CA IDMS - 19.0
Administrating CA IDMS
System Operations

Date: 23-Dec-2017
This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

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

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

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

The manufacturer of this Documentation is CA.

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

Copyright © 2017 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.
# Table of Contents

## Product Intent Module ................................................................. 22
- Create a Product Intent Module ..................................................... 22
  - Example Product Intent Module Creation .................................. 22
- Display Product Intent ................................................................. 24
- Change the Product Intent ............................................................ 24

## Starting Up a DC/UCF System ...................................................... 24
- Define and Generate the DC/UCF System ..................................... 25
- Define Dictionaries and Databases .............................................. 25
- Execute the DC/UCF Startup Routine .......................................... 26
- System Startup Under z/OS .......................................................... 26
  - Execute the Startup Routine using JCL .................................. 27
- System Startup Under z/VM ......................................................... 27
  - Step 1 Assemble the DCPARM Macro ..................................... 28
  - Step 2 Assemble the SVCOP %Macro ....................................... 28
  - Step 3 Link Edit the Startup Routine ..................................... 28
  - Step 4 Execute the startup routine ....................................... 30
- System Startup Under z/VSE ......................................................... 30
  - Step 1 Code DVFILE Macros ................................................... 31
    - Format 1 DVFILE Macro ....................................................... 34
    - Format 1 DVFILE Macro Syntax ......................................... 34
    - Format 1 DVFILE Macro Parameter ................................... 35
    - Format 2 DVFILE Macro ....................................................... 40
    - Format 2 DVFILE Syntax .................................................... 40
    - Format 2 DVFILE Parameter .................................................. 41
    - Format 3 DVFILE Macro ....................................................... 44
  - Step 2 Create an RHDCFTAB Module ....................................... 44
  - Step 3 Execute the Startup Routine ....................................... 44
    - The IDMSLBLs Procedure .................................................. 45
    - Specify Runtime Options .................................................. 45
- What Happens During Startup .................................................... 45
  - How the Startup Routine Acquires Storage .............................. 46
  - How the System is Built ......................................................... 48
  - The DC/UCF Region/Partition Layout ....................................... 53
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler Exits and Entry Points</td>
<td>207</td>
</tr>
<tr>
<td>WTOEXIT and WTOREXIT Implementation</td>
<td>208</td>
</tr>
<tr>
<td>IDMSUXIT Module Creation</td>
<td>208</td>
</tr>
<tr>
<td>DB Exits</td>
<td>208</td>
</tr>
<tr>
<td>IDMSAJNX</td>
<td>209</td>
</tr>
<tr>
<td>IDMSSCLCX</td>
<td>210</td>
</tr>
<tr>
<td>IDMSDPLX</td>
<td>211</td>
</tr>
<tr>
<td>IDMSIOXT</td>
<td>214</td>
</tr>
<tr>
<td>IDMSIOX2</td>
<td>214</td>
</tr>
<tr>
<td>IDMSJNL2</td>
<td>219</td>
</tr>
<tr>
<td>DC/UCF Exits</td>
<td>220</td>
</tr>
<tr>
<td>IDMSSVCX</td>
<td>221</td>
</tr>
<tr>
<td>Steps to Add IDMSSVCX to Your System</td>
<td>227</td>
</tr>
<tr>
<td>TCKREXIT</td>
<td>228</td>
</tr>
<tr>
<td>USRIDIXIT</td>
<td>229</td>
</tr>
<tr>
<td>WAITEXIT</td>
<td>230</td>
</tr>
<tr>
<td>WTOEXIT</td>
<td>231</td>
</tr>
<tr>
<td>WTOEXIT Examples</td>
<td>232</td>
</tr>
<tr>
<td>WTOREXIT</td>
<td>238</td>
</tr>
<tr>
<td>WTOREXIT Examples</td>
<td>240</td>
</tr>
<tr>
<td>CICS Exits</td>
<td>242</td>
</tr>
<tr>
<td>IDMSCEON</td>
<td>243</td>
</tr>
<tr>
<td>IDMSCEOX</td>
<td>244</td>
</tr>
<tr>
<td>OPTIIXIT</td>
<td>245</td>
</tr>
<tr>
<td>OPTIXIT (IDMSINTC)</td>
<td>247</td>
</tr>
<tr>
<td>OPTIXIT Example</td>
<td>249</td>
</tr>
<tr>
<td>OPTIXIT (IDMSINL)</td>
<td>250</td>
</tr>
<tr>
<td>Tools Exits</td>
<td>252</td>
</tr>
<tr>
<td>Hooked Exits</td>
<td>252</td>
</tr>
<tr>
<td>Numbered Exits</td>
<td>255</td>
</tr>
<tr>
<td>Calling Conventions for Numbered Exits</td>
<td>258</td>
</tr>
<tr>
<td>Macros Required for DC/UCF Calling Conventions</td>
<td>259</td>
</tr>
<tr>
<td>MOPT Macro</td>
<td>259</td>
</tr>
<tr>
<td>MOPT Syntax</td>
<td>259</td>
</tr>
<tr>
<td>MOPT Parameter</td>
<td>259</td>
</tr>
<tr>
<td>START Macro</td>
<td>259</td>
</tr>
<tr>
<td>START Syntax</td>
<td>260</td>
</tr>
<tr>
<td>START Parameter</td>
<td>260</td>
</tr>
<tr>
<td>RTN Macro</td>
<td>261</td>
</tr>
<tr>
<td>RTN Syntax</td>
<td>261</td>
</tr>
<tr>
<td>RTN Parameter</td>
<td>261</td>
</tr>
<tr>
<td>RTN Example</td>
<td>262</td>
</tr>
</tbody>
</table>
GETSTK Macro .................................................................................................................. 262
GETSTK Syntax .................................................................................................................... 262
GETSTK Parameter .............................................................................................................. 262
User-invoked Numbered Exits .............................................................................................. 263
XIT exit Syntax ..................................................................................................................... 263
XIT exit Parameter .............................................................................................................. 263
exit Examples ...................................................................................................................... 263
Installing Numbered Exits in the System ........................................................................ 263
Step 1 Code a DFXIT Macro ............................................................................................... 264
DFXIT Syntax ....................................................................................................................... 265
DFXIT Parameters ................................................................................................................ 265
Step 2 Assemble and Link Edit the RHDCUXIT Module ...................................................... 266
System-invoked Numbered Exits ....................................................................................... 268
Exit 0 -- System Initialization Exit .................................................................................... 269
Exit 1 -- Signon Exit ............................................................................................................ 270
Exit 2 -- Signoff Exit ........................................................................................................... 270
Exit 4 -- New Task Exit ....................................................................................................... 270
Exit 5 -- Task Termination Exit I ...................................................................................... 271
Exit 6 -- Task Termination Exit II ..................................................................................... 271
Exit 7 -- Write-to-Log Exit .................................................................................................. 271
Exit 8 -- Log Full Exit ........................................................................................................ 272
Exit 9 -- System Statistics Exit .......................................................................................... 272
Exit 12 -- Terminal I/O Error Exit ...................................................................................... 273
Exit 13 -- Shutdown Exit ..................................................................................................... 273
Exit 14 -- BIND RUN UNIT and READY AREA Exit ........................................................ 273
BIND Parameters ................................................................................................................ 274
Exit 15 -- VIB Statistics Exit ............................................................................................. 275
VIB Parameters ................................................................................................................... 275
Exit 16 -- Write Printer Exit ............................................................................................... 275
Exit 17 -- Input Data Stream Exit ...................................................................................... 276
Input Parameters ................................................................................................................ 276
Exit 18 -- Output Data Stream Exit ................................................................................... 277
Exit 18 Parameter ................................................................................................................. 278
Exit 19 -- Asynchronous Terminal Connection Exit ......................................................... 279
Exit 20 -- Resource Limit Exit .......................................................................................... 279
Exit 20 Parameters ............................................................................................................... 280
Exit 20 Examples ................................................................................................................ 281
Exit 21 -- SYSOUTL Report Termination Exit .................................................................. 282
Exit 22 -- Report Security and Routing Exit ..................................................................... 283
Exit 23 -- Pre-BIND RUN UNIT Exit ................................................................................ 284
Exit 24 -- GET TIME Exit .................................................................................................. 285
Exit 26 -- OLO JCL exit ...................................................................................................... 285
Exit 27 -- Examining the ERE Extension Exit ................................................................... 286
Extended Addressing and Multitasking ........................................ 298

