AUDIT	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the table.
LOCATION	CHAR(16)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBCREATOR	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBNAME	CHAR(18)	DB2 column. Not used by eXadas. Null and default value is spaces.
FRAGMENTS	SMALLINT NOT NULL	eXadas column. Number of fragments used to manage this table.
DBMS	CHAR(8) NOT NULL	eXadas column. Identifies the type of database. Always 'ADABAS'.
DATA_CAPTURE	CHAR(1)	DB2 column. Identifies whether the table has been marked for xSync data capture. A value of 'Y' indicates that changes will be captured. Null and default value is spaces.
SORT_KEYS	SMALLINT	eXadas column. Total number of sort keys for all indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
INDEXES	SMALLINT	eXadas column. Total number of indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
FILE_NUMBER	SMALLINT NOT NULL	eXadas column. ADABAS file number that this table is referencing.
VIEW_NAME	CHAR(40)	eXadas column. PREDICT view name. Null and default value is spaces.
VARIABLE_LENGTH_COLUMNS	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating whether this table contains ADABAS columns that are variable length.

**Table 84: SYSCAC.SYSADABASTABLES**

Column Name	Data Type	Description
DB_ID	SMALLINT NOT NULL	eXadas column. ADABAS DBID number that this table is referencing.
FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with this ADABAS table. Null and default value is zeros.
VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the table is variable length. Null and default value is spaces.

**Table 85: SYSCAC.SYSDATACOMCOLUMNS**

Column Name	Data Type	Description
NAME	CHAR(30) NOT NULL	DB2 column. Name of the column.
TBNAME	CHAR(18) NOT NULL	DB2 column. Name of the table that contains the column.
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table that the column is in.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of column within the table. Numbers start at 1.
COLTYPE	CHAR(8) NOT NULL	DB2 column. Type of column, values are INTEGER, SMALLINT, FLOAT, CHAR, VARCHAR, LONGVAR, DECIMAL, GRAPHIC, VARG, and LONGVARG.
LENGTH	SMALLINT NOT NULL	DB2 column. For a DECIMAL data type the columns precision. for all other data types the columns length.
SCALE	SMALLINT NOT NULL	DB2 column. Scale for a DECIMAL data type, otherwise zero.
NULLS	CHAR(1) NOT NULL	DB2 column. Y/N flag identifying whether the column can contain null values.
COLCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
HIGH2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
LOW2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.

**Table 85: SYSCAC.SYSDATACOMCOLUMNS**

Column Name	Data Type	Description
UPDATES	CHAR(1)	DB2 column. For DB2 tables, the corresponding UPDATES value from the DB2 catalog. For system tables always 'N'. For all other types of tables always 'Y'. Null and default value is 'Y'.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the column. Null and default value is spaces.
DEFAULT	CHAR(1)	DB2 column. For DB2 tables, the corresponding DEFAULT value from the DB2 catalog or for a meta data table whether the column has a NULL IS specification. For all other types of tables 'N'. Null and default value is spaces.
KEYSEQ	SMALLINT	DB2 column. The columns numeric position within the primary key, Zero if the column is not part of the primary key. Null and default value is zeros.
FOREIGNKEY	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
FLDPROC	CHAR(1)	DB2 column. Flag indicating whether the column has a field procedure associated with it. 'Y' - field procedure exists, otherwise 'N'. Null and default value is 'N'.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIELD_OFFSET	SMALLINT NOT NULL	eXadas column. Relative zero offset of the start of the column in the rexord or segment.
FIELD_LENGTH	SMALLINT NOT NULL	eXadas column. Physical length of the field.
NATIVE_DATA_TYPE	CHAR(1) NOT NULL	eXadas column. Native type of the the underlying column. For a description of the different data types refer to the documentation on the Meta Data Utility in the System Reference Manual.
SQL_COLUMN_LENGTH	SMALLINT NOT NULL	eXadas column. The SQL length of the column.
DB2_DATA_TYPE	SMALLINT NOT NULL	eXadas column. The DB2 SQLDA data type value for the column.
PRECISION	SMALLINT NOT NULL	eXadas column. For a DECIMAL data type the columns scale. for all other data types zero.

**Table 85: SYSCAC.SYSDATACOMCOLUMNS**

Column Name	Data Type	Description
COLUMN_SEQUENCE_NUMBER	SMALLINT NOT NULL	eXadas column. Relative zero ordinal position of this column in the table.
USE_RECORD_LENGTH	SMALLINT	eXadas column. For variable length records, whether the entire contents of the record should be mapped to the column. A value of 1 indicates that the record length should be used. Null and default value is spaces.
FRAGMENT_ID	SMALLINT NOT NULL	eXadas column. The ID of the fragment that the column is located in.
FRAGMENT_LEVEL	SMALLINT NOT NULL	eXadas column. The level number of the fragment that the column is located in.
OFFSET_IN_FRAGMENT	SMALLINT NOT NULL	eXadas column. Relative zero starting offset of the column within the fragment.
NULLABLE	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating whether the column supports null values.
SIGNED	CHAR(1) NOT NULL	eXadas column. Is the underlying column's data type signed or unsigned? 'Y' the data is signed, 'N' the data is unsigned.
NULL_DATA_LENGTH	SMALLINT	eXadas column. Length of the NULL IS specification for this column Null and default value is zeros.
NULL_VALUE	CHAR(16)	eXadas column. The value for this column that means the column contains a null value. Null and default value is spaces.
FIELD_PROCEDURE_NAME	CHAR(8)	eXadas column. Name of the field procedure associated with this column. Null and default value is spaces.
OCCURS_DEPENDING_ON	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag that identifies whether the column exists within an OCCURS DEPENDING ON record array. Null and default value is 'N'.
FRAGMENT_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the starting position of the fragment within the linear buffer. Null and default value is -1.
FRAGMENT_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. Length of the fragment. If fragment level is 0 the length is the length of the record or segment. For non-zero fragment levels the length of the RECORD ARRAY. Null and default value is -1.

**Table 85: SYSCAC.SYSDATACOMCOLUMNS**

Column Name	Data Type	Description
FRAGMENT_MAXIMUM_OCCURRENCES	SMALLINT	eXadas column - obtained from fragment definition. Maximum times the fragment can occur. Null and default value is zeros.
NULL_FRAGMENT_RULE	CHAR(30)	eXadas column - obtained from fragment definition. Method used to determine a NULL occurrence in a RECORD ARRAY. Null and default value is spaces.
OCCURS_DEPENDING_ON_COLUMN	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the name of the column that identifies the number of occurrences that exist. Null and default value is spaces.
OCCURS_DEPENDING_ON_COLNO	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the COLUMN_SEQUENCE_NUMBER of the column that identifies the number of occurrences that exist. Null and default value is zeros.
FRAGMENT_NULL_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the length of the NULL IS specification for the fragment. Null and default value is zeros.
FRAGMENT_NULL_VALUE	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the value that identifies an instance as being null. Null and default value is spaces.
DATACOM_NAME	CHAR(5) NOT NULL	eXadas column. Name of the DATACOM element that contains this column.
SECURITY_CODE	CHAR(1)	eXadas column. DATACOM security code that is required to access this element or any portion of it. Null and default value is spaces.
DISPLACEMENT_IN_TABLE	SMALLINT NOT NULL	eXadas column. Offset from the beginning of the DATACOM table to the 1st byte of the element containing this column.
ELEMENT_LENGTH	SMALLINT NOT NULL	eXadas column. Length of the DATACOM element containing this column.
DISPLACEMENT_IN_ELEMENT	SMALLINT NOT NULL	eXadas column. Offset from the beginning of the DATACOM element to the 1st byte of the column.
DATACOM_TYPE	CHAR(1) NOT NULL	eXadas column. DATACOM TYPE attribute of the FIELD entity-occurrence.

**Table 85: SYSCAC.SYSDATACOMCOLUMNS**

Column Name	Data Type	Description
NUMERIC_TYPE	CHAR(1)	eXadas column. DATACOM TYPE-NUMERIC attribute of the FIELD entity-occurrence. Null and default value is spaces.
SIGN	CHAR(1)	eXadas column. DATACOM SIGN attribute of the FIELD entity-occurrence. Null and default value is spaces.
JUSTIFICATION	CHAR(1)	eXadas column. DATACOM JUSTIFICATION attribute of the FIELD entity-occurrence. Null and default value is spaces.
DECIMALS	SMALLINT NOT NULL	eXadas column. DATACOM DECIMALS attribute of the FIELD entity-occurrence.
DATACOM_PRECISION	SMALLINT NOT NULL	eXadas column. DATACOM PRECISION attribute of the FIELD entity-occurrence.

**Table 86: SYSCAC.SYSDATACOMINDEXES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of index.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the index.
TBNAME	CHAR(18) NOT NULL	DB2 column. Name of the table that the index is referencing.
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
UNIQUERULE	CHAR(1) NOT NULL	DB2 column. Whether the index is unique. 'U' yes or 'D' no.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns that make up the key.
CLUSTERING	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
CLUSTERED	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
DBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.

**Table 86: SYSCAC.SYSDATACOMINDEXES**

Column Name	Data Type	Description
OBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
ISOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
DBNAME	CHAR(8) NOT NULL	DB2 column. Not used by eXadas.
INDEXSPACE	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIRSTKEYCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
FULLKEYCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
NLEAF	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
NLEVELS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
BPOOL	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
PGSIZE	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
ERASERULE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
DSETPASS	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLOSERULE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'Y'.
SPACE	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
CLUSTERRATIO	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the index.

**Table 87: SYSCAC.SYSDATACOMKEYS**

Column Name	Data Type	Description
IXNAME	CHAR(18) NOT NULL	DB2 column. Name of the index.
IXCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the index.
COLNAME	CHAR(30) NOT NULL	DB2 column. Name of the column for this key.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of the column in the table.
COLSEQ	SMALLINT NOT NULL	DB2 column. Position of this column within the key.
ORDERING	CHAR(1) NOT NULL	DB2 column. Sort order of this column within the key. 'A' ascending or 'D' decending.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.

**Table 88: SYSCAC.SYSDATACOMTABLES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of the table.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
TYPE	CHAR(1) NOT NULL	DB2 column. Type of object. 'T' for a table or 'V' for a view.
DBNAME	CHAR(8) NOT NULL	DB2 column. Identifies the type of DBMS that the table is referencing. Values are \$ADABAS, \$CFI (System Table), \$DATACOM, \$DB2, \$IAM, \$IMS, \$IDMS, \$MDS (Meta Data Table) \$SEQUENT and \$VSAM.
TSNAME	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
DBID	SMALLINT	DB2 column. Internal identifier of the database. Zero if the the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.



**Table 88: SYSCAC.SYSDATACOMTABLES**

Column Name	Data Type	Description
OBID	SMALLINT	DB2 column. Internal identifier of the table. Zero if the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns defined for this table.
EDPROC	CHAR(8)	DB2 column. Name of the edit procedure. Blank if the row describes a view or the table does not use a edit procedure. Only populated for IDMS tables that use compression routines. Null and default value is spaces.
VALPROC	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERTYPE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERRID	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
CARD	INTEGER	DB2 column. Number of rows in the table. Null and default value is -1.
NPAGES	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
PCTPAGES	SMALLINT	DB2 column. Not used by eXadas. Null and default value is -1.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the table. Null and default value is spaces.
PARENTS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CHILDREN	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
KEYCOLUMNS	SMALLINT	DB2 column. Number of columns that make up the tables primary key. Only maintained for system tables. For all other tables 0. Null and default value is zeros.
RECLENGTH	SMALLINT NOT NULL	DB2 column. Number of bytes in a row. This ia also the size of the linear buffer allocated for this table.

**Table 88: SYSCAC.SYSDATACOMTABLES**

Column Name	Data Type	Description
STATUS	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
KEYOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKFLAG	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKRID	CHAR(4)	DB2 column. Not used by eXadas. Null and default value is spaces.
AUDIT	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the table.
LOCATION	CHAR(16)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBCREATOR	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBNAME	CHAR(18)	DB2 column. Not used by eXadas. Null and default value is spaces.
FRAGMENTS	SMALLINT NOT NULL	eXadas column. Number of fragments used to manage this table.
DBMS	CHAR(8) NOT NULL	eXadas column. Identifies the type of database. Always 'DATACOM'.
DATA_CAPTURE	CHAR(1)	DB2 column. Identifies whether the table has been marked for xSync data capture. A value of 'Y' indicates that changes will be captured. Null and default value is spaces.
SORT_KEYS	SMALLINT	eXadas column. Total number of sort keys for all indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
INDEXES	SMALLINT	eXadas column. Total number of indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.

**Table 88: SYSCAC.SYSDATACOMTABLES**

Column Name	Data Type	Description
DATACOM_ID	SMALLINT NOT NULL	eXadas column. DATACOM database ID. that is referenced by this table.
ENTITY_OCCURRENCE_NAME	CHAR(32) NOT NULL	eXadas column. DATACOM database occurrence name that is referenced by this table.
AREA_OCCURRENCE_NAME	CHAR(32) NOT NULL	eXadas column. DATACOM area occurrence name that is referenced by this table.
AREA_NAME	CHAR(3) NOT NULL	eXadas column. DATACOM area name that is referenced by this table.
TABLE_OCCURRENCE_NAME	CHAR(32) NOT NULL	eXadas column. DATACOM table occurrence name that is referenced by this table.
TABLE_NAME	CHAR(3) NOT NULL	eXadas column. DATACOM table name that is referenced by this table.
VERSION_STATUS	CHAR(4) NOT NULL	eXadas column. DATACOM status and version identifier for this table.
TABLE_LENGTH	SMALLINT NOT NULL	eXadas column. The length of the table as defined to DATACOM.
TABLE_ID	SMALLINT NOT NULL	eXadas column. DATACOM_ID table identifier that is referenced by this table.
DATABASE_ID	SMALLINT NOT NULL	eXadas column. DATACOM_ID database identifier referenced by this DATACOM table.
NUMBER_OF_ELEMENTS	SMALLINT NOT NULL	eXadas column. Number of DATACOM elements referenced by this table.
URT_NAME	CHAR(8) NOT NULL	eXadas column. URT load module name used to access this table.
FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with this DATACOM table. Null and default value is zeros.
VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the DATACOM table is variable length. Null and default value is spaces.

**Table 89: SYSCAC.SYSDDB2COLUMNS**

Column Name	Data Type	Description
NAME	CHAR(30) NOT NULL	DB2 column. Name of the column.
TBNAME	CHAR(18) NOT NULL	DB2 column. Name of the table that contains the column.
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table that the column is in.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of column within the table. Numbers start at 1.
COLTYPE	CHAR(8) NOT NULL	DB2 column. Type of column, values are INTEGER, SMALLINT, FLOAT, CHAR, VARCHAR, LONGVAR, DECIMAL, GRAPHIC, VARG, and LONGVARG.
LENGTH	SMALLINT NOT NULL	DB2 column. For a DECIMAL data type the columns precision. for all other data types the columns length.
SCALE	SMALLINT NOT NULL	DB2 column. Scale for a DECIMAL data type, otherwise zero.
NULLS	CHAR(1) NOT NULL	DB2 column. Y/N flag identifying whether the column can contain null values.
COLCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
HIGH2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
LOW2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
UPDATES	CHAR(1)	DB2 column. For DB2 tables, the corresponding UPDATES value from the DB2 catalog. For system tables always 'N'. For all other types of tables always 'Y'. Null and default value is 'Y'.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the column. Null and default value is spaces.
DEFAULT	CHAR(1)	DB2 column. For DB2 tables, the corresponding DEFAULT value from the DB2 catalog or for a meta data table whether the column has a NULL IS specification. For all other types of tables 'N'. Null and default value is spaces.

**Table 89: SYSCAC.SYSDB2COLUMNS**

Column Name	Data Type	Description
KEYSEQ	SMALLINT	DB2 column. The columns numeric position within the primary key, Zero if the column is not part of the primary key. Null and default value is zeros.
FOREIGNKEY	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
FLDPROC	CHAR(1)	DB2 column. Flag indicating whether the column has a field procedure associated with it. 'Y' - field procedure exists, otherwise 'N'. Null and default value is 'N'.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIELD_LENGTH	SMALLINT NOT NULL	eXadas column. Physical length of the field.
NATIVE_DATA_TYPE	CHAR(1) NOT NULL	eXadas column. Native type of the the underlying column. For a description of the different data types refer to the documentation on the Meta Data Utility in the System Reference Manual.
SQL_COLUMN_LENGTH	SMALLINT NOT NULL	eXadas column. The SQL length of the column.
DB2_DATA_TYPE	SMALLINT NOT NULL	eXadas column. The DB2 SQLDA data type value for the column.
COLUMN_SEQUENCE_NUMBER	SMALLINT NOT NULL	eXadas column. Relative zero ordinal position of this column in the table.

**Table 90: SYSCAC.SYSDB2INDEXES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of index.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the index.
TBNAME	CHAR(18) NOT NULL	DB2 column. Name of the table that the index is referencing.
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
UNIQUERULE	CHAR(1) NOT NULL	DB2 column. Whether the index is unique. 'U' yes or 'D' no.

**Table 90: SYSCAC.SYSDB2INDEXES**

Column Name	Data Type	Description
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns that make up the key.
CLUSTERING	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
CLUSTERED	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
DBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
ISOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
DBNAME	CHAR(8) NOT NULL	DB2 column. Not used by eXadas.
INDEXSPACE	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIRSTKEYCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
FULLKEYCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
NLEAF	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
NLEVELS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
BPOOL	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
PGSIZE	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
ERASERULE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
DSETPASS	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLOSERULE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'Y'.

**Table 90: SYSCAC.SYSDB2INDEXES**

Column Name	Data Type	Description
SPACE	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
CLUSTERRATIO	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the index.

**Table 91: SYSCAC.SYSDB2KEYS**

Column Name	Data Type	Description
IXNAME	CHAR(18) NOT NULL	DB2 column. Name of the index.
IXCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the index.
COLNAME	CHAR(30) NOT NULL	DB2 column. Name of the column for this key.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of the column in the table.
COLSEQ	SMALLINT NOT NULL	DB2 column. Position of this column within the key.
ORDERING	CHAR(1) NOT NULL	DB2 column. Sort order of this column within the key. 'A' ascending or 'D' decending.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.

**Table 92: SYSCAC.SYSDB2TABLES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of the table.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.

**Table 92: SYSCAC.SYSDB2TABLES**

Column Name	Data Type	Description
TYPE	CHAR(1) NOT NULL	DB2 column. Type of object. 'T' for a table or 'V' for a view.
DBNAME	CHAR(8) NOT NULL	DB2 column. Identifies the type of DBMS that the table is referencing. Values are \$ADABAS, \$CFI (System Table), \$DATACOM, \$DB2, \$IAM, \$IMS, \$IDMS, \$MDS (Meta Data Table) \$SEQUENT and \$VSAM.
TSNAME	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
DBID	SMALLINT	DB2 column. Internal identifier of the database. Zero if the the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Internal identifier of the table. Zero if the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns defined for this table.
EDPROC	CHAR(8)	DB2 column. Name of the edit procedure. Blank if the row describes a view or the table does not use a edit procedure. Only populated for IDMS tables that use compression routines. Null and default value is spaces.
VALPROC	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERTYPE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERRID	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
CARD	INTEGER	DB2 column. Number of rows in the table. Null and default value is -1.
NPAGES	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
PCTPAGES	SMALLINT	DB2 column. Not used by eXadas. Null and default value is -1.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.



**Table 92: SYSCAC.SYSDB2TABLES**

Column Name	Data Type	Description
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the table. Null and default value is spaces.
PARENTS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CHILDREN	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
KEYCOLUMNS	SMALLINT	DB2 column. Number of columns that make up the tables primary key. Only maintained for system tables. For all other tables 0. Null and default value is zeros.
RECLENGTH	SMALLINT NOT NULL	DB2 column. Number of bytes in a row. This ia also the size of the linear buffer allocated for this table.
STATUS	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
KEYOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKFLAG	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKRID	CHAR(4)	DB2 column. Not used by eXadas. Null and default value is spaces.
AUDIT	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the table.
LOCATION	CHAR(16)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBCREATOR	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBNAME	CHAR(18)	DB2 column. Not used by eXadas. Null and default value is spaces.
FRAGMENTS	SMALLINT NOT NULL	eXadas column. Number of fragments used to mangage this table.
DBMS	CHAR(8) NOT NULL	eXadas column. Identifies the type of database. Always 'DB2'.

**Table 92: SYSCAC.SYSDB2TABLES**

Column Name	Data Type	Description
DATA_CAPTURE	CHAR(1)	DB2 column. Identifies whether the table has been marked for xSync data capture. A value of 'Y' indicates that changes will be captured. Null and default value is spaces.
SORT_KEYS	SMALLINT	eXadas column. Total number of sort keys for all indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
INDEXES	SMALLINT	eXadas column. Total number of indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
SUBSYSTEM_ID	CHAR(4) NOT NULL	eXadas column. DB2 sub-system ID that the DB2 table was imported from.
DB2_CREATOR	CHAR(8) NOT NULL	eXadas column. DB2 creator ID that this table was imported from.
DB2_TABLE_NAME	CHAR(20) NOT NULL	eXadas column. DB2 table name that this table was imported from.
DB2_TABLE_TYPE	CHAR(1) NOT NULL	eXadas column. The type of DB2 table that this table was imported from. T, V or A.

**Table 93: SYSCAC.SYSIDMS\_INDEXES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of the table.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
TYPE	CHAR(1) NOT NULL	DB2 column. Type of object. 'T' for a table or 'V' for a view.
DBNAME	CHAR(8) NOT NULL	DB2 column. Identifies the type of DBMS that the table is referencing. Values are \$ADABAS, \$CFI (System Table), \$DATACOM, \$DB2, \$IAM, \$IMS, \$IDMS, \$MDS (Meta Data Table) \$SEQUENT and \$VSAM.
TSNAME	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.

**Table 93: SYSCAC.SYSIDMS\_INDEXES**

Column Name	Data Type	Description
DBID	SMALLINT	DB2 column. Internal identifier of the database. Zero if the the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Internal identifier of the table. Zero if the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns defined for this table.
EDPROC	CHAR(8)	DB2 column. Specifies the name of an IDMS compression/decompression routine Null and default value is spaces.
VALPROC	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERTYPE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERRID	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
CARD	INTEGER	DB2 column. Number of rows in the table. Null and default value is -1.
NPAGES	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
PCTPAGES	SMALLINT	DB2 column. Not used by eXadas. Null and default value is -1.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the table. Null and default value is spaces.
PARENTS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CHILDREN	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
KEYCOLUMNS	SMALLINT	DB2 column. Number of columns that make up the tables primary key. Only maintained for system tables. For all other tables 0. Null and default value is zeros.

**Table 93: SYSCAC.SYSIDMS\_INDEXES**

Column Name	Data Type	Description
RECLENGTH	SMALLINT NOT NULL	DB2 column. Total length of all records associated with this table.
STATUS	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
KEYOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKFLAG	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKRID	CHAR(4)	DB2 column. Not used by eXadas. Null and default value is spaces.
AUDIT	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the table.
LOCATION	CHAR(16)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBCREATOR	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBNAME	CHAR(18)	DB2 column. Not used by eXadas. Null and default value is spaces.
FRAGMENTS	SMALLINT NOT NULL	eXadas column. Number of fragments used to manage this table.
DBMS	CHAR(8) NOT NULL	eXadas column. Identifies the type of database. Always 'IDMS'.
DATA_CAPTURE	CHAR(1)	DB2 column. Identifies whether the table has been marked for xSync data capture. A value of 'Y' indicates that changes will be captured. Null and default value is spaces.
SORT_KEYS	SMALLINT	eXadas column. Total number of sort keys for all indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.

**Table 93: SYSCAC.SYSIDMS\_INDEXES**

Column Name	Data Type	Description
INDEXES	SMALLINT	eXadas column. Total number of indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
DATABASE_NAME	CHAR(8)	eXadas column. Name of IDMS database used to map this table. Null and default value is spaces.
SCHEMA_NAME	CHAR(8) NOT NULL	eXadas column. IDMS Schema Name used to map this table.
SUB_SCHEMA_NAME	CHAR(8) NOT NULL	eXadas column. IDMS sub-schema name used to map this table.
SUB_SCHEMA_VERSION	SMALLINT NOT NULL	eXadas column. The IDMS sub-schema version number used to map this table.
RECORD_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS records referenced by this table. Maximum of 10 records supported.
SET_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS sets referenced by this table. Maximum of 9 sets supported.
INDEX_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS indexes referenced by this table. Maximum of 10 indexes supported.
AREA_COUNT	SMALLINT NOT NULL	eXadas column. Total number of areas associated with this table. Maximum of 10 areas supported.
ACCESS_MODULE	CHAR(8) NOT NULL	eXadas column. Name of IDMS load module used to access this table.
ACCESS_METHOD	CHAR(4)	eXadas column. Method used to access the table. Values are IDMS, KSDS, RRDS or ESDS. Null and default value is spaces.
VARIABLE_LENGTH_TABLE	CHAR(1) NOT NULL	eXadas column. Flag indicating whether table is variable length. 'Y' if variable length or 'N' for fixed length tables.
MINIMUM_LENGTH	SMALLINT	eXadas column. Minimum length for variable length tables. Null and default value is zeros.
PARENT_DB_KEY_OFFSET	SMALLINT	eXadas column. For tables marked for data capture, the offset of the parent database key in the IMDS journal. Null and default value is -1.
PAGE_AREA_MIN	INTEGER	eXadas column. For tables marked for data capture, the low page area dbkey value. Null and default value is zeros.

**Table 93: SYSCAC.SYSIDMS\_INDEXES**

Column Name	Data Type	Description
PAGE_AREA_MAX	INTEGER	eXadas column. For tables marked for data capture, the high page area dbkey value. Null and default value is zeros.
INDEX_NAME	CHAR(16) NOT NULL	eXadas column. Name of the index associated with this table.
INDEX_KEY_LENGTH	SMALLINT NOT NULL	eXadas column. Length of the index key associated with this table.
INDEX_SORT_ORDER	CHAR(1) NOT NULL	eXadas column. Sorting order of the index associated with this table. 'A' ascending or 'D' for descending.
INDEX_SET_TYPE	CHAR(30)	eXadas column. Method used to access the index associated with this table. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.

**Table 94: SYSCAC.SYSIDMSAREAS**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of the table.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
TYPE	CHAR(1) NOT NULL	DB2 column. Type of object. 'T' for a table or 'V' for a view.
DBNAME	CHAR(8) NOT NULL	DB2 column. Identifies the type of DBMS that the table is referencing. Values are \$ADABAS, \$CFI (System Table), \$DATACOM, \$DB2, \$IAM, \$IMS, \$IDMS, \$MDS (Meta Data Table) \$SEQUENT and \$VSAM.
TSNAME	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
DBID	SMALLINT	DB2 column. Internal identifier of the database. Zero if the the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Internal identifier of the table. Zero if the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.

**Table 94: SYSCAC.SYSIDMSAREAS**

Column Name	Data Type	Description
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns defined for this table.
EDPROC	CHAR(8)	DB2 column. Specifies the name of an IDMS compression/decompression routine Null and default value is spaces.
VALPROC	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERTYPE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERRID	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
CARD	INTEGER	DB2 column. Number of rows in the table. Null and default value is -1.
NPAGES	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
PCTPAGES	SMALLINT	DB2 column. Not used by eXadas. Null and default value is -1.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the table. Null and default value is spaces.
PARENTS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CHILDREN	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
KEYCOLUMNS	SMALLINT	DB2 column. Number of columns that make up the tables primary key. Only maintained for system tables. For all other tables 0. Null and default value is zeros.
RECLENGTH	SMALLINT NOT NULL	DB2 column. Total length of all records associated with this table.
STATUS	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
KEYOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.

**Table 94: SYSCAC.SYSIDMSAREAS**

Column Name	Data Type	Description
CHECKFLAG	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKRID	CHAR(4)	DB2 column. Not used by eXadas. Null and default value is spaces.
AUDIT	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the table.
LOCATION	CHAR(16)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBCREATOR	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBNAME	CHAR(18)	DB2 column. Not used by eXadas. Null and default value is spaces.
FRAGMENTS	SMALLINT NOT NULL	eXadas column. Number of fragments used to manage this table.
DBMS	CHAR(8) NOT NULL	eXadas column. Identifies the type of database. Always 'IDMS'.
DATA_CAPTURE	CHAR(1)	DB2 column. Identifies whether the table has been marked for xSync data capture. A value of 'Y' indicates that changes will be captured. Null and default value is spaces.
SORT_KEYS	SMALLINT	eXadas column. Total number of sort keys for all indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
INDEXES	SMALLINT	eXadas column. Total number of indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
DATABASE_NAME	CHAR(8)	eXadas column. Name of IDMS database used to map this table. Null and default value is spaces.
SCHEMA_NAME	CHAR(8) NOT NULL	eXadas column. IDMS Schema Name used to map this table.
SUB_SCHEMA_NAME	CHAR(8) NOT NULL	eXadas column. IDMS sub-schema name used to map this table.



**Table 94: SYSCAC.SYSIDMSAREAS**

Column Name	Data Type	Description
SUB_SCHEMA_VERSION	SMALLINT NOT NULL	eXadas column. The IDMS sub-schema version number used to map this table.
RECORD_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS records referenced by this table. Maximum of 10 records supported.
SET_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS sets referenced by this table. Maximum of 9 sets supported.
INDEX_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS indexes referenced by this table. Maximum of 10 indexes supported.
AREA_COUNT	SMALLINT NOT NULL	eXadas column. Total number of areas associated with this table. Maximum of 10 areas supported.
ACCESS_MODULE	CHAR(8) NOT NULL	eXadas column. Name of IDMS load module used to access this table.
ACCESS_METHOD	CHAR(4)	eXadas column. Method used to access the table. Values are IDMS, KSDS, RRDS or ESDS. Null and default value is spaces.
VARIABLE_LENGTH_TABLE	CHAR(1) NOT NULL	eXadas column. Flag indicating whether table is variable length. 'Y' if variable length or 'N' for fixed length tables.
MINIMUM_LENGTH	SMALLINT	eXadas column. Minimum length for variable length tables. Null and default value is zeros.
PARENT_DB_KEY_OFFSET	SMALLINT	eXadas column. For tables marked for data capture, the offset of the parent database key in the IMDS journal. Null and default value is -1.
PAGE_AREA_MIN	INTEGER	eXadas column. For tables marked for data capture, the low page area dbkey value. Null and default value is zeros.
PAGE_AREA_MAX	INTEGER	eXadas column. For tables marked for data capture, the high page area dbkey value. Null and default value is zeros.
AREA_NAME	CHAR(16) NOT NULL	eXadas column. Name of the area associated with table.

**Table 95: SYSCAC.SYSCOLINDEXES**

Column Name	Data Type	Description
NAME	CHAR(30) NOT NULL	DB2 column. Name of the column.
TBNAME	CHAR(18) NOT NULL	DB2 column. Name of the table that contains the column.
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table that the column is in.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of column within the table. Numbers start at 1.
COLTYPE	CHAR(8) NOT NULL	DB2 column. Type of column, values are INTEGER, SMALLINT, FLOAT, CHAR, VARCHAR, LONGVAR, DECIMAL, GRAPHIC, VARG, and LONGVARG.
LENGTH	SMALLINT NOT NULL	DB2 column. For a DECIMAL data type the columns precision. for all other data types the columns length.
SCALE	SMALLINT NOT NULL	DB2 column. Scale for a DECIMAL data type, otherwise zero.
NULLS	CHAR(1) NOT NULL	DB2 column. Y/N flag identifying whether the column can contain null values.
COLCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
HIGH2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
LOW2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
UPDATES	CHAR(1)	DB2 column. For DB2 tables, the corresponding UPDATES value from the DB2 catalog. For system tables always 'N'. For all other types of tables always 'Y'. Null and default value is 'Y'.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the column. Null and default value is spaces.
DEFAULT	CHAR(1)	DB2 column. For DB2 tables, the corresponding DEFAULT value from the DB2 catalog or for a meta data table whether the column has a NULL IS specification. For all other types of tables 'N'. Null and default value is spaces.

**Table 95: SYSCAC.SYSIDMSCOLINDEXES**

Column Name	Data Type	Description
KEYSEQy	SMALLINT	DB2 column. The columns numeric position within the primary key, Zero if the column is not part of the primary key. Null and default value is zeros.
FOREIGNKEY	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
FLDPROC	CHAR(1)	DB2 column. Flag indicating whether the column has a field procedure associated with it. 'Y' - field procedure exists, otherwise 'N'. Null and default value is 'N'.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIELD_OFFSET	SMALLINT NOT NULL	eXadas column. Relative zero offset of the start of the column in the record or segment.
FIELD_LENGTH	SMALLINT NOT NULL	eXadas column. Physical length of the field.
NATIVE_DATA_TYPE	CHAR(1) NOT NULL	eXadas column. Native type of the the underlying column. For a description of the different data types refer to the documentation on the Meta Data Utility in the System Reference Manual.
SQL_COLUMN_LENGTH	SMALLINT NOT NULL	eXadas column. The SQL length of the column.
DB2_DATA_TYPE	SMALLINT NOT NULL	eXadas column. The DB2 SQLDA data type value for the column.
PRECISION	SMALLINT NOT NULL	eXadas column. For a DECIMAL data type the columns scale. for all other data types zero.
COLUMN_SEQUENCE_NUMBER	SMALLINT NOT NULL	eXadas column. Relative zero ordinal position of this column in the table.
USE_RECORD_LENGTH	SMALLINT	eXadas column. For variable length records, whether the entire contents of the record should be mapped to the column. A value of 1 indicates that the record length should be used. Null and default value is spaces.
FRAGMENT_ID	SMALLINT NOT NULL	eXadas column. The ID of the fragment that the column is located in.
FRAGMENT_LEVEL	SMALLINT NOT NULL	eXadas column. The level number of the fragment that the column is located in.
OFFSET_IN_FRAGMENT	SMALLINT NOT NULL	eXadas column. Relative zero starting offset of the column within the fragment.

**Table 95: SYSCAC.SYSIDMSCOLINDEXES**

Column Name	Data Type	Description
NULLABLE	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating whether the column supports null values.
SIGNED	CHAR(1) NOT NULL	eXadas column. Is the underlying column's data type signed or unsigned? 'Y' the data is signed, 'N' the data is unsigned.
NULL_DATA_LENGTH	SMALLINT	eXadas column. Length of the NULL IS specification for this column Null and default value is zeros.
NULL_VALUE	CHAR(16)	eXadas column. The value for this column that means the column contains a null value. Null and default value is spaces.
FIELD_PROCEDURE_NAME	CHAR(8)	eXadas column. Name of the field procedure associated with this column. Null and default value is spaces.
OCCURS_DEPENDING_ON	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag that identifies whether the column exists within an OCCURS DEPENDING ON record array. Null and default value is 'N'.
FRAGMENT_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the starting position of the fragment within the linear buffer. Null and default value is -1.
FRAGMENT_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. Length of the fragment. If fragment level is 0 the length is the length of the record or segment. For non-zero fragment levels the length of the RECORD ARRAY. Null and default value is -1.
FRAGMENT_MAXIMUM_OCCURRENCES	SMALLINT	eXadas column - obtained from fragment definition. Maximum times the fragment can occur. Null and default value is zeros.
NULL_FRAGMENT_RULE	CHAR(30)	eXadas column - obtained from fragment definition. Method used to determine a NULL occurrence in a RECORD ARRAY. Null and default value is spaces.
OCCURS_DEPENDING_ON_COLUMN	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the name of the column that identifies the number of occurrences that exist. Null and default value is spaces.

**Table 95: SYSCAC.SYSIDMSCOLINDEXES**

Column Name	Data Type	Description
OCCURS_DEPENDING_ON_COLNO	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the COLUMN_SEQUENCE_NUMBER of the column that identifies the number of occurrences that exist. Null and default value is zeros.
FRAGMENT_NULL_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the length of the NULL IS specification for the fragment. Null and default value is zeros.
FRAGMENT_NULL_VALUE	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the value that identifies an instance as being null. Null and default value is spaces.
ELEMENT_NAME	CHAR(32) NOT NULL	eXadas column. IDMS element name that this column represents.
CALC_KEY	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating that the element is a CALCKEY.
DEPENDING_ON_COLUMN	CHAR(1) NOT NULL	eXadas column. Y/N flag that indicates whether the column occurs multiple times
RECORD_NAME	CHAR(16) NOT NULL	eXadas column. Name of the IDMS record that contains the column.
NUMBER_OF_INDEXES	SMALLINT NOT NULL	eXadas column. Number of indexes associated with this column.
INDEX_NAME	CHAR(16) NOT NULL	eXadas column. Name of the index associated with this column.
INDEX_KEY_LENGTH	SMALLINT NOT NULL	eXadas column - obtained from table definition. Length of the index key associated with this column.
INDEX_OFFSET_IN_KEY	SMALLINT NOT NULL	eXadas column. Relative zero offset of this columns position in the index key.
INDEX_SORT_ORDER	CHAR(1) NOT NULL	eXadas column - obtained from table definition. Sorting order of the index associated with this column. 'A' ascending or 'D' for descending.
INDEX_SET_TYPE	CHAR(30) NOT NULL	eXadas column - obtained from table definition. Method used to access index associated with this column. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE.

**Table 96: SYSCAC.SYSCOLUMNS**

Column Name	Data Type	Description
NAME	CHAR(30) NOT NULL	DB2 column. Name of the column.
TBNAME	CHAR(18) NOT NULL	DB2 column. Name of the table that contains the column.
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table that the column is in.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of column within the table. Numbers start at 1.
COLTYPE	CHAR(8) NOT NULL	DB2 column. Type of column, values are INTEGER, SMALLINT, FLOAT, CHAR, VARCHAR, LONGVAR, DECIMAL, GRAPHIC, VARG, and LONGVARG.
LENGTH	SMALLINT NOT NULL	DB2 column. For a DECIMAL data type the columns precision. for all other data types the columns length.
SCALE	SMALLINT NOT NULL	DB2 column. Scale for a DECIMAL data type, otherwise zero.
NULLS	CHAR(1) NOT NULL	DB2 column. Y/N flag identifying whether the column can contain null values.
COLCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
HIGH2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
LOW2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
UPDATES	CHAR(1)	DB2 column. For DB2 tables, the corresponding UPDATES value from the DB2 catalog. For system tables always 'N'. For all other types of tables always 'Y'. Null and default value is 'Y'.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the column. Null and default value is spaces.
DEFAULT	CHAR(1)	DB2 column. For DB2 tables, the corresponding DEFAULT value from the DB2 catalog or for a meta data table whether the column has a NULL IS specification. For all other types of tables 'N'. Null and default value is spaces.

**Table 96: SYSCAC.SYSIDMSCOLUMNNS**

Column Name	Data Type	Description
KEYSEQ	SMALLINT	DB2 column. The columns numeric position within the primary key, Zero if the column is not part of the primary key. Null and default value is zeros.
FOREIGNKEY	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
FLDPROC	CHAR(1)	DB2 column. Flag indicating whether the column has a field procedure associated with it. 'Y' - field procedure exists, otherwise 'N'. Null and default value is 'N'.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIELD_OFFSET	SMALLINT NOT NULL	eXadas column. Relative zero offset of the start of the column in the record or segment.
FIELD_LENGTH	SMALLINT NOT NULL	eXadas column. Physical length of the field.
NATIVE_DATA_TYPE	CHAR(1) NOT NULL	eXadas column. Native type of the the underlying column. For a description of the different data types refer to the documentation on the Meta Data Utility in the System Reference Manual.
SQL_COLUMN_LENGTH	SMALLINT NOT NULL	eXadas column. The SQL length of the column.
DB2_DATA_TYPE	SMALLINT NOT NULL	eXadas column. The DB2 SQLDA data type value for the column.
PRECISION	SMALLINT NOT NULL	eXadas column. For a DECIMAL data type the columns scale. for all other data types zero.
COLUMN_SEQUENCE_NUMBER	SMALLINT NOT NULL	eXadas column. Relative zero ordinal position of this column in the table.
USE_RECORD_LENGTH	SMALLINT	eXadas column. For variable length records, whether the entire contents of the record should be mapped to the column. A value of 1 indicates that the record length should be used. Null and default value is spaces.
FRAGMENT_ID	SMALLINT NOT NULL	eXadas column. The ID of the fragment that the column is located in.
FRAGMENT_LEVEL	SMALLINT NOT NULL	eXadas column. The level number of the fragment that the column is located in.
OFFSET_IN_FRAGMENT	SMALLINT NOT NULL	eXadas column. Relative zero starting offset of the column within the fragment.

**Table 96: SYSCAC.SYSIDMSCOLUMNS**

Column Name	Data Type	Description
NULLABLE	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating whether the column supports null values.
SIGNED	CHAR(1) NOT NULL	eXadas column. Is the underlying column's data type signed or unsigned? 'Y' the data is signed, 'N' the data is unsigned.
NULL_DATA_LENGTH	SMALLINT	eXadas column. Length of the NULL IS specification for this column Null and default value is zeros.
NULL_VALUE	CHAR(16)	eXadas column. The value for this column that means the column contains a null value. Null and default value is spaces.
FIELD_PROCEDURE_NAME	CHAR(8)	eXadas column. Name of the field procedure associated with this column. Null and default value is spaces.
OCCURS_DEPENDING_ON	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag that identifies whether the column exists within an OCCURS DEPENDING ON record array. Null and default value is 'N'.
FRAGMENT_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the starting position of the fragment within the linear buffer. Null and default value is -1.
FRAGMENT_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. Length of the fragment. If fragment level is 0 the length is the length of the record or segment. For non-zero fragment levels the length of the RECORD ARRAY. Null and default value is -1.
FRAGMENT_MAXIMUM_OCCURRENCES	SMALLINT	eXadas column - obtained from fragment definition. Maximum times the fragment can occur. Null and default value is zeros.
NULL_FRAGMENT_RULE	CHAR(30)	eXadas column - obtained from fragment definition. Method used to determine a NULL occurrence in a RECORD ARRAY. Null and default value is spaces.
OCCURS_DEPENDING_ON_COLUMN	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the name of the column that identifies the number of occurrences that exist. Null and default value is spaces.



**Table 96: SYSCAC.SYSIDMSCOLUMNS**

Column Name	Data Type	Description
OCCURS_DEPENDING_ON_COLNO	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the COLUMN_SEQUENCE_NUMBER of the column that identifies the number of occurrences that exist. Null and default value is zeros.
FRAGMENT_NULL_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the length of the NULL IS specification for the fragment. Null and default value is zeros.
FRAGMENT_NULL_VALUE	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the value that identifies an instance as being null. Null and default value is spaces.
ELEMENT_NAME	CHAR(32) NOT NULL	eXadas column. IDMS element name that this column represents.
CALC_KEY	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating that the element is a CALCKEY.
DEPENDING_ON_COLUMN	CHAR(1) NOT NULL	eXadas column. Y/N flag that indicates whether the column occurs multiple times
RECORD_NAME	CHAR(16) NOT NULL	eXadas column. Name of the IDMS record that contains the column.
NUMBER_OF_INDEXES	SMALLINT NOT NULL	eXadas column. Number of indexes associated with this column.
INDEX1_NAME	CHAR(16)	eXadas column. Name of the 1st index associated with this column. Null and default value is spaces.
INDEX1_KEY_LENGTH	SMALLINT	eXadas column - obtained from table definition. Length of the 1st index key associated with this column. Null and default value is zeros.
INDEX1_OFFSET_IN_KEY	SMALLINT	eXadas column. Relative zero offset of this columns position in the index key. Null and default value is -1.
INDEX1_SORT_ORDER	CHAR(1)	eXadas column - obtained from table definition. Sorting order of the 1st index associated with this column. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX1_SET_TYPE	CHAR(30)	eXadas column - obtained from table definition. Method used to access the 1st index associated with this column. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.