Extended Addressing Considerations ........................................ 298
  AMODE and RMODE Assignment ............................................. 299
  Program Pool Usage ............................................................. 299
  Storage Pool Usage .............................................................. 301
  Treatment of Dynamically-built Control Blocks ....................... 301

Multitasking Support ................................................................ 301
  How DC Handles Concurrency ................................................. 302
  How to Implement z/OS Multitasking ....................................... 304
  Coding Considerations ......................................................... 305
  Monitoring Multitasking Performance ..................................... 306
  Multitasking Information in Dumps .......................................... 307
    Locating the SCA .............................................................. 307
    SCA Information .............................................................. 307
    Determining the Task Using a Subtask .................................. 308
    Determining the Task’s MPMODE .......................................... 308
    Determining the Locks Held by a Task .................................. 308
    Routing System Snaps to a Sequential File ............................ 309
Statistics ................................................................. 310
  System-Wide Statistics .................................................. 311
    When System-Wide Statistics are Written ......................... 311
  Categories of Statistics ............................................. 312
  Task and External Request Unit Statistics ........................ 319
    Enabling Statistics Collection ..................................... 319
    How Statistics Accumulate at Runtime ......................... 320
      DC/UCF Statistics .............................................. 321
      Non-SQL DB Statistics ........................................ 322
      SQL DB Statistics ............................................. 323
  Transaction Statistics .............................................. 323
    Enabling Statistics Collection ..................................... 324
    How Statistics Accumulate at Runtime ......................... 325
  CA ADS Dialog Statistics (IDMS) .................................. 326
    Enabling Statistics Collection ..................................... 326
    How Statistics Accumulate at Runtime ......................... 327
    Categories of CA ADS Statistics ............................... 328
  Histograms ............................................................ 330
    Overview of Histograms ............................................ 331
    How Histograms Accumulate at Runtime ......................... 332
    System-wide Histograms ......................................... 332
    Task Histograms .................................................. 333
    Line Histograms ................................................... 334
    Overriding Histogram Defaults .................................. 335
      Coding HSTDEF Macros ....................................... 335
      HSTDEF Syntax ............................................... 335
      HSTDEF Parameters ........................................... 335
      HSTDEF Examples ............................................. 336
      Creating the RHDCIST Module ................................ 336
  User-Written Statistics Support .................................. 337
  Examining Statistics .............................................. 338
    Current Runtime Statistics ...................................... 339
    Logged Statistics ................................................ 339

System Performance .................................................. 340
  Database Operations .............................................. 341
    Waiting on Full Journal Message ................................ 342
  Deadlock Detector Performance Management .................... 343
  Program Loading .................................................... 343
General Strategies ................................................................. 344
Defining Load Lists ................................................................. 345
System Resource Management .................................................. 346
  Task Resource and Deadlock Management .............................. 346
  Resource Limits ................................................................. 347
Tasks ....................................................................................... 348
Terminal Exception Response Protocol ....................................... 349
File Cache in Memory .............................................................. 350
  Terminology .................................................................... 350
  Exploiting File Cache in Memory ...................................... 350
Parallel Access Volume Exploitation ........................................... 351
PDSE Support ....................................................................... 352
Journaling Performance .......................................................... 353
Improving Recovery Performance .............................................. 353

System Tracing and the System Log ......................................... 353
Enhanced System Tracing ........................................................ 354
  Trace Tables and Entry Types ............................................ 355
  Saving Trace Data ............................................................ 355
  Specify Trace Options ...................................................... 356
    How to Reduce the Number of Missed Entries .................. 356
    Printing and Archiving Trace Information ....................... 356
Maintaining the System Log ..................................................... 358
  Database Log (DDLDCLOG Area) ...................................... 358
  Sequential Log Files ........................................................ 359
Accessing Logged Information ................................................. 360
How the System Logs Errors ................................................... 361

Applying Optional Functionality .............................................. 362
Creating an RHDCOPTF Module .............................................. 363
  DEFOPTF Macro ............................................................. 363
  Example .......................................................................... 363

CA IDMS in a Sysplex Environment ........................................... 363
Using Shared Cache ............................................................... 364
  About Shared Cache ........................................................ 364
  About Shared Cache Examples ........................................ 367
Overriding z/VSE File Specifications at Runtime ................................................................. 468
Implementing z/VSE Job Accounting Support ........................................................................ 469
SVA-eligible Nucleus Modules ............................................................................................. 469

Sample z/VSE JCL ................................................................................................................ 470
Library, Dataset, and File Name References in z/VSE JCL ....................................................... 470
z/VSE Assemble JCL ................................................................................................................. 476
z/VSE Link JCL .......................................................................................................................... 476
z/VSE DC/UCF Startup JCL ...................................................................................................... 476

DC/UCF Test Environment ..................................................................................................... 477
DC/UCF Test EnvironmentOverview ..................................................................................... 477
Programs and the Test Environment ...................................................................................... 478
General Considerations .......................................................................................................... 479
Programs in the Load Areas .................................................................................................... 479
Programs in Load (core-image) Libraries ............................................................................... 480
Test Environment at Runtime ................................................................................................ 481
Load Lists ................................................................................................................................ 481
SYSLOAD Load List ................................................................................................................. 481
Example of Test Environment Execution ............................................................................... 482

Simulating 3270-Type Terminals ........................................................................................... 484
Online Simulator ....................................................................................................................... 485
Batch Simulator ....................................................................................................................... 486
Starting and Ending a Simulator Session ............................................................................... 486
Facsimile Screens ..................................................................................................................... 486
Composing Input Lines ............................................................................................................ 487
General Instructions ................................................................................................................ 488
Simulating Control Keys ........................................................................................................ 489
Changing the Simulator Environment .................................................................................... 491
Additional Batch Simulator Topics .......................................................................................... 491
System Configuration .............................................................................................................. 491
System Generation Statements ............................................................................................... 492
Examples .................................................................................................................................. 492
Configurations for Multiple Lines and Terminals ................................................................. 493
Control Commands ............................................................................................................... 493
MAXTERM ........................................................................................................................... 494
Administrating CA IDMS System Operations

When you install CA™ IDMS, the software delivery is streamlined to install all CA IDMS product component options. This process makes it easier to do the following activities:

- Install and maintain products with fewer choices and a simplified process.
- Install the software once and use different mixes of products in different environments.
- Try a new product.

Legal use of CA products is subject to your license agreement and verified through LMP (License Management Product) key checking. If a required LMP key is not installed, any attempt to use the associated product results in warning messages.
Product Intent Module

You can proactively prevent unauthorized product use and the consequent LMP warnings by declaring your intended product use. A product intent module is a list of the products that are intended to be used in a given environment; you can assemble one and place it in your custom load library.

If you do not have a product intent module, all products are treated as if they are intended for use. When you have a product intent module only the identified products are eligible for use. An attempt to use a product that is not identified by your product intent module results in a task failure. The only exception to this rule is when the SQL Web Connect feature is employed for the IDMS Server and/or IDMS SQL Option products. For more information, see Using SQL Web Connect (see page 439).

For more information, see the following topics:
- Create a Product Intent Module (see page 22)
- Display Product Intent (see page 24)
- Change the Product Intent (see page 24)

Create a Product Intent Module

The easiest way to create a product intent module is to begin with the sample RHDCPINT source module installed with CA IDMS. After creating the product intent module, you need to assemble and link the module into your custom load library.