**Table 96: SYSCAC.SYSIDMSCOLUMNS**

Column Name	Data Type	Description
INDEX2_NAME	CHAR(16)	eXadas column. Name of the 2nd index associated with this column. Null and default value is spaces.
INDEX2_KEY_LENGTH	SMALLINT	eXadas column - obtained from table definition. Length of the 2nd index key associated with this column. Null and default value is zeros.
INDEX2_OFFSET_IN_KEY	SMALLINT	eXadas column. Relative zero offset of this columns position in the index key. Null and default value is -1.
INDEX2_SORT_ORDER	CHAR(1)	eXadas column - obtained from table definition. Sorting order of the 2nd index associated with this column. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX2_SET_TYPE	CHAR(30)	eXadas column - obtained from table definition. Method used to access the 2nd index associated with this column. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX3_NAME	CHAR(16)	eXadas column. Name of the 3rd index associated with this column. Null and default value is spaces.
INDEX3_KEY_LENGTH	SMALLINT	eXadas column - obtained from table definition. Length of the 3rd index key associated with this column. Null and default value is zeros.
INDEX3_OFFSET_IN_KEY	SMALLINT	eXadas column. Relative zero offset of this columns position in the index key. Null and default value is -1.
INDEX3_SORT_ORDER	CHAR(1)	eXadas column - obtained from table definition. Sorting order of the 3rd index associated with this column. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX3_SET_TYPE	CHAR(30)	eXadas column - obtained from table definition. Method used to access the 3rd index associated with this column. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX4_NAME	CHAR(16)	eXadas column. Name of the 4th index associated with this column. Null and default value is spaces.
INDEX4_KEY_LENGTH	SMALLINT	eXadas column - obtained from table definition. Length of the 4th index key associated with this column. Null and default value is zeros.
INDEX4_OFFSET_IN_KEY	SMALLINT	eXadas column. Relative zero offset of this columns position in the index key. Null and default value is -1.

**Table 96: SYSCAC.SYSIDMSCOLUMNS**

Column Name	Data Type	Description
INDEX4_SORT_ORDER	CHAR(1)	eXadas column - obtained from table definition. Sorting order of the 4th index associated with this column. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX4_SET_TYPE	CHAR(30)	eXadas column - obtained from table definition. Method used to access the 4th index associated with this column. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX5_NAME	CHAR(16)	eXadas column. Name of the 5th index associated with this column. Null and default value is spaces.
INDEX5_KEY_LENGTH	SMALLINT	eXadas column - obtained from table definition. Length of the 5th index key associated with this column. Null and default value is zeros.
INDEX5_OFFSET_IN_KEY	SMALLINT	eXadas column. Relative zero offset of this columns position in the index key. Null and default value is -1.
INDEX5_SORT_ORDER	CHAR(1)	eXadas column - obtained from table definition. Sorting order of the 5th index associated with this column. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX5_SET_TYPE	CHAR(30)	eXadas column - obtained from table definition. Method used to access the 5th index associated with this column. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX6_NAME	CHAR(16)	eXadas column. Name of the 6th index associated with this column. Null and default value is spaces.
INDEX6_KEY_LENGTH	SMALLINT	eXadas column - obtained from table definition. Length of the 6th index key associated with this column. Null and default value is zeros.
INDEX6_OFFSET_IN_KEY	SMALLINT	eXadas column. Relative zero offset of this columns position in the index key. Null and default value is -1.
INDEX6_SORT_ORDER	CHAR(1)	eXadas column - obtained from table definition. Sorting order of the 6th index associated with this column. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX6_SET_TYPE	CHAR(30)	eXadas column - obtained from table definition. Method used to access the 6th index associated with this column. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.

**Table 96: SYSCAC.SYSIDMSCOLUMNS**

Column Name	Data Type	Description
INDEX7_NAME	CHAR(16)	eXadas column. Name of the 7th index associated with this column. Null and default value is spaces.
INDEX7_KEY_LENGTH	SMALLINT	eXadas column - obtained from table definition. Length of the 7th index key associated with this column. Null and default value is zeros.
INDEX7_OFFSET_IN_KEY	SMALLINT	eXadas column. Relative zero offset of this columns position in the index key. Null and default value is -1.
INDEX7_SORT_ORDER	CHAR(1)	eXadas column - obtained from table definition. Sorting order of the 7th index associated with this column. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX7_SET_TYPE	CHAR(30)	eXadas column - obtained from table definition. Method used to access the 7th index associated with this column. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX8_NAME	CHAR(16)	eXadas column. Name of the 8th index associated with this column. Null and default value is spaces.
INDEX8_KEY_LENGTH	SMALLINT	eXadas column - obtained from table definition. Length of the 8th index key associated with this columns. Null and default value is zeros.
INDEX8_OFFSET_IN_KEY	SMALLINT	eXadas column. Relative zero offset of this columns position in the index key. Null and default value is -1.
INDEX8_SORT_ORDER	CHAR(1)	eXadas column - obtained from table definition. Sorting order of the 8th index associated with this column. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX8_SET_TYPE	CHAR(30)	eXadas column - obtained from table definition. Method used to access the 8th index associated with this column. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX9_NAME	CHAR(16)	eXadas column. Name of the 9th index associated with this column. Null and default value is spaces.
INDEX9_KEY_LENGTH	SMALLINT	eXadas column - obtained from table defintion. Length of the 9th index key associated with this column. Null and default value is zeros.
INDEX9_OFFSET_IN_KEY	SMALLINT	eXadas column. Relative zero offset of this columns position in the index key. Null and default value is -1.

**Table 96: SYSCAC.SYSIDMSCOLUMNS**

Column Name	Data Type	Description
INDEX9_SORT_ORDER	CHAR(1)	eXadas column - obtained from table definition. Sorting order of the 9th index associated with this column. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX9_SET_TYPE	CHAR(30)	eXadas column - obtained from table definition. Method used to access the 9th index associated with this column. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX10_NAME	CHAR(16)	eXadas column. Name of the 10th index associated with this column. Null and default value is spaces.
INDEX10_KEY_LENGTH	SMALLINT	eXadas column - obtained from table definition. Length of the 10th index key associated with this column. Null and default value is zeros.
INDEX10_OFFSET_IN_KEY	SMALLINT	eXadas column. Relative zero offset of this columns position in the index key. Null and default value is -1.
INDEX10_SORT_ORDER	CHAR(1)	eXadas column - obtained from table definition. Sorting order of the 10th index associated with this column. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX10_SET_TYPE	CHAR(30)	eXadas column - obtained from table definition. Method used to access the 10th index associated with this column. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.

**Table 97: SYSCAC.SYSIDMSINDEXES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of index.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the index.
TBNAME	CHAR(18) NOT NULL	DB2 column. Name of the table that the index is referencing.
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
UNIQUERULE	CHAR(1) NOT NULL	DB2 column. Whether the index is unique. 'U' yes or 'D' no.

**Table 97: SYSCAC.SYSIDMSINDEXES**

Column Name	Data Type	Description
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns that make up the key.
CLUSTERING	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
CLUSTERED	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
DBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
ISOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
DBNAME	CHAR(8) NOT NULL	DB2 column. Not used by eXadas.
INDEXSPACE	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIRSTKEYCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
FULLKEYCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
NLEAF	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
NLEVELS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
BPOOL	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
PGSIZE	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
ERASERULE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
DSETPASS	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLOSERULE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'Y'.

**Table 97: SYSCAC.SYSIDMSINDEXES**

Column Name	Data Type	Description
SPACE	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
CLUSTERRATIO	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the index.
SORT_ORDER	CHAR(1) NOT NULL	eXadas column. Flag indicating the sorting order of the IDMS index that will be used. 'A' ascending or 'D' for descending.

**Table 98: SYSCAC.SYSIDMSKEYS**

Column Name	Data Type	Description
IXNAME	CHAR(18) NOT NULL	DB2 column. Name of the index.
IXCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the index.
COLNAME	CHAR(30) NOT NULL	DB2 column. Name of the column for this key.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of the column in the table.
COLSEQ	SMALLINT NOT NULL	DB2 column. Position of this column within the key.
ORDERING	CHAR(1) NOT NULL	DB2 column. Sort order of this column within the key. 'A' ascending or 'D' descending.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.

**Table 99: SYSCAC.SYSIDMSRECORDS**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of the table.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
TYPE	CHAR(1) NOT NULL	DB2 column. Type of object. 'T' for a table or 'V' for a view.
DBNAME	CHAR(8) NOT NULL	DB2 column. Identifies the type of DBMS that the table is referencing. Values are \$ADABAS, \$CFI (System Table), \$DATACOM, \$DB2, \$IAM, \$IMS, \$IDMS, \$MDS (Meta Data Table) \$SEQUENT and \$VSAM.
TSNAME	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
DBID	SMALLINT	DB2 column. Internal identifier of the database. Zero if the the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Internal identifier of the table. Zero if the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns defined for this table.
EDPROC	CHAR(8)	DB2 column. Specifies the name of an IDMS compression/decompression routine Null and default value is spaces.
VALPROC	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERTYPE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERRID	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
CARD	INTEGER	DB2 column. Number of rows in the table. Null and default value is -1.
NPAGES	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.



**Table 99: SYSCAC.SYSIDMSRECORDS**

Column Name	Data Type	Description
PCTPAGES	SMALLINT	DB2 column. Not used by eXadas. Null and default value is -1.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the table. Null and default value is spaces.
PARENTS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CHILDREN	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
KEYCOLUMNS	SMALLINT	DB2 column. Number of columns that make up the tables primary key. Only maintained for system tables. For all other tables 0. Null and default value is zeros.
RECLENGTH	SMALLINT NOT NULL	DB2 column. Total length of all records associated with this table.
STATUS	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
KEYOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKFLAG	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKRID	CHAR(4)	DB2 column. Not used by eXadas. Null and default value is spaces.
AUDIT	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the table.
LOCATION	CHAR(16)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBCREATOR	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBNAME	CHAR(18)	DB2 column. Not used by eXadas. Null and default value is spaces.

**Table 99: SYSCAC.SYSIDMSRECORDS**

Column Name	Data Type	Description
FRAGMENTS	SMALLINT NOT NULL	eXadas column. Number of fragments used to manage this table.
DBMS	CHAR(8) NOT NULL	eXadas column. Identifies the type of database. Always 'IDMS'.
DATA_CAPTURE	CHAR(1)	DB2 column. Identifies whether the table has been marked for xSync data capture. A value of 'Y' indicates that changes will be captured. Null and default value is spaces.
SORT_KEYS	SMALLINT	eXadas column. Total number of sort keys for all indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
INDEXES	SMALLINT	eXadas column. Total number of indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
DATABASE_NAME	CHAR(8)	eXadas column. Name of IDMS database used to map this table. Null and default value is spaces.
SCHEMA_NAME	CHAR(8) NOT NULL	eXadas column. IDMS Schema Name used to map this table.
SUB_SCHEMA_NAME	CHAR(8) NOT NULL	eXadas column. IDMS sub-schema name used to map this table.
SUB_SCHEMA_VERSION	SMALLINT NOT NULL	eXadas column. The IDMS sub-schema version number used to map this table.
RECORD_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS records referenced by this table. Maximum of 10 records supported.
SET_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS sets referenced by this table. Maximum of 9 sets supported.
INDEX_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS indexes referenced by this table. Maximum of 10 indexes supported.
AREA_COUNT	SMALLINT NOT NULL	eXadas column. Total number of areas associated with this table. Maximum of 10 areas supported.
ACCESS_MODULE	CHAR(8) NOT NULL	eXadas column. Name of IDMS load module used to access this table.
ACCESS_METHOD	CHAR(4)	eXadas column. Method used to access the table. Values are IDMS, KSDS, RRDS or ESDS. Null and default value is spaces.

**Table 99: SYSCAC.SYSIDMSRECORDS**

Column Name	Data Type	Description
VARIABLE_LENGTH_TABLE	CHAR(1) NOT NULL	eXadas column. Flag indicating whether table is variable length. 'Y' if variable length or 'N' for fixed length tables.
MINIMUM_LENGTH	SMALLINT	eXadas column. Minimum length for variable length tables. Null and default value is zeros.
PARENT_DB_KEY_OFFSET	SMALLINT	eXadas column. For tables marked for data capture, the offset of the parent database key in the IMDS journal. Null and default value is -1.
PAGE_AREA_MIN	INTEGER	eXadas column. For tables marked for data capture, the low page area dbkey value. Null and default value is zeros.
PAGE_AREA_MAX	INTEGER	eXadas column. For tables marked for data capture, the high page area dbkey value. Null and default value is zeros.
RECORD_NAME	CHAR(16) NOT NULL	eXadas column. Name of the record referenced by this table.
RECORD_LENGTH	SMALLINT NOT NULL	eXadas column. Length of the record referenced by this table.
RECORD_FRAGMENT_ID	SMALLINT NOT NULL	eXadas column - obtained from fragment definition. Fragment ID associated with this record.
RECORD_OFFSET_IN_BUFFER	SMALLINT NOT NULL	eXadas column - obtained from fragment definition. Relative zero offset of the start of the record in the linear buffer
RECORD_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the record does not represent an actual record definition and is instead a VSAM relative record number. Null and default value is spaces.
RECORD_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag whether the record is variable length. Null and default value is spaces.
RECORD_DEPENDENT_ON_COLNO	SMALLINT	eXadas column. Number of the controlling column if the record contains a DEPENDING ON column definition, otherwise -1. Null and default value is -1.
RECORD_DEPENDENT_ON_MAX	SMALLINT	eXadas column. Maximum number of occurrences if the record contains a DEPENDING ON column definition. Valid when RECORD_DEPENDENT_ON_COLNO is not null. Null and default value is -1.

**Table 99: SYSCAC.SYSIDMSRECORDS**

Column Name	Data Type	Description
RECORD_DEPENDING_ON_INC	SMALLINT	eXadas column. Buffer increment if the record contains a DEPENDING ON column definition. Valid when RECORD_DEPENDING_ON_COLNO is not null. Null and default value is -1.
RECORD_TARGET_IS_OWNER	CHAR(16)	eXadas column. Y/N flag indicating whether the target for the record is the owner. Null and default value is spaces.
RECORD_AREA_NAME	CHAR(1)	eXadas column. Area name associated with this record. Null and default value is spaces.

**Table 100: SYSCAC.SYSIDMSSETS**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of the table.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
TYPE	CHAR(1) NOT NULL	DB2 column. Type of object. 'T' for a table or 'V' for a view.
DBNAME	CHAR(8) NOT NULL	DB2 column. Identifies the type of DBMS that the table is referencing. Values are \$ADABAS, \$CFI (System Table), \$DATACOM, \$DB2, \$IAM, \$IMS, \$IDMS, \$MDS (Meta Data Table) \$SEQUENT and \$VSAM.
TSNAME	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
DBID	SMALLINT	DB2 column. Internal identifier of the database. Zero if the the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Internal identifier of the table. Zero if the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
COLCOUNT	COLCOUNT	DB2 column. Number of columns defined for this table.
EDPROC	CHAR(8)	DB2 column. Specifies the name of an IDMS compression/decompression routine Null and default value is spaces.

**Table 100: SYSCAC.SYSIDMSSETS**

Column Name	Data Type	Description
VALPROC	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERTYPE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERRID	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
CARD	INTEGER	DB2 column. Number of rows in the table. Null and default value is -1.
NPAGES	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
PCTPAGES	SMALLINT	DB2 column. Not used by eXadas. Null and default value is -1.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the table. Null and default value is spaces.
PARENTS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CHILDREN	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
KEYCOLUMNS	SMALLINT	DB2 column. Number of columns that make up the tables primary key. Only maintained for system tables. For all other tables 0. Null and default value is zeros.
RECLENGTH	SMALLINT NOT NULL	DB2 column. Total length of all records associated with this table.
STATUS	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
KEYOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKFLAG	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKRID	CHAR(4)	DB2 column. Not used by eXadas. Null and default value is spaces.

**Table 100: SYSCAC.SYSSIDMSSETS**

Column Name	Data Type	Description
AUDIT	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the table.
LOCATION	CHAR(16)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBCREATOR	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBNAME	CHAR(18)	DB2 column. Not used by eXadas. Null and default value is spaces.
FRAGMENTS	SMALLINT NOT NULL	eXadas column. Number of fragments used to manage this table.
DBMS	CHAR(8) NOT NULL	eXadas column. Identifies the type of database. Always 'IDMS'.
DATA_CAPTURE	CHAR(1)	DB2 column. Identifies whether the table has been marked for xSync data capture. A value of 'Y' indicates that changes will be captured. Null and default value is spaces.
SORT_KEYS	SMALLINT	eXadas column. Total number of sort keys for all indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
INDEXES	SMALLINT	eXadas column. Total number of indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
DATABASE_NAME	CHAR(8)	eXadas column. Name of IDMS database used to map this table. Null and default value is spaces.
SCHEMA_NAME	CHAR(8) NOT NULL	eXadas column. IDMS Schema Name used to map this table.
SUB_SCHEMA_NAME	CHAR(8) NOT NULL	eXadas column. IDMS sub-schema name used to map this table.
SUB_SCHEMA_VERSION	SMALLINT NOT NULL	eXadas column. The IDMS sub-schema version number used to map this table.
RECORD_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS records referenced by this table. Maximum of 10 records supported.

**Table 100: SYSCAC.SYSIDMSSETS**

Column Name	Data Type	Description
SET_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS sets referenced by this table. Maximum of 9 sets supported.
INDEX_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS indexes referenced by this table. Maximum of 10 indexes supported.
AREA_COUNT	SMALLINT NOT NULL	eXadas column. Total number of areas associated with this table. Maximum of 10 areas supported.
ACCESS_MODULE	CHAR(8) NOT NULL	eXadas column. Name of IDMS load module used to access this table.
ACCESS_METHOD	CHAR(4)	eXadas column. Method used to access the table. Values are IDMS, KSDS, RRDS or ESDS. Null and default value is spaces.
VARIABLE_LENGTH_TABLE	CHAR(1) NOT NULL	eXadas column. Flag indicating whether table is variable length. 'Y' if variable length or 'N' for fixed length tables.
MINIMUM_LENGTH	SMALLINT	eXadas column. Minimum length for variable length tables. Null and default value is zeros.
PARENT_DB_KEY_OFFSET	SMALLINT	eXadas column. For tables marked for data capture, the offset of the parent database key in the IMDS journal. Null and default value is -1.
PAGE_AREA_MIN	INTEGER	eXadas column. For tables marked for data capture, the low page area dbkey value. Null and default value is zeros.
PAGE_AREA_MAX	INTEGER	eXadas column. For tables marked for data capture, the high page area dbkey value. Null and default value is zeros.
SET_NAME	CHAR(16) NOT NULL	eXadas column. Name of the set associated with this table.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of the table.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
TYPE	CHAR(1) NOT NULL	DB2 column. Type of object. 'T' for a table or 'V' for a view.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
DBNAME	CHAR(8) NOT NULL	DB2 column. Identifies the type of DBMS that the table is referencing. Values are \$ADABAS, \$CFI (System Table), \$DATACOM, \$DB2, \$IAM, \$IMS, \$IDMS, \$MDS (Meta Data Table) \$SEQUENT and \$VSAM.
TSNAME	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
DBID	SMALLINT	DB2 column. Internal identifier of the database. Zero if the the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Internal identifier of the table. Zero if the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns defined for this table.
EDPROC	CHAR(8)	DB2 column. Specifies the name of an IDMS compression/decompression routine Null and default value is spaces.
VALPROC	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERTYPE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERRID	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
CARD	INTEGER	DB2 column. Number of rows in the table. Null and default value is -1.
NPAGES	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
PCTPAGES	SMALLINT	DB2 column. Not used by eXadas. Null and default value is -1.
IBMREQDY	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the table. Null and default value is spaces.
PARENTS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.



**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
CHILDREN	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
KEYCOLUMNS	SMALLINT	DB2 column. Number of columns that make up the tables primary key. Only maintained for system tables. For all other tables 0. Null and default value is zeros.
RECLENGTH	SMALLINT NOT NULL	DB2 column. Total length of all records associated with this table.
STATUS	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
KEYOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKFLAG	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKRID	CHAR(4)	DB2 column. Not used by eXadas. Null and default value is spaces.
AUDIT	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the table.
LOCATION	CHAR(16)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBCREATOR	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBNAME	CHAR(18)	DB2 column. Not used by eXadas. Null and default value is spaces.
FRAGMENTS	SMALLINT NOT NULL	eXadas column. Number of fragments used to manage this table.
DBMS	CHAR(8) NOT NULL	eXadas column. Identifies the type of database. Always 'IDMS'.
DATA_CAPTURE	CHAR(1)	DB2 column. Identifies whether the table has been marked for xSync data capture. A value of 'Y' indicates that changes will be captured. Null and default value is spaces.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
SORT_KEYS	SMALLINT	eXadas column. Total number of sort keys for all indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
INDEXES	SMALLINT	eXadas column. Total number of indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
DATABASE_NAME	CHAR(8)	eXadas column. Name of IDMS database used to map this table. Null and default value is spaces.
SCHEMA_NAME	CHAR(8) NOT NULL	eXadas column. IDMS Schema Name used to map this table.
SUB_SCHEMA_NAME	CHAR(8) NOT NULL	eXadas column. IDMS sub-schema name used to map this table.
SUB_SCHEMA_VERSION	SMALLINT NOT NULL	eXadas column. The IDMS sub-schema version number used to map this table.
RECORD_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS records referenced by this table. Maximum of 10 records supported.
SET_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS sets referenced by this table. Maximum of 9 sets supported.
INDEX_COUNT	SMALLINT NOT NULL	eXadas column. Number of IDMS indexes referenced by this table. Maximum of 10 indexes supported.
AREA_COUNT	SMALLINT NOT NULL	eXadas column. Total number of areas associated with this table. Maximum of 10 areas supported.
ACCESS_MODULE	CHAR(8) NOT NULL	eXadas column. Name of IDMS load module used to access this table.
ACCESS_METHOD	CHAR(4)	eXadas column. Method used to access the table. Values are IDMS, KSDS, RRDS or ESDS. Null and default value is spaces.
VARIABLE_LENGTH_TABLE	CHAR(1) NOT NULL	eXadas column. Flag indicating whether table is variable length. 'Y' if variable length or 'N' for fixed length tables.
MINIMUM_LENGTH	SMALLINT	eXadas column. Minimum length for variable length tables. Null and default value is zeros.
PARENT_DB_KEY_OFFSET	SMALLINT	eXadas column. For tables marked for data capture, the offset of the parent database key in the IDMS journal. Null and default value is -1.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
PAGE_AREA_MIN	INTEGER	eXadas column. For tables marked for data capture, the low page area dbkey value. Null and default value is zeros.
PAGE_AREA_MAX	INTEGER	eXadas column. For tables marked for data capture, the high page area dbkey value. Null and default value is zeros.
RECORD1_NAME	CHAR(16) NOT NULL	eXadas column. Name of the 1st record referenced by this table.
RECORD1_LENGTH	SMALLINT NOT NULL	eXadas column. Length of the 1st record referenced by this table.
RECORD1_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 1st record. Null and default value is zeros.
RECORD1_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 1st record in the linear buffer Null and default value is -1.
RECORD1_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 1st record does not represent an actual record definition and is instead a VSAM relative record number. Null and default value is spaces.
RECORD1_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 1st record is variable length. Null and default value is spaces.
RECORD1_DEPENDIN G_ON_COLNO	SMALLINT	eXadas column. Number of the controlling column if the 1st record contains a DEPENDING ON column definition, otherwise -1. Null and default value is -1.
RECORD1_DEPENDIN G_ON_MAX	SMALLINT	eXadas column. Maximum number of occurrences if the 1st record contains a DEPENDING ON column definition. Valid when RECORD1_DEPENDIN G_ON_COLNO is not null. Null and default value is -1.
RECORD1_DEPENDIN G_ON_INC	SMALLINT	eXadas column. Buffer increment if the 1st record contains a DEPENDING ON column definition. Valid when RECORD1_DEPENDIN G_ON_COLNO is not null. Null and default value is -1.
RECORD1_TARGET_IS _OWNER	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating whether the target for the 1st record is the owner.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
RECORD1_AREA_NAME	CHAR(16) NOT NULL	eXadas column. Area name associated with the 1st record.
RECORD2_NAME	CHAR(16)	eXadas column. Name of the 2nd record referenced by this table. Null and default value is spaces.
RECORD2_LENGTH	SMALLINT	eXadas column. Length of the 2nd record referenced by this table. Null and default value is zeros.
RECORD2_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 2nd record. Null and default value is zeros.
RECORD2_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 2nd record in the linear buffer Null and default value is -1.
RECORD2_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 2nd record does not represent an actual record definition and is instead a VSAM relative record number. Null and default value is spaces.
RECORD2_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 2nd record is variable length. Null and default value is spaces.
RECORD2_DEPENDING_ON_COLNO	SMALLINT	eXadas column. Number of the controlling column if the 2nd record contains a DEPENDING ON column definition, otherwise -1. Null and default value is -1.
RECORD2_DEPENDING_ON_MAX	SMALLINT	eXadas column. Maximum number of occurrences if the 2nd record contains a DEPENDING ON column definition. Valid when RECORD2_DEPENDING_ON_COLNO is not null. Null and default value is -1.
RECORD2_DEPENDING_ON_INC	SMALLINT	eXadas column. Buffer increment if the 2nd record contains a DEPENDING ON column definition. Valid when RECORD2_DEPENDING_ON_COLNO is not null. Null and default value is -1.
RECORD2_TARGET_IS_OWNER	CHAR(1)	eXadas column. Y/N flag indicating whether the target for the 2nd record is the owner. Null and default value is spaces.
RECORD2_AREA_NAME	CHAR(16)	eXadas column. Area name associated with the 2nd record. Null and default value is spaces.
RECORD3_NAME	CHAR(16)	eXadas column. Name of the 3rd record referenced by this table. Null and default value is spaces.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
RECORD3_LENGTH	SMALLINT	eXadas column. Length of the 3rd record referenced by this table. Null and default value is zeros.
RECORD3_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 3rd record. Null and default value is zeros.
RECORD3_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 3rd record in the linear buffer Null and default value is -1.
RECORD3_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 3rd record does not represent an actual record definition and is instead a VSAM relative record number. Null and default value is spaces.
RECORD3_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 3rd record is variable length. Null and default value is spaces.
RECORD3_DEPENDIN G_ON_COLNO	SMALLINT	eXadas column. Number of the controlling column if the 3rd record contains a DEPENDING ON column definition, otherwise -1. Null and default value is -1.
RECORD3_DEPENDIN G_ON_MAX	SMALLINT	eXadas column. Maximum number of occurrences if the 3rd record contains a DEPENDING ON column definition. Valid when RECORD3_DEPENDIN G_ON_COLNO is not null. Null and default value is -1.
RECORD3_DEPENDIN G_ON_INC	SMALLINT	eXadas column. Buffer increment if the 3rd record contains a DEPENDING ON column definition. Valid when RECORD3_DEPENDIN G_ON_COLNO is not null. Null and default value is -1.
RECORD3_TARGET_IS _OWNER	CHAR(1)	eXadas column. Y/N flag indicating whether the target for the 3rd record is the owner. Null and default value is spaces.
RECORD3_AREA_NAME	CHAR(16)	eXadas column. Area name associated with the 3rd record. Null and default value is spaces.
RECORD4_NAME	CHAR(16)	eXadas column. Name of the 4th record referenced by this table. Null and default value is spaces.
RECORD4_LENGTH	SMALLINT	eXadas column. Length of the 4th record referenced by this table. Null and default value is zeros.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
RECORD4_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 4th record. Null and default value is zeros.
RECORD4_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 4th record in the linear buffer Null and default value is -1.
RECORD4_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 4th record does not represent an actual record definition and is instead a VSAM relative record number. Null and default value is spaces.
RECORD4_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 4th record is variable length. Null and default value is spaces.
RECORD4_DEPENDING_ON_COLNO	SMALLINT	eXadas column. Number of the controlling column if the 4th record contains a DEPENDING ON column definition, otherwise -1. Null and default value is -1.
RECORD4_DEPENDING_ON_MAX	SMALLINT	eXadas column. Maximum number of occurrences if the 4th record contains a DEPENDING ON column definition. Valid when RECORD4_DEPENDING_ON_COLNO is not null. Null and default value is -1.
RECORD4_DEPENDING_ON_INC	SMALLINT	eXadas column. Buffer increment if the 4th record contains a DEPENDING ON column definition. Valid when RECORD4_DEPENDING_ON_COLNO is not null. Null and default value is -1.
RECORD4_TARGET_IS_OWNER	CHAR(1)	eXadas column. Y/N flag indicating whether the target for the 4th record is the owner. Null and default value is spaces.
RECORD4_AREA_NAME	CHAR(16)	eXadas column. Area name associated with the 4th record. Null and default value is spaces.
RECORD5_NAME	CHAR(16)	eXadas column. Name of the 5th record referenced by this table. Null and default value is spaces.
RECORD5_LENGTH	SMALLINT	eXadas column. Length of the 5th record referenced by this table. Null and default value is zeros.
RECORD5_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 5th record. Null and default value is zeros.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
RECORD5_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 5th record in the linear buffer Null and default value is -1.
RECORD5_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 5th record does not represent an actual record definition and is instead a VSAM relative record number. Null and default value is spaces.
RECORD5_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 5th record is variable length. Null and default value is spaces.
RECORD5_DEPENDING_ON_COLNO	SMALLINT	eXadas column. Number of the controlling column if the 5th record contains a DEPENDING ON column definition, otherwise -1. Null and default value is -1.
RECORD5_DEPENDING_ON_MAX	SMALLINT	eXadas column. Maximum number of occurrences if the 5th record contains a DEPENDING ON column definition. Valid when RECORD5_DEPENDING_ON_COLNO is not null. Null and default value is -1.
RECORD5_DEPENDING_ON_INC	SMALLINT	eXadas column. Buffer increment if the 5th record contains a DEPENDING ON column definition. Valid when RECORD5_DEPENDING_ON_COLNO is not null. Null and default value is -1.
RECORD5_TARGET_IS_OWNER	CHAR(1)	eXadas column. Y/N flag indicating whether the target for the 5th record is the owner. Null and default value is spaces.
RECORD5_AREA_NAME	CHAR(16)	eXadas column. Area name associated with the 5th record. Null and default value is spaces.Y
RECORD6_NAME	CHAR(16)	eXadas column. Name of the 6th record referenced by this table. Null and default value is spaces.
RECORD6_LENGTH	SMALLINT	eXadas column. Length of the 6th record referenced by this table. Null and default value is zeros.
RECORD6_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 6th record. Null and default value is zeros.
RECORD6_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 6th record in the linear buffer Null and default value is -1.

**Table 101: SYSCAC.SYSDMSTABLES**

Column Name	Data Type	Description
RECORD6_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 6th record does not represent an actual record definition and is instead a VSAM relative record number. Null and default value is spaces.
RECORD6_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 6th record is variable length. Null and default value is spaces.
RECORD6_DEPENDING_ON_COLNO	SMALLINT	SMALLINT
RECORD6_DEPENDING_ON_MAX	SMALLINT	eXadas column. Maximum number of occurrences if the 6th record contains a DEPENDING ON column definition. Valid when RECORD6_DEPENDING_ON_COLNO is not null. Null and default value is -1.
RECORD6_DEPENDING_ON_INC	SMALLINT	eXadas column. Buffer increment if the 6th record contains a DEPENDING ON column definition. Valid when RECORD6_DEPENDING_ON_COLNO is not null. Null and default value is -1.
RECORD6_TARGET_IS_OWNER	CHAR(1)	eXadas column. Y/N flag indicating whether the target for the 6th record is the owner. Null and default value is spaces.
RECORD6_AREA_NAME	CHAR(16)	eXadas column. Area name associated with the 6th record. Null and default value is spaces.
RECORD7_NAME	CHAR(16)	eXadas column. Name of the 7th record referenced by this table. Null and default value is spaces.
RECORD7_LENGTH	SMALLINT	eXadas column. Length of the 7th record referenced by this table. Null and default value is zeros.
RECORD7_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 7th record. Null and default value is zeros.
RECORD7_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 7th record in the linear buffer Null and default value is -1.
RECORD7_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 7th record does not represent an actual record definition and is instead a VSAM relative record number. Null and default value is spaces.



**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
RECORD7_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 7th record is variable length. Null and default value is spaces.
RECORD7_DEPENDIN G_ON_COLNO	SMALLINT	eXadas column. Number of the controlling column if the 7th record contains a DEPENDING ON column definition, otherwise -1. Null and default value is -1.
RECORD7_DEPENDIN G_ON_MAX	SMALLINT	eXadas column. Maximum number of occurrences if the 7th record contains a DEPENDING ON column definition. Valid when RECORD7_DEPENDIN G_ON_COLNO is not null. Null and default value is -1.
RECORD7_DEPENDIN G_ON_INC	SMALLINT	eXadas column. Buffer increment if the 7th record contains a DEPENDING ON column definition. Valid when RECORD7_DEPENDIN G_ON_COLNO is not null. Null and default value is -1.
RECORD7_TARGET_IS _OWNER	CHAR(1)	eXadas column. Y/N flag indicating whether the target for the 7th record is the owner. Null and default value is spaces.
RECORD7_AREA_NAME	CHAR(16)	eXadas column. Area name associated with the 7th record. Null and default value is spaces.
RECORD8_NAME	CHAR(16)	eXadas column. Name of the 8th record referenced by this table. Null and default value is spaces.
RECORD8_LENGTH	SMALLINT	eXadas column. Length of the 8th record referenced by this table. Null and default value is zeros.
RECORD8_FRAGMENT _ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 8th record. Null and default value is zeros.
RECORD8_OFFSET_IN _BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 8th record in the linear buffer Null and default value is -1.
RECORD8_SYSTEM_K EY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 8th record does not represent an actual record definition and is instead a VSAM relative record number. Null and default value is spaces.
RECORD8_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 8th record is variable length. Null and default value is spaces.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
RECORD8_DEPENDIN G_ON_COLNO	SMALLINT	eXadas column. Number of the controlling column if the 8th record contains a DEPENDING ON column definition, otherwise -1. Null and default value is -1.
RECORD8_DEPENDIN G_ON_MAX	SMALLINT	eXadas column. Maximum number of occurrences if the 8th record contains a DEPENDING ON column definition. Valid when RECORD8_DEPENDIN_G_ON_COLNO is not null. Null and default value is -1.
RECORD8_DEPENDIN G_ON_INC	SMALLINT	eXadas column. Buffer increment if the 8th record contains a DEPENDING ON column definition. Valid when RECORD8_DEPENDIN_G_ON_COLNO is not null. Null and default value is -1.
RECORD8_TARGET_IS _OWNER	CHAR(1)	eXadas column. Y/N flag indicating whether the target for the 8th record is the owner. Null and default value is spaces.
RECORD8_AREA_NAM E	CHAR(16)	eXadas column. Area name associated with the 8th record. Null and default value is spaces.
RECORD9_NAME	CHAR(16)	eXadas column. Name of the 9th record referenced by this table. Null and default value is spaces.
RECORD9_LENGTH	SMALLINT	eXadas column. Length of the 9th record referenced by this table. Null and default value is zeros.
RECORD9_FRAGMENT _ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 9th record. Null and default value is zeros.
RECORD9_OFFSET_IN _BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 9th record in the linear buffer Null and default value is -1.
RECORD9_SYSTEM_K EY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 9th record does not represent an actual record definition and is instead a VSAM relative record number. Null and default value is spaces.
RECORD9_VARIABLE_ LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 9th record is variable length. Null and default value is spaces.
RECORD9_DEPENDIN G_ON_COLNO	SMALLINT	eXadas column. Number of the controlling column if the 9th record contains a DEPENDING ON column definition, otherwise -1. Null and default value is -1.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
RECORD9_DEPENDIN G_ON_MAX	SMALLINT	eXadas column. Maximum number of occurrences if the 9th record contains a DEPENDING ON column definition. Valid when RECORD9_DEPENDING_ON_COLNO is not null. Null and default value is -1.
RECORD9_DEPENDIN G_ON_INC	SMALLINT	eXadas column. Buffer increment if the 9th record contains a DEPENDING ON column definition. Valid when RECORD9_DEPENDING_ON_COLNO is not null. Null and default value is -1.
RECORD9_TARGET_IS _OWNER	CHAR(1)	eXadas column. Y/N flag indicating whether the target for the 9th record is the owner. Null and default value is spaces.
RECORD9_AREA_NAM E	CHAR(16)	eXadas column. Area name associated with the 9th record. Null and default value is spaces.
RECORD10_NAME	CHAR(16)	eXadas column. Name of the 10th record referenced by this table. Null and default value is spaces.
RECORD10_LENGTH	SMALLINT	eXadas column. Length of the 10th record referenced by this table. Null and default value is zeros.
RECORD10_FRAGMEN T_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 10th record. Null and default value is zeros.Y
RECORD10_OFFSET_I N_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 10th record in the linear buffer Null and default value is -1.
RECORD10_SYSTEM_ KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 10th record does not represent an actual record definition and is instead a VSAM relative record number. Null and default value is spaces.
RECORD10_VARIABLE _LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 10th record is variable length. Null and default value is spaces.
RECORD10_DEPENDIN G_ON_COLNO	SMALLINT	eXadas column. Number of the controlling column if the 10th record contains a DEPENDING ON column definition, otherwise -1. Null and default value is -1.
RECORD10_DEPENDIN G_ON_MAX	SMALLINT	eXadas column. Maximum number of occurrences if the 10th record contains a DEPENDING ON column definition. Valid when RECORD10_DEPENDING_ON_COLNO is not null. Null and default value is -1.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
RECORD10_DEPENDIN G_ON_INC	SMALLINT	eXadas column. Buffer increment if the 10th record contains a DEPENDING ON column definition. Valid when RECORD10_DEPENDING_ON_COLNO is not null. Null and default value is -1.
RECORD10_TARGET_I S_OWNER	CHAR(1)	eXadas column. Y/N flag indicating whether the target for the 10th record is the owner. Null and default value is spaces.
RECORD10_AREA_NA ME	CHAR(16)	eXadas column. Area name associated with the 10th record. Null and default value is spaces.
SET1_NAME	CHAR(16)	eXadas column. Name of the 1st set associated with this table. Null and default value is spaces.
SET2_NAME	CHAR(16)	eXadas column. Name of the 2nd set associated with this table. Null and default value is spaces.
SET3_NAME	CHAR(16)	eXadas column. Name of the 3rd set associated with this table. Null and default value is spaces.
SET4_NAME	CHAR(16)	eXadas column. Name of the 4th set associated with this table. Null and default value is spaces.
SET5_NAME	CHAR(16)	eXadas column. Name of the 5th set associated with this table. Null and default value is spaces.
SET6_NAME	CHAR(16)	eXadas column. Name of the 6th set associated with this table. Null and default value is spaces.
SET7_NAME	CHAR(16)	eXadas column. Name of the 7th set associated with this table. Null and default value is spaces.
SET8_NAME	CHAR(16)	eXadas column. Name of the 8th set associated with this table. Null and default value is spaces.
SET9_NAME	CHAR(16)	eXadas column. Name of the 9th set associated with this table. Null and default value is spaces.
INDEX1_NAME	CHAR(16)	eXadas column. Name of the 1st index associated with this table. Null and default value is spaces.
INDEX1_KEY_LENGTH	SMALLINT	eXadas column. Length of the 1st index key associated with this table. Null and default value is zeros.
INDEX1_SORT_ORDER	CHAR(1)	eXadas column. Sorting order of the 1st index associated with this table. 'A' ascending or 'D' for descending. Null and default value is spaces.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
INDEX1_SET_TYPE	CHAR(30)	eXadas column. Method used to access the 1st index associated with this table. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX2_NAME	CHAR(16)	eXadas column. Name of the 2nd index associated with this table. Null and default value is spaces.
INDEX2_KEY_LENGTH	SMALLINT	eXadas column. Length of the 2nd index key associated with this table. Null and default value is zeros.
INDEX2_SORT_ORDER	CHAR(1)	eXadas column. Sorting order of the 2nd index associated with this table. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX2_SET_TYPE	CHAR(30)	eXadas column. Method used to access the 2nd index associated with this table. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX3_NAME	CHAR(16)	eXadas column. Name of the 3rd index associated with this table. Null and default value is spaces.
INDEX3_KEY_LENGTH	SMALLINT	eXadas column. Length of the 3rd index key associated with this table. Null and default value is zeros.
INDEX3_SORT_ORDER	CHAR(1)	eXadas column. Sorting order of the 3rd index associated with this table. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX3_SET_TYPE	CHAR(30)	eXadas column. Method used to access the 3rd index associated with this table. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX4_NAME	CHAR(16)	eXadas column. Name of the 4th index associated with this table. Null and default value is spaces.
INDEX4_KEY_LENGTH	SMALLINT	eXadas column. Length of the 4th index key associated with this table. Null and default value is zeros.
INDEX4_SORT_ORDER	CHAR(1)	eXadas column. Sorting order of the 4th index associated with this table. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX4_SET_TYPE	CHAR(30)	eXadas column. Method used to access the 4th index associated with this table. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
INDEX5_NAME	CHAR(16)	eXadas column. Name of the 5th index associated with this table. Null and default value is spaces.
INDEX5_KEY_LENGTH	SMALLINT	eXadas column. Length of the 5th index key associated with this table. Null and default value is zeros.
INDEX5_SORT_ORDER	CHAR(1)	eXadas column. Sorting order of the 5th index associated with this table. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX5_SET_TYPE	CHAR(30)	eXadas column. Method used to access the 5th index associated with this table. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX6_NAME	CHAR(16)	eXadas column. Name of the 6th index associated with this table. Null and default value is spaces.
INDEX6_KEY_LENGTH	SMALLINT	eXadas column. Length of the 6th index key associated with this table. Null and default value is zeros.
INDEX6_SORT_ORDER	CHAR(1)	eXadas column. Sorting order of the 6th index associated with this table. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX6_SET_TYPE	CHAR(30)	eXadas column. Method used to access the 6th index associated with this table. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX7_NAME	CHAR(16)	eXadas column. Name of the 7th index associated with this table. Null and default value is spaces.
INDEX7_KEY_LENGTH	SMALLINT	eXadas column. Length of the 7th index key associated with this table. Null and default value is zeros.
INDEX7_SORT_ORDER	CHAR(1)	eXadas column. Sorting order of the 7th index associated with this table. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX7_SET_TYPE	CHAR(30)	eXadas column. Method used to access the 7th index associated with this table. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX8_NAME	CHAR(16)	eXadas column. Name of the 8th index associated with this table. Null and default value is spaces.
INDEX8_KEY_LENGTH	SMALLINT	eXadas column. Length of the 8th index key associated with this table. Null and default value is zeros.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
INDEX8_SORT_ORDER	CHAR(1)	eXadas column. Sorting order of the 8th index associated with this table. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX8_SET_TYPE	CHAR(30)	eXadas column. Method used to access the 8th index associated with this table. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX9_NAME	CHAR(16)	eXadas column. Name of the 9th index associated with this table. Null and default value is spaces.
INDEX9_KEY_LENGTH	SMALLINT	eXadas column. Length of the 9th index key associated with this table. Null and default value is zeros.
INDEX9_SORT_ORDER	CHAR(1)	eXadas column. Sorting order of the 9th index associated with this table. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX9_SET_TYPE	CHAR(30)	eXadas column. Method used to access the 9th index associated with this table. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
INDEX10_NAME	CHAR(16)	eXadas column. Name of the 10th index associated with this table. Null and default value is spaces.
INDEX10_KEY_LENGTH	SMALLINT	eXadas column. Length of the 10th index key associated with this table. Null and default value is zeros.
INDEX10_SORT_ORDER	CHAR(1)	eXadas column. Sorting order of the 10th index associated with this table. 'A' ascending or 'D' for descending. Null and default value is spaces.
INDEX10_SET_TYPE	CHAR(30)	eXadas column. Method used to access the 10th index associated with this table. Values are a combination of MANUAL, OPTIONAL, VSAM or NONE. Null and default value is spaces.
AREA1_NAME	CHAR(16)	eXadas column. Name of the 1st area associated with this table. Null and default value is spaces.
AREA2_NAME	CHAR(16)	eXadas column. Name of the 2nd area associated with this table. Null and default value is spaces.
AREA3_NAME	CHAR(16)	eXadas column. Name of the 3rd area associated with this table. Null and default value is spaces.
AREA4_NAME	CHAR(16)	eXadas column. Name of the 4th area associated with this table. Null and default value is spaces.