To create a product intent module and link it to your custom load library

1. Access the sample RHDCPINT module.
2. Uncomment the lines for the products you intend to use.
3. Save it to your custom source library.
4. Execute the z/OS Assemble and Link-edit JCL (see page 465). Substitute the name of your source member and insert the following binder statements:

   ENTRY PINTEP1
   NAME RHDCPINT(R)

Example Product Intent Module Creation

The example product intent module authorizes the use of these CA products:

- CA IDMS/DB
- CA IDMS Server
• CA IDMS/DC
• CA IDMS Performance Monitor
• CA IDMS Presspack
• CA ADS
• CA IDMS SQL
• CA IDMS Culprit for CA IDMS
• CA IDMS Dictionary Module Editor
• CA IDMS DML Online

TITLE 'RHDCPINT - Product Intent Table'
* RHDCPINT
* CA IDMS Core Products
* ---------------------------------------
  #DEFPINT ADS  CA ADS
  #DEFPINT ADSB  CA ADS Batch
  #DEFPINT APPC  CA ADS APPC
  #DEFPINT CMS  CA IDMS CMS Option
  #DEFPINT DDS  CA IDMS DDS
  #DEFPINT DICTLODR  CA IDMS Dictionary Loader
  #DEFPINT EDPAUDIT  CA EDP Auditor
  #DEFPINT ICMS  CA ICMS
  #DEFPINT IDMSCULP  CA Culprit for CA IDMS
  #DEFPINT IDMSDB  CA IDMS/DB
  #DEFPINT IDMSDC  CA IDMS/DC
  #DEFPINT OLQ  CA OLQ Online Query for CA IDMS
  #DEFPINT OTPMON  CA IDMS/TP Monitor
  #DEFPINT PERFMON  CA IDMS Performance Monitor
  #DEFPINT PRESPACK  CA IDMS Presspack
  #DEFPINT SERVER  CA IDMS Server
  #DEFPINT SQL  CA IDMS SQL
  #DEFPINT UCF  CA IDMS UCF
* CA IDMS Transparency Options
* ---------------------------------------
  #DEFPINT DBOMP/T  CA IDMS DBOMP Transparency
  #DEFPINT DL1/T  CA IDMS DL1 Transparency
  #DEFPINT TOTAL/T  CA IDMS TOTAL Transparency
  #DEFPINT VSAM/T  CA IDMS VSAM Transparency
* CA IDMS Tools Products
* ---------------------------------------
  #DEFPINT ADSALIVE  CA ADS Alive
  #DEFPINT ADSTRACE  CA ADS Trace
  #DEFPINT DBANALYZ  CA IDMS/DB Analyzer
  #DEFPINT DBAUDIT  CA IDMS/DB Audit
  #DEFPINT DBEXTRCT  CA IDMS Extractor
  #DEFPINT DBREORG  CA IDMS/DB Reorg
  #DEFPINT DICTMIGR  CA IDMS Dictionary Migrator
  #DEFPINT DML0  CA IDMS DML Online
  #DEFPINT DME  CA IDMS Dictionary Module Editor
  #DEFPINT DJF  CA IDMS Dictionary Query Facility
  #DEFPINT ENFORCER  CA IDMS Enforcer
  #DEFPINT JRNLANLZ  CA IDMS Journal Analyzer
  #DEFPINT LOGANALZ  CA IDMS Log Analyzer
  #DEFPINT MSTRKEY  CA IDMS Masterkey
  #DEFPINT ONLINLOG  CA IDMS Online Log Display
  #DEFPINT SASO  CA IDMS SASO
  #DEFPINT SCHEMAPR  CA IDMS Schema Mapper
  #DEFPINT TPSORT  CA IDMS/DC Sort
  #DEFPINT TSKANALZ  CA IDMS Task Analyzer
* CA Endevor/DB for IDMS
* ---------------------------------------
Display Product Intent

The DCPROFIL system task shows which products you intend to use.

<table>
<thead>
<tr>
<th>CA IDMS Core Products</th>
<th>* Product Intent Status *</th>
<th>CA IDMS Tools Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA ADS</td>
<td>YES</td>
<td>CA ADS Alive</td>
</tr>
<tr>
<td>CA ADS Batch</td>
<td>NO</td>
<td>CA ADS Trace</td>
</tr>
<tr>
<td>CA ADS APPC</td>
<td>NO</td>
<td>CA IDMS/DB Analyzer</td>
</tr>
<tr>
<td>CA IDMS CMS Option</td>
<td>NO</td>
<td>CA IDMS/DB Audit</td>
</tr>
<tr>
<td>CA IDMS DDS</td>
<td>YES</td>
<td>CA IDMS/DB Extractor</td>
</tr>
<tr>
<td>CA IDMS Dictionary Loader</td>
<td>NO</td>
<td>CA IDMS/DB Reorg</td>
</tr>
<tr>
<td>CA EDP Auditor</td>
<td>NO</td>
<td>CA IDMS Dictionary Migrator</td>
</tr>
<tr>
<td>CA ICMS</td>
<td>NO</td>
<td>CA IDMS DML Online</td>
</tr>
<tr>
<td>CA CULPRIT for CA IDMS</td>
<td>NO</td>
<td>CA IDMS Dictionary Module Editor</td>
</tr>
<tr>
<td>CA IDMS/DB</td>
<td>YES</td>
<td>CA IDMS Dictionary Query Facility</td>
</tr>
<tr>
<td>CA IDMS/DC</td>
<td>YES</td>
<td>CA IDMS Enforcer</td>
</tr>
<tr>
<td>CA OLO</td>
<td>NO</td>
<td>CA IDMS Journal Analyzer</td>
</tr>
<tr>
<td>CA IDMS/TP Monitor</td>
<td>NO</td>
<td>CA IDMS Log Analyzer</td>
</tr>
<tr>
<td>CA IDMS Performance Monitor</td>
<td>YES</td>
<td>CA IDMS Masterkey</td>
</tr>
<tr>
<td>CA IDMS Presspack</td>
<td>YES</td>
<td>CA IDMS Online Log Display</td>
</tr>
<tr>
<td>CA IDMS Server</td>
<td>YES</td>
<td>CA IDMS SASO</td>
</tr>
<tr>
<td>CA IDMS SOL</td>
<td>YES</td>
<td>CA IDMS Schema Mapper</td>
</tr>
<tr>
<td>CA IDMS UCF</td>
<td>YES</td>
<td>CA IDMS/DC Sort</td>
</tr>
<tr>
<td>CA IDMS Transparency Options</td>
<td>CAN-endevor/db for idms</td>
<td>CA IDMS Task Analyzer</td>
</tr>
</tbody>
</table>