**Table 101: SYSCAC.SYSIDMSTABLES**

Column Name	Data Type	Description
AREA5_NAME	CHAR(16)	eXadas column. Name of the 5th area associated with this table. Null and default value is spaces.
AREA6_NAME	CHAR(16)	eXadas column. Name of the 6th area associated with this table. Null and default value is spaces.
AREA7_NAME	CHAR(16)	eXadas column. Name of the 7th area associated with this table. Null and default value is spaces.
AREA8_NAME	CHAR(16)	eXadas column. Name of the 8th area associated with this table. Null and default value is spaces.
AREA9_NAME	CHAR(16)	eXadas column. Name of the 9th area associated with this table. Null and default value is spaces.
AREA10_NAME	CHAR(16)	eXadas column. Name of the 10th area associated with this table. Null and default value is spaces.

**Table 102: SYSCAC.SYSIMSCOLUMNNS**

Column Name	Data Type	Description
NAME	CHAR(30) NOT NULL	DB2 column. Name of the column.
TBNAME	CHAR(18) NOT NULL	DB2 column. Name of the table that contains the column.
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table that the column is in.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of column within the table. Numbers start at 1.
COLTYPE	CHAR(8) NOT NULL	DB2 column. Type of column, values are INTEGER, SMALLINT, FLOAT, CHAR, VARCHAR, LONGVAR, DECIMAL, GRAPHIC, VARG, and LONGVARG.
LENGTH	SMALLINT NOT NULL	DB2 column. For a DECIMAL data type the columns precision. for all other data types the columns length.
SCALE	SMALLINT NOT NULL	DB2 column. Scale for a DECIMAL data type, otherwise zero.
NULLS	CHAR(1) NOT NULL	DB2 column. Y/N flag identifying whether the column can contain null values.
COLCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.



**Table 102: SYSCAC.SYSIMSCOLUMNS**

Column Name	Data Type	Description
HIGH2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
LOW2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
UPDATES	CHAR(1)	DB2 column. For DB2 tables, the corresponding UPDATES value from the DB2 catalog. For system tables always 'N'. For all other types of tables always 'Y'. Null and default value is 'Y'.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the column. Null and default value is spaces.
DEFAULT	CHAR(1)	DB2 column. For DB2 tables, the corresponding DEFAULT value from the DB2 catalog or for a meta data table whether the column has a NULL IS specification. For all other types of tables 'N'. Null and default value is spaces.
KEYSEQ	SMALLINT	DB2 column. The columns numeric position within the primary key, Zero if the column is not part of the primary key. Null and default value is zeros.
FOREIGNKEY	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
FLDPROC	CHAR(1)	DB2 column. Flag indicating whether the column has a field procedure associated with it. 'Y' - field procedure exists, otherwise 'N'. Null and default value is 'N'.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIELD_OFFSET	SMALLINT NOT NULL	eXadas column. Relative zero offset of the start of the column in the record or segment.
FIELD_LENGTH	SMALLINT NOT NULL	eXadas column. Physical length of the field.
NATIVE_DATA_TYPE	CHAR(1) NOT NULL	eXadas column. Native type of the the underlying column. For a description of the different data types refer to the documentation on the Meta Data Utility in the System Reference Manual.
SQL_COLUMN_LENGTH	SMALLINT NOT NULL	eXadas column. The SQL length of the column.

**Table 102: SYSCAC.SYSIMSCOLUMNS**

Column Name	Data Type	Description
DB2_DATA_TYPE	SMALLINT NOT NULL	eXadas column. The DB2 SQLDA data type value for the column.
PRECISION	SMALLINT NOT NULL	eXadas column. For a DECIMAL data type the columns scale. for all other data types zero.
COLUMN_SEQUENCE_NUMBER	SMALLINT NOT NULL	eXadas column. Relative zero ordinal position of this column in the table.
USE_RECORD_LENGTH	SMALLINT	eXadas column. For variable length records, whether the entire contents of the record should be mapped to the column. A value of 1 indicates that the record length should be used. Null and default value is spaces.
FRAGMENT_ID	SMALLINT NOT NULL	eXadas column. The ID of the fragment that the column is located in.
FRAGMENT_LEVEL	SMALLINT NOT NULL	eXadas column. The level number of the fragment that the column is located in.
OFFSET_IN_FRAGMENT	SMALLINT NOT NULL	eXadas column. Relative zero starting offset of the column within the fragment.
NULLABLE	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating whether the column supports null values.
SIGNED	CHAR(1) NOT NULL	eXadas column. Is the underlying column's data type signed or unsigned? 'Y' the data is signed, 'N' the data is unsigned.
NULL_DATA_LENGTH	SMALLINT	eXadas column. Length of the NULL IS specification for this column Null and default value is zeros.
NULL_VALUE	CHAR(16)	eXadas column. The value for this column that means the column contains a null value. Null and default value is spaces.
FIELD_PROCEDURE_NAME	CHAR(8)	eXadas column. Name of the field procedure associated with this column. Null and default value is spaces.
OCCURS_DEPENDENT_ON	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag that identifies whether the column exists within an OCCURS DEPENDING ON record array. Null and default value is 'N'.
FRAGMENT_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the starting position of the fragment within the linear buffer. Null and default value is -1.

**Table 102: SYSCAC.SYSIMSCOLUMNS**

Column Name	Data Type	Description
FRAGMENT_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. Length of the fragment. If fragment level is 0 the length is the length of the record or segment. For non-zero fragment levels the length of the RECORD ARRAY. Null and default value is -1.
FRAGMENT_MAXIMUM_OCCURRENCES	SMALLINT	eXadas column - obtained from fragment definition. Maximum times the fragment can occur. Null and default value is zeros.
NULL_FRAGMENT_RULE	CHAR(30)	eXadas column - obtained from fragment definition. Method used to determine a NULL occurrence in a RECORD ARRAY. Null and default value is spaces.
OCCURS_DEPENDING_ON_COLUMN	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the name of the column that identifies the number of occurrences that exist. Null and default value is spaces.
OCCURS_DEPENDING_ON_COLNO	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the COLUMN_SEQUENCE_NUMBER of the column that identifies the number of occurrences that exist. Null and default value is zeros.
FRAGMENT_NULL_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the length of the NULL IS specification for the fragment. Null and default value is zeros.
FRAGMENT_NULL_VALUE	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the value that identifies an instance as being null. Null and default value is spaces.
IMS_FIELD_NAME	CHAR(8)	eXadas column. Name of the IMS FIELD associated with this column. Null and default value is spaces.
SEGMENT_INDEX	SMALLINT NOT NULL	eXadas column. The relative zero index into the segment table that identifies the segment associated with this column.
SEGMENT_NAME	CHAR(8) NOT NULL	eXadas column. Name of the IMS segment that the column is located in.
FIELD_TYPE	CHAR(8)	eXadas column. Identifies whether the column is part of a HDAM key field or an XDFLD. A value of 'HDAMKEY' indicates that the column maps to a HDAM primary key. A value of 'XDFLD' indicates the column is mapped against an XDFLD. Null and default value is spaces.

**Table 102: SYSCAC.SYSIMSCOLUMNS**

Column Name	Data Type	Description
PCBPREFIX	CHAR(8)	eXadas column. For a reference to an XDFLD, the PCBNAME prefix that is used to select a PCB to access the IMS DBD using a secondary index. Null and default value is spaces.
XDFLD_NAME	CHAR(8)	eXadas column. The name of the the XDFLD to use in generated SSA's when this table is accessed via a secondary index. Null and default value is spaces.
XDFLD_IN_SEGMENT	CHAR(1)	eXadas column. Y/N flag that indicates, if the column is mapped to an XDFLD and the XDFLD physically exists in the segment. Null and default value is spaces.
KEY_OFFSET	SMALLINT	eXadas column. If the column is mapped against part or all of the primary key, then the relative zero offset of the columns position within the segment. Null and default value is zeros.
KEY_LENGTH	SMALLINT	eXadas column. If the column is mapped against part or all of the primary key, then the length of this column. Null and default value is zeros.

**Table 103: SYSCAC.SYSIMSINDEXES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of index.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the index.
TBNAME	CHAR(18) NOT NULL	DB2 column. Name of the table that the index is referencing.
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
UNIQUERULE	CHAR(1) NOT NULL	DB2 column. Whether the index is unique. 'U' yes or 'D' no.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns that make up the key.
CLUSTERING	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
CLUSTERED	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.

**Table 103: SYSCAC.SYSIMSINDEXES**

Column Name	Data Type	Description
DBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
ISOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
DBNAME	CHAR(8) NOT NULL	DB2 column. Not used by eXadas.
INDEXSPACE	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIRSTKEYCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
FULLKEYCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
NLEAF	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
NLEVELS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
BPOOL	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
PGSIZE	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
ERASERULE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
DSETPASS	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLOSERULE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'Y'.
SPACE	SPACE	DB2 column. Not used by eXadas. Null and default value is zeros.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
CLUSTERRATIO	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.

**Table 103: SYSCAC.SYSIMSINDEXES**

Column Name	Data Type	Description
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the index.
FIELD_TYPE	CHAR(1) NOT NULL	eXadas column. Identifies the type of IMS field that the index is referencing. 'P' - primary key, 'X' secondary index XDFLD.
FIELD_NAME	CHAR(8) NOT NULL	eXadas column. The name of the primary key FIELD or the name of the XDFLD that the index is defined for.
PCBPREFIX	CHAR(8)	eXadas column. The PCBNAME prefix that is used to select a PCB to access the IMS DBD via this index. Null and default value is spaces.

**Table 104: SYSCAC.SYSIMSKEYS**

Column Name	Data Type	Description
IXNAME	CHAR(18) NOT NULL	DB2 column. Name of the index.
IXCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the index.
COLNAME	CHAR(30) NOT NULL	DB2 column. Name of the column for this key.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of the column in the table.
COLSEQ	SMALLINT NOT NULL	DB2 column. Position of this column within the key.
ORDERING	CHAR(1) NOT NULL	DB2 column. Sort order of this column within the key. 'A' ascending or 'D' decending.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.

**Table 105: SYSCAC.SYSIMSSEGMENTS**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of the table.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
TYPE	CHAR(1) NOT NULL	DB2 column. Type of object. 'T' for a table or 'V' for a view.
DBNAME	CHAR(8) NOT NULL	DB2 column. Identifies the type of DBMS that the table is referencing. Values are \$ADABAS, \$CFI (System Table), \$DATACOM, \$DB2, \$IAM, \$IMS, \$IDMS, \$MDS (Meta Data Table) \$SEQUENT and \$VSAM.
TSNAME	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
DBID	SMALLINT	DB2 column. Internal identifier of the database. Zero if the the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Internal identifier of the table. Zero if the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns defined for this table.
EDPROC	CHAR(8)	DB2 column. Name of the edit procedure. Blank if the row describes a view or the table does not use a edit procedure. Only populated for IDMS tables that use compression routines. Null and default value is spaces.
VALPROC	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERTYPE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERRID	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
CARD	INTEGER	DB2 column. Number of rows in the table. Null and default value is -1.
NPAGES	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.

**Table 105: SYSCAC.SYSIMSSEGMENTS**

Column Name	Data Type	Description
PCTPAGES	SMALLINT	DB2 column. Not used by eXadas. Null and default value is -1.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the table. Null and default value is spaces.
PARENTS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CHILDREN	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
KEYCOLUMNS	SMALLINT	DB2 column. Number of columns that make up the tables primary key. Only maintained for system tables. For all other tables 0. Null and default value is zeros.
RECLENGTH	SMALLINT NOT NULL	DB2 column. Number of bytes in a row. This is also the size of the linear buffer allocated for this table.
STATUS	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
KEYOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKFLAG	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKRID	CHAR(4)	DB2 column. Not used by eXadas. Null and default value is spaces.
AUDIT	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the table.
LOCATION	CHAR(16)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBCREATOR	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBNAME	CHAR(18)	DB2 column. Not used by eXadas. Null and default value is spaces.



**Table 105: SYSCAC.SYSIMSSEGMENTS**

Column Name	Data Type	Description
FRAGMENTS	SMALLINT NOT NULL	eXadas column. Number of fragments used to manage this table.
DBMS	CHAR(8) NOT NULL	eXadas column. Identifies the type of database. Always 'IMS'.
DATA_CAPTURE	CHAR(1)	DB2 column. Identifies whether the table has been marked for xSync data capture. A value of 'Y' indicates that changes will be captured. Null and default value is spaces.
SORT_KEYS	SMALLINT	eXadas column. Total number of sort keys for all indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
INDEXES	SMALLINT	eXadas column. Total number of indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
PRIMARY_KEY_OFFSET	SMALLINT NOT NULL	eXadas column. Offset in the root segment of the primary key sequence field.
PRIMARY_KEY_LENGTH	SMALLINT NOT NULL	eXadas column. Length of the primary key sequence field.
DBD_NAME	CHAR(8) NOT NULL	eXadas column. Name of the DBD that the table is accessing.
DBD_TYPE	CHAR(8) NOT NULL	eXadas column. Type of DBD. Values are HDAM, HIDAM, DEDB SHISAM, HSAM, LOGICAL, INDEX or OTHER.
PCBPREFIX	CHAR(8)	eXadas column. PCBNAME prefix that is used to select a PCB to access the IMS DBD. Null and default value is spaces.
STANDARD_PSB	CHAR(8)	eXadas column. Name of the PSB that is scheduled to access the IMS DBD in non-join queries. Null and default value is spaces.
JOIN_PSB	CHAR(8)	eXadas column. Name of the PSB that is scheduled to access the IMS DBD in join queries. Null and default value is spaces.
KEY_COLUMN	SMALLINT NOT NULL	eXadas column. Ordinal number number of the column that maps to the left most portion of the primary key sequence field.
KEY_OFFSET	SMALLINT NOT NULL	eXadas column. Relative zero offset of the primary key sequence field in the root segment.

**Table 105: SYSCAC.SYSIMSSEGMENTS**

Column Name	Data Type	Description
KEY_LENGTH	SMALLINT NOT NULL	eXadas column. Length of the primary key sequence field.
KEY_FIELD_NAME	CHAR(8) NOT NULL	eXadas column. IMS FIELD name of the primary key sequence field.
NUMBER_OF_SEGMENTS	SMALLINT NOT NULL	eXadas column. Number of segments that this table references.
SEGM_NAME	CHAR(8) NOT NULL	eXadas column. Name of segment.
SEGM_LEVEL	CHAR(2) NOT NULL	eXadas column, Hierachic level of segment.
SEGM_LENGTH	SMALLINT NOT NULL	eXadas column. Length of segment. For variable length segments this is the maximum length.
SEGM_FRAGMENT_ID	SMALLINT NOT NULL	eXadas column - obtained from fragment definition. Fragment ID associated with the segment.
SEGM_OFFSET_IN_BUFFER	SMALLINT NOT NULL	eXadas column - obtained from fragment definition. Relative zero offset of the start of the segment in the linear buffer.
SEGM_SYSTEM_KEY	CHAR(1) NOT NULL	eXadas column - obtained from fragment definition. Y/N flag indicating whether the segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area.
SEGM_VARIABLE_LENGTH	CHAR(1) NOT NULL	eXadas column - obtained from fragment definition. Y/N flag indicating whether the segment is variable length.

**Table 106: SYSCAC.SYSIMSTABLES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of the table.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
TYPE	CHAR(1) NOT NULL	DB2 column. Type of object. 'T' for a table or 'V' for a view.

**Table 106: SYSCAC.SYSIMSTABLES**

Column Name	Data Type	Description
DBNAME	CHAR(8) NOT NULL	DB2 column. Identifies the type of DBMS that the table is referencing. Values are \$ADABAS, \$CFI (System Table), \$DATACOM, \$DB2, \$IAM, \$IMS, \$IDMS, \$MDS (Meta Data Table) \$SEQUENT and \$VSAM.
TSNAME	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
DBID	SMALLINT	DB2 column. Internal identifier of the database. Zero if the the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Internal identifier of the table. Zero if the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns defined for this table.
EDPROC	CHAR(8)	DB2 column. Name of the edit procedure. Blank if the row describes a view or the table does not use a edit procedure. Only populated for IDMS tables that use compression routines. Null and default value is spaces.
VALPROC	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERTYPE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERRID	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
CARD	INTEGER	DB2 column. Number of rows in the table. Null and default value is -1.
NPAGES	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
PCTPAGES	SMALLINT	DB2 column. Not used by eXadas. Null and default value is -1.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the table. Null and default value is spaces.

**Table 106: SYSCAC.SYSIMSTABLES**

Column Name	Data Type	Description
PARENTS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CHILDREN	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
KEYCOLUMNS	SMALLINT	DB2 column. Number of columns that make up the tables primary key. Only maintained for system tables. For all other tables 0. Null and default value is zeros.
RECLENGTH	SMALLINT NOT NULL	DB2 column. Number of bytes in a row. This ia also the size of the linear buffer allocated for this table.
STATUS	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
KEYOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKFLAG	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKRID	CHAR(4)	DB2 column. Not used by eXadas. Null and default value is spaces.
AUDIT	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the table.
LOCATION	CHAR(16)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBCREATOR	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBNAME	CHAR(18)	DB2 column. Not used by eXadas. Null and default value is spaces.
FRAGMENTS	SMALLINT NOT NULL	eXadas column. Number of fragments used to mangage this table.
DBMS	CHAR(8) NOT NULL	eXadas column. Identifies the type of database. Always 'IMS'.

**Table 106: SYSCAC.SYSIMSTABLES**

Column Name	Data Type	Description
DATA_CAPTURE	CHAR(1)	DB2 column. Identifies whether the table has been marked for xSync data capture. A value of 'Y' indicates that changes will be captured. Null and default value is spaces.
SORT_KEYS	SMALLINT	eXadas column. Total number of sort keys for all indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
INDEXES	SMALLINT	eXadas column. Total number of indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
PRIMARY_KEY_OFFSET	SMALLINT NOT NULL	eXadas column. Offset in the root segment of the primary key sequence field.
PRIMARY_KEY_LENGTH	SMALLINT NOT NULL	eXadas column. Length of the primary key sequence field.
DBD_NAME	CHAR(8) NOT NULL	eXadas column. Name of the DBD that the table is accessing.
DBD_TYPE	CHAR(8) NOT NULL	eXadas column. Type of DBD. Values are HDAM, HIDAM, DEDB SHISAM, HSAM, LOGICAL, INDEX or OTHER.
PCBPREFIX	CHAR(8)	eXadas column. PCBNAME prefix that is used to select a PCB to access the IMS DBD. Null and default value is spaces.
STANDARD_PSB	CHAR(8)	eXadas column. Name of the PSB that is scheduled to access the IMS DBD in non-join queries. Null and default value is spaces.
JOIN_PSB	CHAR(8)	eXadas column. Name of the PSB that is scheduled to access the IMS DBD in join queries. Null and default value is spaces.
KEY_COLUMN	SMALLINT NOT NULL	eXadas column. Ordinal number number of the column that maps to the left most portion of the primary key sequence field.
KEY_OFFSET	SMALLINT NOT NULL	eXadas column. Relative zero offset of the primary key sequence field in the root segment.
KEY_LENGTH	SMALLINT NOT NULL	eXadas column. Length of the primary key sequence field.
KEY_FIELD_NAME	CHAR(8) NOT NULL	eXadas column. IMS FIELD name of the primary key sequence field.

**Table 106: SYSCAC.SYSIMSTABLES**

Column Name	Data Type	Description
NUMBER_OF_SEGMENTS	SMALLINT NOT NULL	eXadas column. Number of segments that this table references.
SEGM1_NAME	CHAR(8) NOT NULL	eXadas column. Name of the 1st segment referenced by this table.
SEGM1_LEVEL	SEGM1_LEVEL	eXadas column. Hierarchic level of the 1st segment referenced by this table.
SEGM1_LENGTH	SMALLINT NOT NULL	eXadas column. Length of the 1st segment referenced by this table. For variable length segments, this is the maximum length.
SEGM1_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 1st segment. Null and default value is zeros.
SEGM1_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 1st segment in the linear buffer. Null and default value is -1.
SEGM1_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 1st segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.
SEGM1_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 1st segment is variable length. Null and default value is spaces.
SEGM2_NAME	CHAR(8)	eXadas column. Name of the 2nd segment referenced by this table. Null and default value is spaces.
SEGM2_LEVEL	CHAR(2)	eXadas column. Hierarchic level of the 2nd segment referenced by this table. Null and default value is '00'.
SEGM2_LENGTH	SMALLINT	eXadas column. Length of the 2nd segment referenced by this table. For variable length segments, this is the maximum length. Null and default value is zeros.
SEGM2_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 2nd segment. Null and default value is zeros.
SEGM2_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 2nd segment in the linear buffer. Null and default value is -1.