Change the Product Intent

To change your intended product use, you need to create a new product intent module and use the DCMT VARY NUCLEUS command to make the new RHDCPINT module available in each of your CA IDMS systems.

Starting Up a DC/UCF System

Before you can start up a DC/UCF system, you need to perform the following procedures:

- Define and Generate the DC/UCF System (see page 25)
- Define Dictionaries and Databases (see page 25)
- Execute the DC/UCF Startup Routine (see page 26)
Define and Generate the DC/UCF System

You define a DC/UCF system by using the DC/UCF system generation compiler.

You can define a system by copying and modifying an existing system (for example, installation system 90). Alternatively, you can define a totally new system.

Note: For detailed information about defining DC/UCF systems, see the System Generation Guide.

Define Dictionaries and Databases

Data dictionaries and databases are defined by using the following CA IDMS supplied compilers:

- Non-SQL schema and subschema definition
- SQL database definition
- Physical database definition

The following table lists the dictionary and database areas that you include in the DMCL module used by the DC/UCF system under the central version:

<table>
<thead>
<tr>
<th>Area</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDL</td>
<td>Contains definitions of DC/UCF systems, maps, dialogs, source modules, and records. Each DML dictionary must have its own DDLDML area.</td>
</tr>
<tr>
<td>DDL</td>
<td>Contains load modules associated with entities contained in the DDLDML area; for example, map load modules and subschema load modules.</td>
</tr>
<tr>
<td>DCL</td>
<td>Each dictionary that requires a unique load area must have its own DDLDCLOD area.</td>
</tr>
<tr>
<td>CAT</td>
<td>Contains definitions of physical databases accessible from this runtime environment (for example, segments, DMCLs, database name tables). At sites with the SQL option, contains definitions of SQL entities (for example, tables, constraints, indexes).</td>
</tr>
<tr>
<td>CATX</td>
<td>Contains indexes defined on entities stored in the DDLCAT area. Each DDLCAT area must have its own DDLCATX area.</td>
</tr>
<tr>
<td>CATL</td>
<td>Contains load modules associated with entities contained in the DDLCAT area; for example DMCL load modules, database name table load modules, and access modules at sites with the SQL option. Each DDLCAT area must have its own DDLCATLOD area.</td>
</tr>
<tr>
<td>DOL</td>
<td>Stores messages for use at runtime. All dictionaries in a DC/UCF system share the same DDLDCMSG area.</td>
</tr>
<tr>
<td>DCM</td>
<td>Contains runtime queue information used by CA-supplied tools and online user programs.</td>
</tr>
<tr>
<td>SG</td>
<td></td>
</tr>
<tr>
<td>DCR</td>
<td></td>
</tr>
<tr>
<td>UN</td>
<td></td>
</tr>
</tbody>
</table>
### Execute the DC/UCF Startup Routine

You need to execute the DC/UCF startup routines for the z/OS, z/VSE, and z/VM environments.

- **More:**
  - System Startup Under z/OS
  - System Startup Under z/VSE
  - System Startup Under z/VM

### System Startup Under z/OS

IDMS CVs and DC/UCF systems share a common CA-delivered startup module: RHDCOMVS in z/OS.

Installation assigns the aliases IDMSDC and IDMSCV to RHDCOMVS. You can refer to the startup module in your EXEC statement using either alias (IDMSDC or IDMSCV) or the standard name (RHDCOMVS). However, to attach a CV in z/OS, you must use the name IDMSCV to invoke it.

The PARM statement in JCL specifies the Startup System number and other options. These options are documented in *Specifying Runtime Options (see page 508)*.

WTO and WTOR user exits that are identified by name, such as WTOEXIT, are either enabled using the PARM field in your startup JCL or are linked with IDMSUXIT. For more information on named user exits, see *IDMS User Exits (see page 205)*.
Execute the Startup Routine using JCL

You need to execute the startup routine using JCL modeled after the sample JCL in the z/OS DC/UCF Startup JCL (see page 466).

The following example illustrates how to identify the target system (in this case system version 400) and the DMCL (DMCL400) at runtime:

```
//STARTUP EXEC PGM=IDMSDC,PARM='SYSTEM=400,DMCL=DMCL400'
```

In addition to identifying the target system and DMCL, the EXEC statement PARM parameter can also be used to specify many other runtime options, such as whether multitasking is in effect.

⚠️ **Note:** For a complete description of the execution options that you can specify, see the section Specify Runtime Options (see page 45).

JCL to execute the DC/UCF startup routine must include definitions of all database files that are not dynamically allocated.

⚠️ **Note:** For more information on dynamic file allocation, see the Database Administering section.

The sample z/OS JCL lists all the files used in system startup. These files can be allocated dynamically by specifying data set names as part of their definition.

System Startup Under z/VM

**Contents**
- Step 1 Assemble the DCPARM Macro (see page 28)
- Step 2 Assemble the SVCOPT Macro (see page 28)
- Step 3 Link Edit the Startup Routine (see page 28)
- Step 4 Execute the startup routine (see page 30)

To start up DC/UCF in a z/VM virtual machine, perform the following steps:

1. Optionally assemble the #DCPARM macro object.
2. Optionally assemble the #SVCOPT macro object.
3. Optionally link edit the startup routine.
4. Execute the startup routine.

⚠️ **Note:** For more information on z/VM systems, see the *Installing section -- z/VM.*

---

**Step 1 Assemble the DCPARM Macro**

The #DCPARM macro can be used to specify basic information on the DC/UCF system to be started. Its use is optional. For more information and a description of the #DCPARM syntax, see Coding a #DCPARM Macro. To assemble the #DCPARM macro in z/VM, use the following commands:

```
z/VM RHDCPARM
GLOBAL MACLIB idmslib opsys.maclib(s)
FILEDEF TEXT DISK RHDCPARM TEXT A
ASSEMBLE dcparm (print noterm object
dcparm filename of the file that contains the #DCPARM macro statement
idmslib filename of the CA IDMS MACLIB library
opsys.maclib z/VM macro libraries.
   DMSGPI DMSOM OSMACRO OSMACRO1
```

---

**Step 2 Assemble the SVCOPT Macro**

The #SVCOPT macro defines the type of z/VM environment that will be used. The following example was created during the base installation and is called SVCOPT ASSEMBLE:

```
z/VM #SVCOPT
GLOBAL MACLIB idmslib opsys.maclib(s)
FILEDEF text &numsign.SVCOPT TEXT A
ASSEMBLE svcopt (print noterm object
idmslib filename of the CA IDMS MACLIB library
opsys.maclib z/VM macro libraries.
   DMSGPI DMSOM OSMACRO OSMACRO1
svcopt filename of the file that contains the #SVCOPT macro statement
```

---

**Step 3 Link Edit the Startup Routine**

This step is required only under one or more of the following conditions:

- A #DCPARM macro was assembled in step 1.
- A #SVCOPT macro was assembled in step 2.
You want to hard link a WTOEXIT or WTOREXIT module with your startup routine instead of specifying its name through an execution parameter at runtime.

If none of these conditions necessitate linking a startup routine, you can use the startup module called IDMSDC created during installation.

To link edit the DC/UCF startup routine, use the following commands:

**z/VM DC/UCF startup routine**

```
FILEDEF SYSLST PRINTER
FILEDEF dbalib DISK dbalib LOADDR LIB a (RECFM V LRECL 1024 BLKSIZE 1024
FILEDEF idmslib DISK idmslib LOADDR LIB a (RECFM V LRECL 1024 BLKSIZE 1024
LKED linkctl

Linkage editor control statements (linkctl):

INCLUDE wtoexit            << User-write-to-operator exit - optional
INCLUDE wtoexit            << User-write-to-operator-reply exit - optional
INCLUDE rhdcparm          << #DCPARM macro assembly - optional
INCLUDE usvcopt           << #SVCOPT macro assembly - required
INCLUDE RHDCOCMS
INCLUDE CA$IOS
INCLUDE IDMSUSVC
INCLUDE IDMSUSVM
INCLUDE IDMSCMSO
INCLUDE RHDCOWP
INCLUDE RHDCOCOC
INCLUDE RHDCOPTS
INCLUDE RHDCACHE
ENTRY STARTUP
NAME dcucfsys(R)
```

**wtoexit** name assigned to the WTO exit module

**Note:** A WTO exit module called WTOEXIT is supplied during installation..note off

**wtorexit** name assigned to the WTOR exit module

**rhdcparm** name assigned to the #DCPARM module

**Note:** During installation, a module called RHDCPARM is created by assembling #DCPARM..note off

**usvcopt** name assigned to the #SVCOPT module

**Note:** During installation, a module called USVCOPT is created by assembling #SVCOPT..note off

**dcucfsys** name assigned to the DC/UCF startup routine

**dbalib** ddname of the CA IDMS load library containing the DMCL and database name table load modules

**dbalib LOADLIB a6** file ID of the CA IDMS load library containing the DMCL and database name table load modules

**idmslib** ddname of the CA IDMS load library

**idmslib loadlib a6** file ID of the CA IDMS load library

**linkctl** filename of the file that contains linkage editor control statements
Step 4 Execute the startup routine

Commands to execute the DC/UCF startup routine must include definitions of all user database files and the terminal network. A sample EXEC is delivered as part of the installation process as STARTUP EXEC.

The program that is executed is either the startup routine that you created in the preceding step or the IDMSDC module residing in the installation load library. If executing the installed IDMSDC module, you must identify the DMCL and target DC/UCF system using an OSRUN command PARM parameter. If executing a startup routine that you created, you can use the PARM parameter to override the information that was specified in the #DCPARM macro. The following example illustrates how to identify the target system (in this case, system version 400) and the DMCL (DMCL400) at runtime:

"OSRUN IDMSDC PARM='SYSTEM=400,DMCL=DMCL400'"

Note: For a complete description of the execution options that can be specified, see Specifying Runtime Options.