**Table 106: SYSCAC.SYSIMSTABLES**

Column Name	Data Type	Description
SEGM2_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 2nd segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.
SEGM2_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 2nd segment is variable length. Null and default value is spaces.
SEGM3_NAME	CHAR(8)	eXadas column. Name of the 3rd segment referenced by this table. Null and default value is spaces.
SEGM3_LEVEL	CHAR(2)	eXadas column. Hierarchic level of the 3rd segment referenced by this table. Null and default value is '00'.
SEGM3_LENGTH	SMALLINT	eXadas column. Length of the 3rd segment referenced by this table. For variable length segments, this is the maximum length. Null and default value is zeros.
SEGM3_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 3rd segment. Null and default value is zeros.
SEGM3_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 3rd segment in the linear buffer. Null and default value is -1.
SEGM3_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 3rd segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.
SEGM3_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 3rd segment is variable length. Null and default value is spaces.
SEGM4_NAME	CHAR(8)	eXadas column. Name of the 4th segment referenced by this table. Null and default value is spaces.
SEGM4_LEVEL	CHAR(2)	eXadas column. Hierarchic level of the 4th segment referenced by this table. Null and default value is '00'.
SEGM4_LENGTH	SMALLINT	eXadas column. Length of the 4th segment referenced by this table. For variable length segments, this is the maximum length. Null and default value is zeros.

**Table 106: SYSCAC.SYSIMSTABLES**

Column Name	Data Type	Description
SEGM4_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 4th segment. Null and default value is zeros.
SEGM4_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 4th segment in the linear buffer. Null and default value is -1.
SEGM4_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 4th segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.
SEGM4_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 4th segment is variable length. Null and default value is spaces.
SEGM5_NAME	CHAR(8)	eXadas column. Name of the 5th segment referenced by this table. Null and default value is spaces.
SEGM5_LEVEL	CHAR(2)	eXadas column. Hierarchic level of the 5th segment referenced by this table. Null and default value is '00'.
SEGM5_LENGTH	SMALLINT	eXadas column. Length of the 5th segment referenced by this table. For variable length segments, this is the maximum length. Null and default value is zeros.
SEGM5_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 5th segment. Null and default value is zeros.
SEGM5_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 5th segment in the linear buffer. Null and default value is -1.
SEGM5_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 5th segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.
SEGM5_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 5th segment is variable length. Null and default value is spaces.
SEGM6_NAME	CHAR(8)	eXadas column. Name of the 6th segment referenced by this table. Null and default value is spaces.



**Table 106: SYSCAC.SYSIMSTABLES**

Column Name	Data Type	Description
SEGM6_LEVEL	CHAR(2)	eXadas column. Hierarchic level of the 6th segment referenced by this table. Null and default value is '00'.
SEGM6_LENGTH	SMALLINT	eXadas column. Length of the 6th segment referenced by this table. For variable length segments, this is the maximum length. Null and default value is zeros.
SEGM6_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 6th segment. Null and default value is zeros.
SEGM6_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 6th segment in the linear buffer. Null and default value is -1.
SEGM6_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 6th segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.
SEGM6_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 6th segment is variable length. Null and default value is spaces.
SEGM7_NAME	CHAR(8)	eXadas column. Name of the 7th segment referenced by this table. Null and default value is spaces.
SEGM7_LEVEL	CHAR(2)	eXadas column. Hierarchic level of the 7th segment referenced by this table. Null and default value is '00'.
SEGM7_LENGTH	SMALLINT	eXadas column. Length of the 7th segment referenced by this table. For variable length segments, this is the maximum length. Null and default value is zeros.
SEGM7_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 7th segment. Null and default value is zeros.
SEGM7_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 7th segment in the linear buffer. Null and default value is -1.
SEGM7_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 7th segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.

**Table 106: SYSCAC.SYSIMSTABLES**

Column Name	Data Type	Description
SEGM7_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 7th segment is variable length. Null and default value is spaces.
SEGM8_NAME	CHAR(8)	eXadas column. Name of the 8th segment referenced by this table. Null and default value is spaces.
SEGM8_LEVEL	CHAR(2)	eXadas column. Hierarchic level of the 8th segment referenced by this table. Null and default value is '00'.
SEGM8_LENGTH	SMALLINT	eXadas column. Length of the 8th segment referenced by this table. For variable length segments, this is the maximum length. Null and default value is zeros.
SEGM8_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 8th segment. Null and default value is zeros.
SEGM8_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 8th segment in the linear buffer. Null and default value is -1.
SEGM8_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 8th segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.
SEGM8_VARIABLE_LENGTH	CHAR(1)	eXadas column obtained from fragment definition. Y/N flag indicating whether the 8th segment is variable length. Null and default value is spaces.
SEGM9_NAME	CHAR(8)	eXadas column. Name of the 9th segment referenced by this table. Null and default value is spaces.
SEGM9_LEVEL	CHAR(2)	eXadas column. Hierarchic level of the 9th segment referenced by this table. Null and default value is '00'.
SEGM9_LENGTH	SMALLINT	eXadas column. Length of the 9th segment referenced by this table. For variable length segments, this is the maximum length. Null and default value is zeros.
SEGM9_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 9th segment. Null and default value is zeros.
SEGM9_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 9th segment in the linear buffer Null and default value is -1.

**Table 106: SYSCAC.SYSIMSTABLES**

Column Name	Data Type	Description
SEGM9_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 9th segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.
SEGM9_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 9th segment is variable length. Null and default value is spaces.
SEGM10_NAME	CHAR(8)	eXadas column. Name of the 10th segment referenced by this table. Null and default value is spaces.
SEGM10_LEVEL	CHAR(2)	eXadas column. Hierarchic level of the 10th segment referenced by this table. Null and default value is '00'.
SEGM10_LENGTH	SMALLINT	eXadas column. Length of the 10th segment referenced by this table. For variable length segments, this is the maximum length. Null and default value is zeros.
SEGM10_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 10th segment. Null and default value is zeros.
SEGM10_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 10th segment in the linear buffer. Null and default value is -1.
SEGM10_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 10th segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.
SEGM10_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 10th segment is variable length. Null and default value is spaces.
SEGM11_NAME	CHAR(8)	eXadas column. Name of the 11th segment referenced by this table. Null and default value is spaces.
SEGM11_LEVEL	CHAR(2)	eXadas column. Hierarchic level of the 11th segment referenced by this table. Null and default value is '00'.
SEGM11_LENGTH	SMALLINT	eXadas column. Length of the 11th segment referenced by this table. For variable length segments, this is the maximum length. Null and default value is zeros.

**Table 106: SYSCAC.SYSIMSTABLES**

Column Name	Data Type	Description
SEGM11_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 11th segment. Null and default value is zeros.
SEGM11_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 11th segment in the linear buffer. Null and default value is -1.
SEGM11_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 11th segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.
SEGM11_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 11th segment is variable length. Null and default value is spaces.
SEGM12_NAME	CHAR(8)	eXadas column. Name of the 12th segment referenced by this table. Null and default value is spaces.
SEGM12_LEVEL	CHAR(2)	eXadas column. Hierarchic level of the 12th segment referenced by this table. Null and default value is '00'.
SEGM12_LENGTH	SMALLINT	SMALLINT
SEGM12_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 12th segment. Null and default value is zeros.
SEGM12_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragments definition. Relative zero offset of the start of the 12th segment in the linear buffer Null and default value is -1.
SEGM12_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 12th segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.
SEGM12_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 12th segment is variable length. Null and default value is spaces.
SEGM13_NAME	CHAR(8)	eXadas column. Name of the 13th segment referenced by this table. Null and default value is spaces.
SEGM13_LEVEL	CHAR(2)	eXadas column. Hierarchic level of the 13th segment referenced by this table. Null and default value is '00'.

**Table 106: SYSCAC.SYSIMSTABLES**

Column Name	Data Type	Description
SEGM13_LENGTH	SMALLINT	eXadas column. Length of the 13th segment referenced by this table. For variable length segments, this is the maximum length. Null and default value is zeros.
SEGM13_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 13th segment. Null and default value is zeros.
SEGM13_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 13th segment in the linear buffer. Null and default value is -1.
SEGM13_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 13th segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.
SEGM13_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 13th segment is variable length. Null and default value is spaces.
SEGM14_NAME	CHAR(8)	eXadas column. Name of the 14th segment referenced by this table. Null and default value is spaces.
SEGM14_LEVEL	CHAR(2)	eXadas column. Hierarchic level of the 14th segment referenced by this table. Null and default value is '00'.
SEGM14_LENGTH	SMALLINT	eXadas column. Length of the 14th segment referenced by this table. For variable length segments, this is the maximum length. Null and default value is zeros.
SEGM14_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 14th segment. Null and default value is zeros.
SEGM14_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 14th segment in the linear buffer. Null and default value is -1.
SEGM14_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 14th segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.
SEGM14_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 14th segment is variable length. Null and default value is spaces.

**Table 106: SYSCAC.SYSIMSTABLES**

Column Name	Data Type	Description
SEGM15_NAME	CHAR(8)	eXadas column. Name of the 15th segment referenced by this table. Null and default value is spaces.
SEGM15_LEVEL	CHAR(2)	eXadas column. Hierarchic level of the 15th segment referenced by this table. Null and default value is '00'.
SEGM15_LENGTH	SMALLINT	eXadas column. Length of the 15th segment referenced by this table. For variable length segments, this is the maximum length. Null and default value is zeros.
SEGM15_FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the 15th segment. Null and default value is zeros.
SEGM15_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the start of the 15th segment in the linear buffer Null and default value is -1.
SEGM15_SYSTEM_KEY	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 15th segment does not represent an actual segment definition and is instead an XDFLD key value extracted from the key feedback area. Null and default value is spaces.
SEGM15_VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the 15th segment is variable length. Null and default value is spaces.

**Table 107: SYSCAC.SYSMETACOLUMNS**

Column Name	Data Type	Description
NAME	CHAR(30) NOT NULL	DB2 column. Name of the column.
TBNAME	CHAR(18) NOT NULL	DB2 column. Name of the table that contains the column.
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table that the column is in.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of column within the table. Numbers start at 1.
COLTYPE	CHAR(8) NOT NULL	DB2 column. Type of column, values are INTEGER, SMALLINT, FLOAT, CHAR, VARCHAR, LONGVAR, DECIMAL, GRAPHIC, VARG, and LONGVARG.

**Table 107: SYSCAC.SYSMETACOLUMNS**

Column Name	Data Type	Description
LENGTH	SMALLINT NOT NULL	DB2 column. For a DECIMAL data type the columns precision. for all other data types the columns length.
SCALE	SMALLINT NOT NULL	DB2 column. Scale for a DECIMAL data type, otherwise zero.
NULLS	CHAR(1) NOT NULL	DB2 column. Y/N flag identifying whether the column can contain null values.
COLCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
HIGH2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
LOW2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
UPDATES	CHAR(1)	DB2 column. For DB2 tables, the corresponding UPDATES value from the DB2 catalog. For system tables always 'N'. For all other types of tables always 'Y'. Null and default value is 'Y'.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the column. Null and default value is spaces.
DEFAULT	CHAR(1)	DB2 column. For DB2 tables, the corresponding DEFAULT value from the DB2 catalog or for a meta data table whether the column has a NULL IS specification. For all other types of tables 'N'. Null and default value is spaces.
KEYSEQ	KEYSEQ	DB2 column. The columns numeric position within the primary key, Zero if the column is not part of the primary key. Null and default value is zeros.
FOREIGNKEY	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
FLDPROC	CHAR(1)	DB2 column. Flag indicating whether the column has a field procedure associated with it. 'Y' - field procedure exists, otherwise 'N'. Null and default value is 'N'.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIELD_OFFSET	SMALLINT NOT NULL	eXadas column. Relative zero offset of the start of the column in the rexord or segment.

**Table 107: SYSCAC.SYSMETACOLUMNS**

Column Name	Data Type	Description
FIELD_LENGTH	SMALLINT NOT NULL	eXadas column. Physical length of the field.
NATIVE_DATA_TYPE	CHAR(1) NOT NULL	eXadas column. Native type of the the underlying column. For a description of the different data types refer to the documentation on the Meta Data Utility in the System Reference Manual.
SQL_COLUMN_LENGTH	SMALLINT NOT NULL	eXadas column. The SQL length of the column.
DB2_DATA_TYPE	SMALLINT NOT NULL	eXadas column. The DB2 SQLDA data type value for the column.
PRECISION	SMALLINT NOT NULL	eXadas column. For a DECIMAL data type the columns scale. for all other data types zero.
COLUMN_SEQUENCE_NUMBER	SMALLINT NOT NULL	eXadas column. Relative zero ordinal position of this column in the table.
USE_RECORD_LENGTH	SMALLINT	eXadas column. For variable length records, whether the entire contents of the record should be mapped to the column. A value of 1 indicates that the record length should be used. Null and default value is spaces.
FRAGMENT_ID	SMALLINT NOT NULL	eXadas column. The ID of the fragment that the column is located in.
FRAGMENT_LEVEL	SMALLINT NOT NULL	eXadas column. The level number of the fragment that the column is located in.
OFFSET_IN_FRAGMENT	SMALLINT NOT NULL	eXadas column. Relative zero starting offset of the column within the fragment.
NULLABLE	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating whether the column supports null values.
SIGNED	CHAR(1) NOT NULL	eXadas column. Is the underlying column's data type signed or unsigned?'Y' the data is signed, 'N' the data is unsigned.
NULL_DATA_LENGTH	SMALLINT	eXadas column. Length of the NULL IS specification for this column Null and default value is zeros.
NULL_VALUE	CHAR(16)	eXadas column. The value for this column that means the column contains a null value. Null and default value is spaces.
FIELD_PROCEDURE_NAME	CHAR(8)	eXadas column. Name of the field procedure associated with this column. Null and default value is spaces.



**Table 107: SYSCAC.SYSMETACOLUMNS**

Column Name	Data Type	Description
OCCURS_DEPENDING_ON	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag that identifies whether the column exists within an OCCURS DEPENDING ON record array. Null and default value is 'N'.
FRAGMENT_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the starting position of the fragment within the linear buffer. Null and default value is -1.
FRAGMENT_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. Length of the fragment. If fragment level is 0 the length is the length of the record or segment. For non-zero fragment levels the length of the RECORD ARRAY. Null and default value is -1.
FRAGMENT_MAXIMUM_OCCURRENCES	SMALLINT	eXadas column - obtained from fragment definition. Maximum times the fragment can occur. Null and default value is zeros.
NULL_FRAGMENT_RULE	CHAR(30)	eXadas column - obtained from fragment definition. Method used to determine a NULL occurrence in a RECORD ARRAY. Null and default value is spaces.
OCCURS_DEPENDING_ON_COLUMN	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the name of the column that identifies the number of occurrences that exist. Null and default value is spaces.
OCCURS_DEPENDING_ON_COLNO	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the COLUMN_SEQUENCE_NUMBER of the column that identifies the number of occurrences that exist. Null and default value is zeros.
FRAGMENT_NULL_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the length of the NULL IS specification for the fragment. Null and default value is zeros.
FRAGMENT_NULL_VALUE	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the value that identifies an instance as being null. Null and default value is spaces.

**Table 108: SYSCAC.SYSMETAFRAGMENTS**

Column Name	Data Type	Description
CREATOR	CHAR(8)	Owner of the fragment. Null and default value is spaces.
NAME	CHAR(32)	CHAR(32)
DBNAME	CHAR(32)	Database containing the fragment. Null and default value is spaces.
TYPE	CHAR(1) NOT NULL	Fragment type. T=top-level fragment, E=embedded fragment.
VARIABLE_OCCURRENCES	CHAR(1) NOT NULL	Is number of occurrences variable? Y or N.
SOURCE_DBMS	CHAR(4)	Database Type of source. VSAM, etc. Null and default value is spaces.
SOURCE_CREATOR	CHAR(8)	Owner of source table definition. Null and default value is spaces.
SOURCE_TABLE	CHAR(32)	Table name of source table definition. Null and default value is spaces.
SOURCE_NAME	CHAR(32)	Name of the source object. For IMS tables, this is the segment name. For IDMS this is the record name. Null and default value is spaces.
PARENT_NAME	CHAR(32)	If embedded, name of containing fragment. Null and default value is spaces.
SYSTEM_FIELD	CHAR(1) NOT NULL	Fragment is a database system field. For IMS 'Y' = XDFLD KEY from PCB key feedback area. For IDMS 'Y' = DBKEY and RRDS information.
VARIABLE_LENGTH	CHAR(1) NOT NULL	Variable length (Y). Indicates that the record of segment starts with a halfword that identifies the actual length of the fragment.
ID	SMALLINT NOT NULL	ID of fragment - order of definition.
LEVEL	SMALLINT NOT NULL	Level 0 = TOP, 1, 2, 3 etc if the fragment is associated with a RECORD ARRAY.
OFFSET	SMALLINT NOT NULL	Relative zero offset of fragment within parent fragment. (within record if no parent, ie, top-level).
LENGTH	SMALLINT NOT NULL	Length of fragment (in bytes).

**Table 108: SYSCAC.SYSMETAFRAGMENTS**

Column Name	Data Type	Description
MAXIMUM_OCCURRENCES	SMALLINT NOT NULL	Maximum number of times the fragment occurs within its parent.
NULLIS_RULE	SMALLINT	If variable number of occurrences present, how a null instance is determined. Null and default value is spaces.
TBCREATOR	CHAR(8) NOT NULL	Owner of table to which fragment is mapped.
TBNAME	CHAR(18) NOT NULL	Name of table to which fragment is mapped.
CONTROL_COLUMN	CHAR(32)	Name of column with occurs count, if any. Null and default value is spaces.
CONTROL_COLUMN_SEQUENCE_NUMBER	SMALLINT	Control columns, COLUMN_SEQUENCE_NUMBER. Null and default value is spaces.
NULL_LENGTH	SMALLINT	Length of null value string. Null and default value is zeros.
NULL_VALUE	CHAR(32)	String value that identifies an instance as null. Null and default value is spaces.
DEPCOL_SQL_TYPE	SMALLINT	SQL data type for depending on column. Null and default value is spaces.
DEPCOL_FRAGMENT_LEVEL	SMALLINT	Fragment level of the depending on column. Null and default value is spaces.
DEPCOL_OFFSET	SMALLINT	Depending on columns offset in its fragment. Null and default value is spaces.
DEPCOL_LENGTH	SMALLINT	Depending on columns length in its fragment. Null and default value is spaces.

**Table 109: SYSCAC.SYSMETATABLES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of the table.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
TYPE	CHAR(1) NOT NULL	DB2 column. Type of object. 'T' for a table or 'V' for a view.

**Table 109: SYSCAC.SYSMETATABLES**

Column Name	Data Type	Description
DBNAME	CHAR(8) NOT NULL	DB2 column. Identifies the type of DBMS that the table is referencing. Values are \$ADABAS, \$CFI (System Table), \$DATACOM, \$DB2, \$IAM, \$IMS, \$IDMS, \$MDS (Meta Data Table) \$SEQUENT and \$VSAM.
TSNAME	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
DBID	SMALLINT	DB2 column. Internal identifier of the database. Zero if the the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Internal identifier of the table. Zero if the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns defined for this table.
EDPROC	CHAR(8)	DB2 column. Name of the edit procedure. Blank if the row describes a view or the table does not use a edit procedure. Only populated for IDMS tables that use compression routines. Null and default value is spaces.
VALPROC	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERTYPE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERRID	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
CARD	INTEGER	DB2 column. Number of rows in the table. Null and default value is -1.
NPAGES	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
PCTPAGES	SMALLINT	DB2 column. Not used by eXadas. Null and default value is -1.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the table. Null and default value is spaces.

**Table 109: SYSCAC.SYSMETATABLES**

Column Name	Data Type	Description
PARENTS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CHILDREN	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
KEYCOLUMNS	SMALLINT	DB2 column. Number of columns that make up the tables primary key. Only maintained for system tables. For all other tables 0. Null and default value is zeros.
RECLENGTH	SMALLINT NOT NULL	DB2 column. Number of bytes in a row. This ia also the size of the linear buffer allocated for this table.
STATUS	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
KEYOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKFLAG	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKRID	CHAR(4)	DB2 column. Not used by eXadas. Null and default value is spaces.
AUDIT	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the table.
LOCATION	CHAR(16)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBCREATOR	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBNAME	CHAR(18)	DB2 column. Not used by eXadas. Null and default value is spaces.
FRAGMENTS	SMALLINT NOT NULL	eXadas column. Number of fragments used to manage this table.
DBMS	CHAR(8) NOT NULL	Type identifier for the database being mapped. Values are ABABAS, CFI, DB2, DATACOM, IDMS, IMS, MDS, VSAM and SEQUENTI.

**Table 110: SYSCAC.SYSSEQCOLUMNS**

Column Name	Data Type	Description
NAME	CHAR(3) NOT NULL	DB2 column. Name of the column.
TBNAME	CHAR(18) NOT NULL	CHAR(18) NOT NULL
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table that the column is in.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of column within the table. Numbers start at 1.
COLTYPE	CHAR(8) NOT NULL	DB2 column. Type of column, values are INTEGER, SMALLINT, FLOAT, CHAR, VARCHAR, LONGVAR, DECIMAL, GRAPHIC, VARG, and LONGVARG.
LENGTH	SMALLINT NOT NULL	DB2 column. For a DECIMAL data type the columns precision. for all other data types the columns length.
SCALE	SMALLINT NOT NULL	DB2 column. Scale for a DECIMAL data type, otherwise zero.
NULLS	CHAR(1) NOT NULL	DB2 column. Y/N flag identifying whether the column can contain null values.
COLCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
HIGH2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
LOW2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
UPDATES	CHAR(1)	DB2 column. For DB2 tables, the corresponding UPDATES value from the DB2 catalog. For system tables always 'N'. For all other types of tables always 'Y'. Null and default value is 'Y'.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the column. Null and default value is spaces.
DEFAULT	CHAR(1)	DB2 column. For DB2 tables, the corresponding DEFAULT value from the DB2 catalog or for a meta data table whether the column has a NULL IS specification. For all other types of tables 'N'. Null and default value is spaces.

**Table 110: SYSCAC.SYSSEQCOLUMNS**

Column Name	Data Type	Description
KEYSEQ	SMALLINT	DB2 column. The columns numeric position within the primary key, Zero if the column is not part of the primary key. Null and default value is zeros.
FOREIGNKEY	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
FLDPROC	CHAR(1)	DB2 column. Flag indicating whether the column has a field procedure associated with it. 'Y' - field procedure exists, otherwise 'N'. Null and default value is 'N'.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIELD_OFFSET	SMALLINT NOT NULL	eXadas column. Relative zero offset of the start of the column in the record or segment.
FIELD_LENGTH	SMALLINT NOT NULL	eXadas column. Physical length of the field.
NATIVE_DATA_TYPE	CHAR(1) NOT NULL	eXadas column. Native type of the the underlying column. For a description of the different data types refer to the documentation on the Meta Data Utility in the System Reference Manual.
SQL_COLUMN_LENGTH	SMALLINT NOT NULL	eXadas column. The SQL length of the column.
DB2_DATA_TYPE	SMALLINT NOT NULL	eXadas column. The DB2 SQLDA data type value for the column.
PRECISION	SMALLINT NOT NULL	eXadas column. For a DECIMAL data type the columns scale. for all other data types zero.
COLUMN_SEQUENCE_NUMBER	SMALLINT NOT NULL	eXadas column. Relative zero ordinal position of this column in the table.
USE_RECORD_LENGTH	SMALLINT	eXadas column. For variable length records, whether the entire contents of the record should be mapped to the column. A value of 1 indicates that the record length should be used. Null and default value is spaces.
FRAGMENT_ID	SMALLINT NOT NULL	eXadas column. The ID of the fragment that the column is located in.
FRAGMENT_LEVEL	SMALLINT NOT NULL	eXadas column. The level number of the fragment that the column is located in.
OFFSET_IN_FRAGMENT	SMALLINT NOT NULL	eXadas column. Relative zero starting offset of the column within the fragment.

**Table 110: SYSCAC.SYSSEQCOLUMNS**

Column Name	Data Type	Description
NULLABLE	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating whether the column supports null values.
SIGNED	CHAR(1) NOT NULL	eXadas column. Is the underlying column's data type signed or unsigned? 'Y' the data is signed, 'N' the data is unsigned.
NULL_DATA_LENGTH	SMALLINT	eXadas column. Length of the NULL IS specification for this column Null and default value is zeros.
NULL_VALUE	CHAR(16)	eXadas column. The value for this column that means the column contains a null value. Null and default value is spaces.
FIELD_PROCEDURE_NAME	CHAR(8)	eXadas column. Name of the field procedure associated with this column. Null and default value is spaces.
OCCURS_DEPENDING_ON	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag that identifies whether the column exists within an OCCURS DEPENDING ON record array. Null and default value is 'N'.
FRAGMENT_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the starting position of the fragment within the linear buffer. Null and default value is -1.
FRAGMENT_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. Length of the fragment. If fragment level is 0 the length is the length of the record or segment. For non-zero fragment levels the length of the RECORD ARRAY. Null and default value is -1.
FRAGMENT_MAXIMUM_OCCURRENCES	SMALLINT	eXadas column - obtained from fragment definition. Maximum times the fragment can occur. Null and default value is zeros.
NULL_FRAGMENT_RULE	CHAR(30)	eXadas column - obtained from fragment definition. Method used to determine a NULL occurrence in a RECORD ARRAY. Null and default value is spaces.
OCCURS_DEPENDING_ON_COLUMN	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the name of the column that identifies the number of occurrences that exist. Null and default value is spaces.



**Table 110: SYSCAC.SYSSEQCOLUMNS**

Column Name	Data Type	Description
OCCURS_DEPENDING_ON_COLNO	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the COLUMN_SEQUENCE_NUMBER of the column that identifies the number of occurrences that exist. Null and default value is zeros.
FRAGMENT_NULL_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the length of the NULL IS specification for the fragment. Null and default value is zeros.
FRAGMENT_NULL_VALUE	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the value that identifies an instance as being null. Null and default value is spaces.

**Table 111: SYSCAC.SYSSEQTABLES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of the table.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
TYPE	CHAR(1) NOT NULL	DB2 column. Type of object. 'T' for a table or 'V' for a view.
DBNAME	CHAR(8) NOT NULL	DB2 column. Identifies the type of DBMS that the table is referencing. Values are \$ADABAS, \$CFI (System Table), \$DATACOM, \$DB2, \$IAM, \$IMS, \$IDMS, \$MDS (Meta Data Table) \$SEQUENT and \$VSAM.
TSNAME	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
DBID	SMALLINT	DB2 column. Internal identifier of the database. Zero if the the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Internal identifier of the table. Zero if the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns defined for this table.

**Table 111: SYSCAC.SYSSEQTABLES**

Column Name	Data Type	Description
EDPROC	CHAR(8)	DB2 column. Name of the edit procedure. Blank if the row describes a view or the table does not use a edit procedure. Only populated for IDMS tables that use compression routines. Null and default value is spaces.
VALPROC	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERTYPE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERRID	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
CARD	INTEGER	DB2 column. Number of rows in the table. Null and default value is -1.
NPAGES	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
PCTPAGES	SMALLINT	DB2 column. Not used by eXadas. Null and default value is -1.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the table. Null and default value is spaces.
PARENTS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CHILDREN	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
KEYCOLUMNS	SMALLINT	DB2 column. Number of columns that make up the tables primary key. Only maintained for system tables. For all other tables 0. Null and default value is zeros.
RECLENGTH	SMALLINT NOT NULL	DB2 column. Number of bytes in a row. This ia also the size of the linear buffer allocated for this table.
STATUS	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
KEYOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.

**Table 111: SYSCAC.SYSSEQTABLES**

Column Name	Data Type	Description
CHECKFLAG	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKRID	CHAR(4)	DB2 column. Not used by eXadas. Null and default value is spaces.
AUDIT	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the table.
LOCATION	CHAR(16)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBCREATOR	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBNAME	CHAR(18)	DB2 column. Not used by eXadas. Null and default value is spaces.
FRAGMENTS	SMALLINT NOT NULL	eXadas column. Number of fragments used to manage this table.
DBMS	CHAR(8) NOT NULL	eXadas column. Identifies the type of database. Always 'SEQUENTI'.
RECORD_EXIT_NAME	CHAR(8)	eXadas column. Load module name of the exit routine invoked to process this table. Null and default value is spaces.
RECORD_EXIT_MAXIMUM_LENGTH	INTEGER	eXadas column. Maximum buffer length used by the record exit. Null and default value is spaces.
DYNAMIC_ALLOCATION	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating whether the file is dynamically allocated.
FILE_NAME	CHAR(44) NOT NULL	eXadas column. Name of the file. If the file is dynamically allocated this is the full dataset name. If the file is not dynamically allocated this is the DD name to be opened.
RECORD_LENGTH	INTEGER NOT NULL	eXadas column. Physical length of record. For variable length records this is the maximum length of the record.
RECORD_FORMAT	CHAR(4)	eXadas column. File record format. Null and default value is spaces.
ACCESS_MODE	CHAR(40)	eXadas column. Method used to access the file. This column is only populated in VSE versions of Exadas. Null and default value is spaces.

**Table 111: SYSCAC.SYSSEQTABLES**

Column Name	Data Type	Description
FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with this file. Null and default value is zeros.
VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the record is variable length. Null and default value is spaces.

**Table 112: SYSCAC.SYSVSAMCOLUMNS**

Column Name	Data Type	Description
NAME	CHAR(30) NOT NULL	DB2 column. Name of the column.
TBNAME	CHAR(18) NOT NULL	DB2 column. Name of the table that contains the column.
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table that the column is in.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of column within the table. Numbers start at 1.
COLTYPE	CHAR(8) NOT NULL	DB2 column. Type of column, values are INTEGER, SMALLINT, FLOAT, CHAR, VARCHAR, LONGVAR, DECIMAL, GRAPHIC, VARG, and LONGVARG.
LENGTH	SMALLINT NOT NULL	DB2 column. For a DECIMAL data type the columns precision. for all other data types the columns length.
SCALE	SMALLINT NOT NULL	DB2 column. Scale for a DECIMAL data type, otherwise zero.
NULLS	CHAR(1) NOT NULL	DB2 column. Y/N flag identifying whether the column can contain null values.
COLCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.
HIGH2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
LOW2KEY	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.

**Table 112: SYSCAC.SYSVSAMCOLUMNS**

Column Name	Data Type	Description
UPDATES	CHAR(1)	DB2 column. For DB2 tables, the corresponding UPDATES value from the DB2 catalog. For system tables always 'N'. For all other types of tables always 'Y'. Null and default value is 'Y'.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the column. Null and default value is spaces.
DEFAULT	CHAR(1)	DB2 column. For DB2 tables, the corresponding DEFAULT value from the DB2 catalog or for a meta data table whether the column has a NULL IS specification. For all other types of tables 'N'. Null and default value is spaces.
KEYSEQ	SMALLINT	DB2 column. The columns numeric position within the primary key, Zero if the column is not part of the primary key. Null and default value is zeros.
FOREIGNKEY	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
FLDPROC	CHAR(1)	DB2 column. Flag indicating whether the column has a field procedure associated with it. 'Y' - field procedure exists, otherwise 'N'. Null and default value is 'N'.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIELD_OFFSET	SMALLINT NOT NULL	eXadas column. Relative zero offset of the start of the column in the rexord or segment.
FIELD_LENGTH	SMALLINT NOT NULL	eXadas column. Physical length of the field.
NATIVE_DATA_TYPE	CHAR(1) NOT NULL	eXadas column. Native type of the the underlying column. For a description of the different data types refer to the documentation on the Meta Data Utility in the System Reference Manual.
SQL_COLUMN_LENGTH	SMALLINT NOT NULL	eXadas column. The SQL length of the column.
DB2_DATA_TYPE	SMALLINT NOT NULL	eXadas column. The DB2 SQLDA data type value for the column.
PRECISION	SMALLINT NOT NULL	eXadas column. For a DECIMAL data type the columns scale. for all other data types zero.

**Table 112: SYSCAC.SYVSAMCOLUMNS**

Column Name	Data Type	Description
COLUMN_SEQUENCE_NUMBER	SMALLINT NOT NULL	eXadas column. Relative zero ordinal position of this column in the table.
USE_RECORD_LENGTH	SMALLINT	eXadas column. For variable length records, whether the entire contents of the record should be mapped to the column. A value of 1 indicates that the record length should be used. Null and default value is spaces.
FRAGMENT_ID	SMALLINT NOT NULL	eXadas column. The ID of the fragment that the column is located in.
FRAGMENT_LEVEL	SMALLINT NOT NULL	eXadas column. The level number of the fragment that the column is located in.
OFFSET_IN_FRAGMENT	SMALLINT NOT NULL	eXadas column. Relative zero starting offset of the column within the fragment.
NULLABLE	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating whether the column supports null values.
SIGNED	CHAR(1) NOT NULL	eXadas column. Is the underlying column's data type signed or unsigned? 'Y' the data is signed, 'N' the data is unsigned.
NULL_DATA_LENGTH	SMALLINT	eXadas column. Length of the NULL IS specification for this column Null and default value is zeros.
NULL_VALUE	CHAR(16)	eXadas column. The value for this column that means the column contains a null value. Null and default value is spaces.
FIELD_PROCEDURE_NAME	CHAR(8)	eXadas column. Name of the field procedure associated with this column. Null and default value is spaces.
OCCURS_DEPENDENT_ON	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag that identifies whether the column exists within an OCCURS DEPENDING ON record array. Null and default value is 'N'.
FRAGMENT_OFFSET_IN_BUFFER	SMALLINT	eXadas column - obtained from fragment definition. Relative zero offset of the starting position of the fragment within the linear buffer. Null and default value is -1.
FRAGMENT_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. Length of the fragment. If fragment level is 0 the length is the length of the record or segment. For non-zero fragment levels the length of the RECORD ARRAY. Null and default value is -1.

**Table 112: SYSCAC.SYSVSAMCOLUMNS**

Column Name	Data Type	Description
FRAGMENT_MAXIMUM_OCCURRENCES	SMALLINT	eXadas column - obtained from fragment definition. Maximum times the fragment can occur. Null and default value is zeros.
NULL_FRAGMENT_RULE	CHAR(30)	eXadas column - obtained from fragment definition. Method used to determine a NULL occurrence in a RECORD ARRAY. Null and default value is spaces.
OCCURS_DEPENDING_ON_COLUMN	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the name of the column that identifies the number of occurrences that exist. Null and default value is spaces.
OCCURS_DEPENDING_ON_COLNO	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the COLUMN_SEQUENCE_NUMBER of the column that identifies the number of occurrences that exist. Null and default value is zeros.
FRAGMENT_NULL_LENGTH	SMALLINT	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the length of the NULL IS specification for the fragment. Null and default value is zeros.
FRAGMENT_NULL_VALUE	CHAR(32)	eXadas column - obtained from fragment definition. For RECORD ARRAY fragments, the value that identifies an instance as being null. Null and default value is spaces.
CLUSTER_TYPE	CHAR(1)	eXadas column. For a column that references the primary key, the type of VSAM cluster being referenced. Values are 'R' or 'C'. Null and default value is spaces.
FILE_ORGANIZATION	CHAR(4)	eXadas column. For a column that references the primary key, the dataset organization of the VSAM file. Values are KSDS, ESDS or RRDS. This column is only populated for VSE versions of eXadas. Null and default value is spaces.
CATALOG_NAME	CHAR(44)	eXadas column. For a column that references the primary key, the name of the catalog where the VSAM file is defined. This column is only populated for VSE versions of eXadas. Null and default value is spaces.
DYNAMIC_ALLOCATION	CHAR(1)	eXadas column. For a column that references the primary key or an alternate index whether the VSAM file is dynamically allocated. Y - yes, N - the file is allocated to the server. Null and default value is spaces.

**Table 112: SYSCAC.SYSVSAMCOLUMNS**

Column Name	Data Type	Description
FILE_NAME	CHAR(44)	eXadas column. For a column that references the primary key or an alternate index the dataset name (when using dynamic allocation) or the DD name that is used to access the file. Null and default value is spaces.
KEY_OFFSET	SMALLINT	eXadas column. For a column that references the primary key or an alternate index the offset of the key within the record. Null and default value is zeros.
KEY_LENGTH	SMALLINT	eXadas column. For a column that references the primary key or an alternate index the length of the key. Null and default value is zeros.
RECORD_LENGTH	SMALLINT	eXadas column. Maximum record length of the VSAM file. Null and default value is zeros.

**Table 113: SYSCAC.SYSVSAMINDEXES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of index.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the index.
TBNAME	CHAR(18) NOT NULL	DB2 column. Name of the table that the index is referencing.
TBCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
UNIQUERULE	CHAR(1) NOT NULL	DB2 column. Whether the index is unique. 'U' yes or 'D' no.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns that make up the key.
CLUSTERING	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
CLUSTERED	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
DBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.



**Table 113: SYSCAC.SYSVSAMINDEXES**

Column Name	Data Type	Description
ISOBID	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
DBNAME	CHAR(8) NOT NULL	DB2 column. Not used by eXadas.
INDEXSPACE	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
FIRSTKEYCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
FULLKEYCARD	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
NLEAF	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
NLEVELS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
BPOOL	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
PGSIZE	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
ERASERULE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
DSETPASS	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLOSERULE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'Y'.
SPACE	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
CLUSTERRATIO	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the index.
DYNAMIC_ALLOCATION	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating whether the VSAM file is dynamically allocated.

**Table 113: SYSCAC.SYSVSAMINDEXES**

Column Name	Data Type	Description
FILE_NAME	CHAR(44) NOT NULL	eXadas column. Dataset name of the VSAM file to be dynamically allocated or the DD name to be opened when not using dynamic allocation.
CATALOG_NAME	CHAR(44)	eXadas column. Name of the VSAM catalog that the file is defined in. This column is only populated in VSE versions of eXadas. Null and default value is spaces.

**Table 114: SYSCAC.SYSVSAMKEYS**

Column Name	Data Type	Description
IXNAME	CHAR(18) NOT NULL	DB2 column. Name of the index.
IXCREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the index.
COLNAME	CHAR(30) NOT NULL	DB2 column. Name of the column for this key.
COLNO	SMALLINT NOT NULL	DB2 column. Ordinal position of the column in the table.
COLSEQ	SMALLINT NOT NULL	DB2 column. Position of this column within the key.
ORDERING	CHAR(1) NOT NULL	DB2 column. Sort order of this column within the key. 'A' ascending or 'D' descending.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.

**Table 115: SYSCAC.SYSVSAMTABLES**

Column Name	Data Type	Description
NAME	CHAR(18) NOT NULL	DB2 column. Name of the table.
CREATOR	CHAR(8) NOT NULL	DB2 column. Authorization ID of the owner of the table.
TYPE	CHAR(1) NOT NULL	DB2 column. Type of object. 'T' for a table or 'V' for a view.
DBNAME	CHAR(8) NOT NULL	DB2 column. Identifies the type of DBMS that the table is referencing. Values are \$ADABAS, \$CFI (System Table), \$DATACOM, \$DB2, \$IAM, \$IMS, \$IDMS, \$MDS (Meta Data Table) \$SEQUENT and \$VSAM.
TSNAME	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
DBID	SMALLINT	DB2 column. Internal identifier of the database. Zero if the the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
OBID	SMALLINT	DB2 column. Internal identifier of the table. Zero if the row describes a view or the database does not have an identifier. Only populated for imported DB2 tables and IDMS tables. Null and default value is zeros.
COLCOUNT	SMALLINT NOT NULL	DB2 column. Number of columns defined for this table.
EDPROC	CHAR(8)	DB2 column. Name of the edit procedure. Blank if the row describes a view or the table does not use a edit procedure. Only populated for IDMS tables that use compression routines. Null and default value is spaces.
VALPROC	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERTYPE	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CLUSTERRID	INTEGER	DB2 column. Not used by eXadas. Null and default value is zeros.
CARD	INTEGER	DB2 column. Number of rows in the table. Null and default value is -1.
NPAGES	INTEGER	DB2 column. Not used by eXadas. Null and default value is -1.

**Table 115: SYSCAC.SYSVSAMTABLES**

Column Name	Data Type	Description
PCTPAGES	SMALLINT	DB2 column. Not used by eXadas. Null and default value is -1.
IBMREQD	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is 'N'.
REMARKS	CHAR(254)	DB2 column. A user supplied character string describing the table. Null and default value is spaces.
PARENTS	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
CHILDREN	SMALLINT	DB2 column. Not used by eXadas. Null and default value is zeros.
KEYCOLUMNS	SMALLINT	DB2 column. Number of columns that make up the tables primary key. Only maintained for system tables. For all other tables 0. Null and default value is zeros.
RECLENGTH	SMALLINT NOT NULL	DB2 column. Number of bytes in a row. This is also the size of the linear buffer allocated for this table.
STATUS	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
KEYOBID	KEYOBID	DB2 column. Not used by eXadas. Null and default value is zeros.
LABEL	CHAR(30)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKFLAG	CHAR(1)	DB2 column. Not used by eXadas. Null and default value is spaces.
CHECKRID	CHAR(4)	DB2 column. Not used by eXadas. Null and default value is spaces.
AUDIT	CHAR(1)	CHAR(1)
CREATEDBY	CHAR(8) NOT NULL	DB2 column. Primary authorization ID of the user that created the table.
LOCATION	CHAR(16)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBCREATOR	CHAR(8)	DB2 column. Not used by eXadas. Null and default value is spaces.
TBNAME	CHAR(18)	DB2 column. Not used by eXadas. Null and default value is spaces.

**Table 115: SYSCAC.SYSVSAMTABLES**

Column Name	Data Type	Description
FRAGMENTS	SMALLINT NOT NULL	eXadas column. Number of fragments used to manage this table.
DBMS	CHAR(8) NOT NULL	eXadas column. Identifies the type of database. Always 'VSAM'.
DATA_CAPTURE	CHAR(1)	DB2 column. Identifies whether the table has been marked for xSync data capture. A value of 'Y' indicates that changes will be captured. Null and default value is spaces.
SORT_KEYS	SMALLINT	eXadas column. Total number of sort keys for all indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
INDEXES	SMALLINT	eXadas column. Total number of indexes defined for this table. The value for this column is automatically maintained by eXadas. Null and default value is zeros.
RECORD_EXIT_NAME	CHAR(8)	eXadas column. Load module name of the exit routine invoked to process this table. Null and default value is spaces.
RECORD_EXIT_MAXIMUM_LENGTH	INTEGER	eXadas column. Maximum buffer length used by the record exit. Null and default value is spaces.
CLUSTER_TYPE	CHAR(1) NOT NULL	eXadas column. Identifies type of cluster of the VSAM file that the table is referencing. 'C' base cluster, 'P' alternate index path.
FILE_ORGANIZATION	CHAR(4) NOT NULL	eXadas column. Identifies the VSAM files organization. Values are KSDS, ESDS and RRDS. This column is only populated for VSE versions of eXadas.
CATALOG_NAME	CHAR(44)	eXadas column. Name of the catalog where the VSAM file is defined. This column is only populated for VSE versions of eXadas. Null and default value is spaces.
DYNAMIC_ALLOCATION	CHAR(1) NOT NULL	eXadas column. Y/N flag indicating whether the VSAM file is dynamically allocated.
FILE_NAME	CHAR(44) NOT NULL	eXadas column. Name of the VSAM file. If the file is dynamically allocated this is the full dataset name. If the file is not dynamically allocated this is the DD name to be opened.
RECORD_LENGTH	INTEGER NOT NULL	eXadas column. Physical length of record. For variable length records this is the maximum length of the record.