System Startup Under z/VSE

Contents
- Step 1 Code DVFILE Macros (see page 31)
  - Format 1 DVFILE Macro (see page 34)
  - Format 1 DVFILE Macro Syntax (see page 34)
  - Format 1 DVFILE Macro Parameter (see page 35)
  - Format 2 DVFILE Macro (see page 40)
  - Format 2 DVFILE Syntax (see page 40)
  - Format 2 DVFILE Parameter (see page 41)
  - Format 3 DVFILE Macro (see page 44)
- Step 2 Create an RHDCFTAB Module (see page 44)
- Step 3 Execute the Startup Routine (see page 44)
  - The IDMSLBLS Procedure (see page 45)
  - Specify Runtime Options (see page 45)

To start up DC/UCF on a z/VSE system, perform the following steps:

1. Code #DVFILE macros to describe device-dependent characteristics of sequential files.
2. Create an RHDCFTAB phase to make #DVFILE macro definitions available to the system.
3. Execute the startup routine.
Step 1 Code DVFILE Macros

You code #DVFILE macros to describe sequential files to a z/VSE system. Sample file definitions for a z/VSE system follow. You need to code a #DVFILE macro for each of the following types of files:

- Batch simulator files for use by the DC/UCF batch simulator. At system generation time, a batch simulator file is defined by a LINE statement that specifies the following:
  - TYPE IS S3270Q
  - INPUT DDNAME IS filename
  - OUTPUT DDNAME IS filename

- SYSIN/SYSOUT files for use in handling input and output data. At system generation time, a SYSIN/SYSOUT file is defined by a LINE statement that specifies the following:
  - TYPE INOUTL
  - INPUT DDNAME filename
  - OUTPUT DDNAME filename

- Sequential log files for all log files assigned to sequential files or devices. At system generation time, use of a sequential log file is determined by the LOG parameter of the SYSTEM statement.

The following table describes the three formats for the #DVFILE macro. Format 1 is recommended for all files for which a DTF does not need to be manually coded. Syntax for formats 1, 2, and 3 of #DVFILE are presented below, after the sample z/VSE file definitions.

Depending on the #DVFILE format used for a file, you also may need to code an IBM DTF (define-the-file) macro for the file. Cases where a DTF macro must be manually coded are noted in this section.

**Format 1**

The following table describes #DVFILE format 1:

<table>
<thead>
<tr>
<th>Files defined</th>
<th>Usage considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch simulator input files</td>
<td>Format 1 is recommended for batch simulator input and SYSIN files. Format 1 #DVFILE:</td>
</tr>
<tr>
<td>SYSIN/SYSOUT files</td>
<td>Automatically generates an IBM DTF (define-the-file) macro for the specified file</td>
</tr>
<tr>
<td></td>
<td>Provides extensive verification</td>
</tr>
</tbody>
</table>

**Format 2**

The following table describes #DVFILE format 2:
Files defined | Usage considerations
---|---
Batch simulator input/output files | Format 2 is recommended when the DTF macro must be coded separately. Format 2 #DVFILE:
Sequential log files | Does not generate a DTF macro for the file
SYSIN/SYSOUT files | Provides limited verification

**Format 3**

The following table describes #DVFILE format 3:

<table>
<thead>
<tr>
<th>Files defined</th>
<th>Usage considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>All files</td>
<td>(#DVFILE END) Signals the end of the source file to be assembled into the RHDCFTAB phase.</td>
</tr>
</tbody>
</table>

**DC/UCF system generation statements**

The log, SYSIN, SYSOUT, and simulator file assignments used at runtime are determined by system generation statements.

**z/VSE DC/UCF system generation statements**

```
*********************************************************************
*                      SEQUENTIAL LOG FILES                         *
*********************************************************************
SYSTEM 88
  LOG FILE1 IS LGDD1 COUNT IS 1000
  LOG FILE2 IS LGDD2.
*********************************************************************
*                       SYSIN/SYSOUT FILES                            *
*********************************************************************
LINE LNA
  TYPE IS INOUTL
  INPUT DDNAME IS SYSIN1
  OUTPUT DDNAME IS SYSOUT1.
PTERM PTA1
  TYPE IS INOUTT
  PAGE LENGTH IS 24
  MAXIMUM ERRORS IS 3.
LTERM LTA1.
*********************************************************************
*                        BATCH SIMULATOR FILES                       *
*********************************************************************
LINE LNB
  TYPE IS S3278Q
  INPUT DDNAME IS SIMIN1
  OUTPUT DDNAME IS SIMOUT1.
PTERM PTB1
  TYPE IS S3278
  MODEL IS 2.
LTERM LTB1.
```

**RHDCFTAB source file**
Files defined in the RHDCFTAB module can be named in DC/UCF system generation statements (see above).

z/VSE RHDCFTAB source file

*********************************************************************
* LOG FILE ASSIGNED TO A PRINTER                                     *
*********************************************************************

#DVFILE FILENAM=LGDD1, DEVTYPE=PR, BLKSIZE=133, DTFNAME=LGDD1
LGDD1 DTFPR BLKSIZE=132, CTLCHR=ASA, DEVADDR=SYS027, DEVICE=1403, IOAREA1=BUFF1, RECFORM=FIXUNB

*********************************************************************
* LOG FILE ASSIGNED TO A TAPE UNIT                                   *
*********************************************************************

#DVFILE FILENAM=LGDD2, DEVTYPE=MT, BLKSIZE=133, DTFNAME=LGDD2
LGDD2 DTFMT DEVADDR=SYS028, IOAREA1=BUFF4, BLKSIZE=133, RECFORM=FIXUNB, TYPEFILE=OUTPUT, FILABL=N0, ERREXT=YES, ERROPT=ERRADDR