**Table 115: SYSCAC.SYSVSAMTABLES**

Column Name	Data Type	Description
KEY_COLUMN_SEQUENCE_NUMBER	SMALLINT	eXadas column. The COLUMN_SEQUENCE_NUMBER of the first column in this table that references the starting offset of the primary sequence field. Null and default value is -1.
KEY_OFFSET	SMALLINT	eXadas column. Relative zero offset of the beginning of the primary key field in the record. Null and default value is -1.
KEY_LENGTH	SMALLINT	eXadas column. Length of the VSAM primary key. Null and default value is zeros.
CICS_LOCAL_LU	CHAR(8)	eXadas column. Name of the local LU used to communicate with CICS. Null and default value is spaces.
CICS_REMOTE_LU	CHAR(8)	eXadas column. Name of the remote LU that CICS is using for VTAM communications. Null and default value is spaces.
CICS_LOGMODE	CHAR(8)	eXadas column, MODETAB entry name used to communication with CICS. Null and default value is spaces.
CICS_TRANSACTION_NAME	CHAR(4)	eXadas column. Name of the CICS transaction used to access or update the VSAM file. Null and default value is spaces.
CICS_NETWORK_NAME	CHAR(8)	eXadas column. Network name for a remote CICS subsystem. Null and default value is spaces.
FRAGMENT_ID	SMALLINT	eXadas column - obtained from fragment definition. Fragment ID associated with the VSAM file. Null and default value is zeros.
VARIABLE_LENGTH	CHAR(1)	eXadas column - obtained from fragment definition. Y/N flag indicating whether the VSAM file contains variable length records. Null and default value is spaces.

## A

- ABORT 21
- Active services, displaying 119
- ADABAS
  - Meta Data Utility customization 148
  - SAF exit validation 227
  - SQL update 353, 360
  - table source definition 158
  - USE Statement Generator 192–195
- ADK-based programming interface 22
- Administrative components 9
- Alternate index, VSAM 113
- API 2, 16–17
- APPC, connection parameters for Meta Data Utility 191
- Application components 9
- Application Group Name 41
- Application migration 16
- Application models 16–17
- Application programming interface
  - See API
- Applications
  - deploying 26–33
  - tuning 97–116
- ASMTDLI interface 6, 35, 110
- ATM 64
- Authorization violations 235

## B

- Bandwidth 64
- Batch jobs 27, 117
  - Enterprise Servers 339
- BMP interface 35, 40–42
  - compared with DBB interface 25
  - supported concurrent users 19
- Brio Query 20
- BTREE BUFFERS parameter 385

## C

- C precompiler 2, 3
- CACSAMP 20, 22
  - inner/outer loops 23
  - parameter markers 22–23
- CACSX04 security exit 44, 45
- CA-DATACOM/DB
  - accessing a different MUF 53
  - communicating with Query Processors 53
  - data, preparing servers to access 6
  - initialization service 52
  - load library 52
  - Meta Data Utility customization 146
  - Multi-User Facility 6
  - security 54, 222
  - Source Language Generation 146

- SQL update 360
- table source definition 159
- CALL statement 293
- CICS VSAM
  - grammar 190–192
  - SQL update 353, 366
  - stored procedures 267–269, 297
    - CACSP62 abend codes 306
    - CACSPBR interface 297–302
    - CACSPBR return codes 305
    - compiling 304
    - parameters 303
    - transaction scheduling for 282
  - table source definition 160
  - transport protocol 366
- CICS, stored procedure s414
- CLIENT CONFIG parameter 405
- Client connection requests, listening for 4, 7
- Client connectors 8, 16
  - loading 8
- Client Interface Module 2, 7
- Client/server application 16
- COBOL copybooks
  - as input for Meta Data Utility 143
  - importing into DataMapper 87
- COBOL II 7
- Columns
  - creating in DataMapper 87
  - defining 161
  - subset views 133
  - using views to filter 132
- COMMIT statement 354, 356
- two-phase commit t354
- Communications
  - APPC parameters for Meta Data Utility 191
  - ATM 64
  - bandwidth 64
  - configuring 55–56
  - Cross Memory 57
    - communications compound protocol field 57
    - configuration 65
    - Enterprise Server scheduling/connection balancing 349
    - in Enterprise Servers 336, 346
    - Services Communications Handler 18
  - ESCON channels 64
  - frame relay 64
  - hostnames versus IP addresses 64
  - IBM MQ Series 57, 58
    - client connectors 8
    - configuration 58–62, 66
    - Connection Handler 18
    - local queue 18
    - prerequisites 62
    - two-queue definition 58
    - selecting 64

- TCP/IP 63
    - client connectors 8
    - configuration 67
    - Connection Handler Service 18
    - dot-decimal notation 63
    - in Enterprise Servers 346
    - port number limitations 68
    - TSO NETSTAT SOCKET command 19
  - Compound Address field 56
  - Concurrent users
    - increasing number supported with Enterprise Servers 335
    - number supported 19
  - Configuration parameter s381
  - CONNECT TO CICS statement t190
  - CONNECT TO DB2 statement nt184, 185
  - Connection Handler Service, activating 67
  - Connection handlers 4
  - Connections with servers, establishing and maintaining 7
  - Connectors
    - See Client connectors
  - Coordination controller 36
  - Copybooks
    - See COBOL copybooks
  - CPU Governor parameter r385
  - CPU Resource Governor Exit 235
    - activating 236
    - initialization 240
    - validation 240
  - CPU time, limiting 26, 235–242
  - CREATE PROCEDURE statement 145, 275
  - CREATE VIEW statement 130, 145
  - Cross Memory
    - See Communications, Cross Memory
- D**
- Data catalogs, creating in DataMapper 87
  - Data Savants
    - optimizing Query Processor performance 101
    - stored procedure s263
  - Data Source Handler 7
  - Data sources
    - accessing 3
    - JOINing 3
  - Data Space Name 32
  - Data types, supported 170
  - Data warehouse 16
  - Database administrators, deploying application s14
  - Database privileges 74
  - Database types 71
  - Datacom
    - See CA-DATACOM/DB
  - DataMapper 2, 5, 9–11, 21, 87–88
    - and the Meta Data Utility 143
    - field procedure s370
    - FTP support 11, 88
    - generating Meta Data Grammar 87
    - importing COBOL copybooks 87
    - mapping IDMS data sources 49
    - updating Meta Data Catalogs 88
    - See also Mapping data
  - DataMart 16
  - DATASOURCE parameter 386
  - DB2 353
    - Call Attach Facility 6
    - data, preparing servers to access 6
    - grammar 184–190
    - SQL update 362
    - subsystem, connecting to 6
  - DB2 Thread Management Exit 251
    - activating 252
    - developing a custom 254
  - DBB interface 24, 35, 40–42
    - supported concurrent users 19
  - DBCTL subsystem 36
  - DBDs, loading into DataMapper 87
  - DC/UCF system 44
  - DDNAME CACLOG 137
  - Decision Support System 16
  - DECODE BUFFER SIZE parameter 387
  - DEDB databases 107
    - fast path buffer s108
  - DEFLOC parameter 388
  - DELETE command 5, 356
    - CICS VSAM 366
  - Developers, deploying applications 14
  - DISPLAY,CONFIGS command 120
  - DISPLAY,SERVICES command 119, 341
  - DLI calls, reducing 106
  - DML calls, native 44
  - Dot-decimal notation 63
  - DRA interface 35, 36–40
    - compared with DBB interface 25
    - configuration 38
    - DBCTL subsystem 36
    - IMS stored procedure s320–324
    - initializing 6
    - interfacing with IMS stored procedures 270
    - optimizing data access 111
    - setting up p37
    - using in production environment t25
  - DROP INDEX statement 144, 183
  - DROP PROCEDURE statement t145, 281
  - DROP TABLE statement 144, 182
  - DROP VIEW statement 135, 145
- E**
- Enterprise Servers 2, 7, 27, 335
    - adding to existing eXadas networks 347
    - communication protocols 346
    - communications configuration formats 346
    - communications protocols 31
    - configuration 345–347
    - controlling 340
    - deploying 336
    - displaying active services 341
    - displaying configuration information 342
    - displaying connected servers 341
    - displaying memory usage 343
    - dynamic scheduling 350
    - monitoring 340
    - starting 339
    - starting/stopping services 343
    - stopping 344
    - using Cross Memory for scheduling/connection balancing 349
  - ESCON channels 64
  - eXadas eData Engine, 3GL hooks 2
  - Executive Information System 16



**F**

Fast path buffer s108  
 FETCH BUFFER SIZE parameter 388  
 Field data type conversion 167  
 Field encoding 370  
 Field procedures 369  
   samples 378  
   specifying 370  
   value descriptors 373  
 Filtering data with views 132  
 FLUSH command 123  
 Frame relay 64

**G**

Governor limits 23  
 GRANT ALL PRIVILEGES statement 78  
 GRANT EXECUTE statement 76  
 GRANT ON DATABASE statement 74  
 GRANT statement 72, 144  
 Grouped views 133

**H**

HDAM databases  
   optimizing 104  
 HIDAM database  
   defining primary sequence field s104  
   indexes 90  
 High-level language environments, initializing 6  
 High-speed sequential processing 107  
 Hiperspace, improving performance 24  
 Horizontal views 132  
 Hostnames 64  
 HSSP 107

**I**

IBM MQ Series  
   *See* Communications, IBM MQ Series  
 IDMS  
   ACCESS LOADMODs 49  
   central version 46  
   data security 45  
   IDD dictionary definitions as input for Meta Data Utility 143  
   indexes 91  
   mapping data 46  
   MAXERUS value 44  
   Meta Data Utility customization 148  
   METAU 47  
   S047 abend 46  
   schema/subschema 11  
   security 45  
   SQL DELETE command 364  
   SQL INSERT command 363  
   SQL update 353, 362–364  
   table source definition 156  
 Immediate return of data 102  
 IMPORT DB2 INDEX statement 184, 188  
 IMPORT DB2 TABLE statement 184, 186  
 IMPORT DB2 VIEW statement 184  
 IMS  
   ASMTDLI interface 35  
   BMP interface 27  
   data access optimization 103  
   data, preparing servers to access 6

  database definitions 11  
   DBB interface 27  
   DBCTL region, connecting to 6  
   DBD source definitions as input for Meta Data Utility 143  
   defining indexes 104, 182  
   DRA interface. *See* DRA interface  
   fast path buffers 108  
   Fastpath DEDB databases 107  
   indexes 90  
   loading DBDs into DataMapper 87  
   native access s103  
   partial keys 105  
   path calls 106  
   primary indexes 104  
   PSB 24  
   Region Controller 6  
   SAF exit validation 226  
   search fields 105  
   secondary indexes 104, 364  
   security 221  
   segment mapping 96  
   SQL update 353, 364  
   STAGE 1 generations 24  
   stored procedures, accessing/updating with 269  
   table source definitions 153

**Indexes**

  creating in DataMapper 88, 89  
   defining IMS 104, 182  
   defining VSAM 182  
   multi-part keys 89  
   optimizing IMS data access 104  
   primary 113  
   secondary 113  
 Initialization services 6  
   CA-DATACOM/DB 52  
   DRA 270  
 Inner loops 23  
 INSERT 5  
 Insert positioning 358  
 INSERT statement 355  
 Interfaces, ASMTDLI 110  
 INTERLEAVE INTERVAL 389  
 Interlink TCP/IP 68  
 IP addresses  
   defining 63  
   versus hostnames 64

**J**

JCL, customizing 19  
 JDBC connector 2  
 JOIN MAX TABLES ANALYZED parameter 390  
 Joining CA-DATACOM/DB and other data types 53  
 JOINS  
   optimizing 3, 99  
   with view s134

**K**

Keys  
   multi-part 89  
   using to optimize queries 98

**L**

Language Environment  
   deactivating 282

stored procedure s266  
 LD TEMP SPACE paramete r391  
 Listen port number, defining 63  
 Load balancin g7, 335  
 Local queue 18  
 LOCALE parameter 392  
 Log print utili t y39  
 Logging 6, 137  
   trace levels 138  
 Logical tables  
   defining 9  
   mapping 20  
 LONG VARCH A R75  
 LONG VARGRAPHI C176  
**M**  
 Mapping dat a19–21, 85–96  
   IDMS 46  
   IMS segment mapping 96  
   Meta Data Utility 142  
   verification 20  
 Master Configuration Member 28  
   Enterprise Server 32  
 Master Terminal Operator 4, 33  
   saving configuration member updates to dis k34  
 MAX ROWS EXAMINED parameter 21, 393  
 MAX ROWS EXCEEDED ACTION paramete r21, 393  
 MAX ROWS RETURNED parameter 394  
 MAXERUS value 44  
 Memory  
   displaying utilization 123  
   used in Enterprise Servers 343  
   used in stored procedures 264  
 MESSAGE POOL SIZE parameter 395  
 Meta Data Catalogs 9  
   creating 19–20  
   interfacing with DataMapper 87  
   reducing number of reads 102  
   updating with DataMapper 88  
 Meta Data Grammar 9, 144  
   generating 87  
   overview 86  
 Meta Data Utility 9, 47, 88, 142–183  
   ADABAS customization 148  
   APPC connection parameter s191  
   CA-DATACOM/DB customization 146  
   IDMS customization 148  
   running 145  
 METAU 47  
 Microsoft Access 20  
 Microsoft Query 20  
 Model Queue 18  
 MTO commands 423–430  
   *See also individual command names*  
 Multi-part keys 89  
 Multi-User Facility, accessing different 53  
**N**  
 Network administrators, deploying applications 14  
 NL CAT parameter 395  
 NL parameter 395  
 NT ODBC  
   IBM MQ Series 57  
 NULL IS definitions in record arrays 94

**O**  
 OCCURS clauses  
   SQL update 359  
 Occurs processing 92–96  
 ODBC connector 2  
 ODBC Driver Manager, loading connectors 8  
 ODBC stored procedure suppor t295  
 Off-load gateway s68  
 Operations staff, deploying application s14  
 Optimization 97–116  
   of JOINS 99  
   server dispatching priorit y115  
   using DRA interface 111  
   using keys 98  
   VSAM data acces s112  
   *See also Performance*  
 ORDER BY sortin g3  
 OS/390 Master Terminal Operator  
   *See Master Terminal Operator*  
 OS/390 Work Load Manager subsystem  
   *See Work Load Manager*  
 Outer loops 23  
**P**  
 Packed decimal s172  
 Parameter markers 22–23  
 Partial keys  
   IMS 105  
   index definitions 90  
   VSAM 114  
 Partitioned dat a20  
 Path calls  
   IMS 106  
 PCB 108–109  
   selection by nam e109  
   selection by verificatio n108  
 PDQ parameter 396  
 Performance  
   and views 130  
   Hiperspace 24  
   IMS 24  
   monitoring 23–24  
   optimization 97–116  
   Record Processing Exit 259  
   *See also Optimization*  
 Post-query processin g3  
 Precompiler 17  
 Primary indexes  
   IMS 104  
   VSAM 113  
 PSB 109–110  
   creating composite 25  
   scheduling 109  
**Q**  
 Queries  
   creating 22–24  
   optimizing 3, 98–99  
   using views 130  
 Query Processors 5  
   communicating with CA-DATACOM/DB 53  
   configuring data sourc e b8  
   handling views 129  
   optimizing 101–103

- optimizing VSAM 115
  - parsing CA-DATACOM/DB data 53
  - TCB 24
- R**
- Record arrays 92, 176
    - multiple in one record 94
    - NULL IS definitions 94
  - Record data type conversion 167
  - Record Processing Exit 256
    - initialization 257
    - performance 259
  - Record typing 95
  - Records, filtering with view s132
  - Region Controller 4
    - starting servers 124
  - Relational views, creating in DataMapper 88
  - RESPONSE TIME OUT parameter r397
  - Result sets
    - staging 103
    - translating 3
  - RETURN 21
  - REVOKE statement 72, 144
  - RMF Monitor II/III reporting facilities 24
  - ROLLBACK command 354, 357
    - CICS VSAM 366
  - Row subset views 133
- S**
- SAF EXIT
    - parameter description 397
  - SAF exit 220–228
    - activating 221
    - initialization 225
    - security 82
    - termination 228
    - validation 226
  - SDSF 23
  - Search fields
    - using in IMS 105
  - Secondary indexes
    - IMS 104
  - Security 70
    - ACF-2 54
    - and view s130, 134
    - CA-DATACOM/DB 54, 222
    - database privileges 74
    - defining user privilege s72
    - IDMS 45
    - IMS 221
    - RACF 54
    - SAF exit 82, 220–228
      - initialization 225
      - validation 226
    - stored procedures
      - privileges 76
    - system privileges 72
    - user authorizations 81
    - user types 70
    - validation processing g223
  - Security administrators, deploying applications 14
  - Sequential
    - SAF exit validation 227
    - table source definitions 155
  - Server Memory Pool 123
  - Servers 2, 3–7
    - configuration parameters, modifying 122
    - configurations, displaying 120
    - creating 18
    - dispatching priority 115
    - displaying active services 119
    - displaying connected user s119
    - displaying memory utilization n123
    - establishing communications 7
    - logging 137
    - monitoring 33
    - running as started tasks/batch jobs 27, 117
    - starting 33, 118
      - See also* Enterprise Servers
  - SERVICE INFO ENTRY parameter 398
  - Services
    - displaying active 119
    - displaying in Enterprise Servers 341
    - starting/stopping 124
    - starting/stopping in an Enterprise Server 343
  - SET command 122, 343
  - SMF exit 24, 228
    - activating 228
    - authorization violation n335
    - initialization 233
    - parameter 402
    - reporting 26
    - validation 233
  - SQL
    - foreign key s358
    - host variables, dereferencing 7
    - inner/outer loops 23
    - limiting result sets 23–24
    - parameter markers 22–23
    - performance monitoring 23–24
    - rewriting into native file/database access language e3
    - transactions 354
    - update 353
      - insert positioning 358
      - mappings with multiple records 357
      - OCCURS clauses 359
    - validating 3
      - See also* Queries
  - SQL GRANT command 71
  - SQL queries
    - See* Queries
  - SQL update 353
  - Staging, and immediate return of data 102
  - Started tasks 27, 117
    - Enterprise Servers 339
    - starting servers as 337
  - Static catalogs 102
  - STATIC CATALOGS parameter r403
  - Stay resident parameter 265
  - STOP command 124, 344
  - Stored procedure s261, 412
    - CICS
      - CACSP62 abend codes 306
      - CACSPBR interface 302
      - CACSPBR return codes 305
      - compiling 304
      - parameters 303
    - CICS VSAM 267–269
    - defining 274
    - DRA initialization service 270

- examples 273
  - IMS, accessing/updating 269
  - invoking 292–296
  - loading multiple copie s265
  - memory us e264
  - ODBC support 295
  - parameters 277
  - privileges 76
  - SAF exit validation 227
  - scheduling CICS transactions 282
  - stay resident parameter 265
  - support routines 272, 331–333
  - writing 284–292
  - Supported data type s170
  - System exits 219
  - System privileges 72
  - System programmers, deploying applications 14
- T**
- Tables
    - creating in DataMapper 87
    - definitions 151
    - security 78
  - TASK PARAMETERS 404
  - Tasks, monitoring 4
  - TCB
    - communication between Query Processors and CA-DATACOM/DB MUF 53
    - initialization/termination 249
  - TCP/IP
    - See* Communications, TCP/IP
  - TRACE LEVEL paramete r405
  - Trace levels 138
  - Tracing Level field 138
  - Transport layer
    - IBM MQ Series 58
    - loading 7
  - Transport protocol
    - CICS VSAM 366
  - Troubleshooting
    - 817 SQLCODE 354
    - 9999 SQLCODE 362
    - All PCBs are in use 25, 364
    - changes made to stored procedure not appearin g265
    - Connection failures 63
    - Connection Handler Service failure at initializatio n63
    - FY status call 107
    - garbled data returne d21
    - IMS abend during ROLLBACK cal 1270
    - incorrect result sets with signed or unsigned packed decimal fields 172
    - Query Processor terminating unexpectedly y285
    - S047 abend 46
    - stored procedures abending in LE environmen t282
    - truncated data returne d21
  - TSO NETSTAT SOCKET command 19
  - Tuning applications 97–116
  - Two-phase commit 354
- U**
- UPDATE statement 5, 356
  - USE [UNIQUE] INDEX statement 144, 180
  - Use Grammar
    - See* Meta Data Grammar
- USE INDEX statement**  
defining IMS indexes 104
- Use statements**  
*See* Meta Data Grammar
- USE TABLE statement** 144, 150
- User privileges, defining** 72
- User Requirements Table** 147
- USERID parameter** 406
- USERPASSWORD parameter** 407
- Users, supported concurrent** 19
- V**
- Validation processin g223
  - VARCHAR 173
  - VARGRAPHIC 175
  - Vertical views 133
  - Views 128
    - advantages/disadvantages 130
    - creating 130
    - dropping 135
    - filtering dat a132
    - grouped 133
    - horizontal 132
    - joined 134
    - row/column subset 133
    - security 78
    - vertical 133
- VSAM**  
alternate indexes 113  
data access optimizatio n112  
defining indexes 182  
indexes 90  
primary indexes 113  
SAF exit validation 227  
table source definition 156
- VSAM AMPARMS parameter** 407
- VTAM resource definitions** 412
- W**
- Web-based application 16
  - WHERE clauses 359
  - WLM goal mode 7
    - optimization 116
    - support 26
  - WLM system exit 24
  - WLM UOW parameter 409
  - Work Load Manager subsystem 7
  - Workload Manager Exit 242
    - activating 243
    - initialization 248
    - management 249
- X**
- XDFLDs
    - defining for HIDAM/HDAM databases 104
- Z**
- Zoned decimal support 172