*********************************************************************
* LOG FILE ASSIGNED TO A SEQUENTIAL DISK DATA SET                   *
*********************************************************************

#DVFILE FILENAM=LGDD21, DEVADDR=SYS070, DEVTYPE=SD, DEVICE=3350, BLKSIZE=129, RECSIZE=121, RECFORM=FIXBLK, TYPEFILE=OUTPUT
LGDD1 DTFSD BLKSIZE=129, RECSIZE=121, RECFORM=FIXBLK, DEVADDR=SYS070, TYPEFILE=OUTPUT

*********************************************************************
* SYSIN FILE ASSIGNED TO A CARD READER                              *
*********************************************************************

#DVFILE FILENAM=SYSIN1, DEVTYPE=CD, DEVADDR=SYS025, BLKSIZE=80, DEVICE=2501

*********************************************************************
* SYSOUT FILE ASSIGNED TO A PRINTER                                 *
*********************************************************************

#DVFILE FILENAM=SYSOUT1, DEVTYPE=PR, BLKSIZE=137, RECFORM=VARUNB, DTFNAME=SYSOUT1
SYSOUT1 DTFPR BLKSIZE=137, CTLCHR=ASA, IOAREA1=BUFF1, DEVADDR=SYS026, WORKA=YES,
RECFORM=VARUNB
*********************************************************************
* SIMULATOR INPUT FILE ASSIGNED TO A CARD READER *
*********************************************************************
#DVFILE DEVTYPE=CD, FILENAME=SIMIN1, DEVICE=2501, DEVADDR=SYS029,
BLKSIZE=80
*********************************************************************
* SIMULATOR OUTPUT ASSIGNED TO A PRINTER *
*********************************************************************
#DVFILE DEVTYPE=PR, FILENAME=SIMOUT1, DTFNAME=SIMOUT1,
BLKSIZE=133
SIMOUT1 DTFPR BLKSIZE=133, CTLCHR=ASA, IOAREA1=BUFF6
*********************************************************************
* END OF RHDCFTAB *
*********************************************************************

Format 1 DVFILE Macro

Format 1 #DVFILE macros are recommended when you define batch simulator input files and SYSIN files. This format automatically generates a DTF macro for the specified file.

The following table provides considerations for coding format 1 #DVFILE macros:

<table>
<thead>
<tr>
<th>File type</th>
<th>#DVFILE definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch simulator input files</td>
<td>Define as input files with 80-character, fixed-length records. For example: BLKSIZE=80,TYPEFILE=INPUT,RECFORM=FIXUNB</td>
</tr>
<tr>
<td>SYISN files</td>
<td>Define as output files with 133-character, variable-length records. For variable length, add 4 bytes to the blocksize (in this case, 133 + 4 = 137). For example: BLKSIZE=137,RECFORM=VARUNB,TYPEFILE=OUTPUT</td>
</tr>
</tbody>
</table>

For SYISN files assigned to printers and any other files requiring special DTF parameters, use format 2 of the #DVFILE macro (described later in this section).

Format 1 DVFILE Macro Syntax

#DVFILE FILENAM=filename
 ,DEVTYPE= CD
 ,DEVADDR= SYSnnn
 ,DI
 ,MT
 ,PR
 ,SD
 ,SYISIPT
 ,SYSLST
 ,SYSPCH
 ,SYSRDR
 ,SYSLOG
CA IDMS - 19.0

,BLKSIZE=block-size
,RECSIZE=record-size
,DEVICE=device-number

,TYPEFILE= INPUT
       OUTPUT
,RECFORM= FIXBLK
       FIXUNB

,FILABL= NO
       STD
       NSTD

,LABADDR=label-address

,REWIND= UNLOAD
       NORWD

Format 1 DVFILE Macro Parameter

- **FILENAME**
  Specifies the name of the file. The specified file name in the #DVFILE macro must be identical to the file name specified for the file at system generation time.

- **DEVTYPE**
  Specifies one of the following device types for the named file:
  - CD -- Card reader/punch
  - DI -- Device independent (system logical unit)
  - MT -- Magnetic tape
  - PR -- Printer
  - SD -- Sequential disk

**Considerations:** Valid values for remaining clauses depend on the device type specified for DEVTYPE. Valid #DVFILE specifications for remaining clauses are presented following these parameter descriptions.

- **DEVADDR**
  Specifies one of the following symbolic units to be assigned to the named file:
  - SYSnnn -- For variable devices, assigned at runtime in JCL for the job
  - SYSIPT -- For card reader devices
  - SYSLST -- For print devices
SYSPCH -- For card punch devices
SYSRDR -- For card reader devices (job control)
SYSLOG -- For terminal devices (operator communication)

**BLKSIZE**
Specifies the block size, in bytes, for the named file. Not valid for DI devices. Required for the following devices:

- CD
- MT
- PR
- SD

**RECSIZE**
Specifies the record size, in bytes, for the named file. Required for the following:

- DI devices
- MT fixed-block files
- SD fixed-block files

**DEVICE**
Specifies the device on which the named file is located. Not valid for MT and DI device types. For valid device-number values, see the appropriate operating system supervisor and I/O macro documentation. Required for the following devices:

- CD
- PR
- SD

**TYPEFLE**
Specifies whether the named file is an input file or an output file as follows:

- CD devices: INPUT or OUTPUT
- MT devices: INPUT
- PR devices: OUTPUT or INPUT
- SD devices: INPUT or OUTPUT

⚠️ **Note:** This parameter is not valid for DI devices.
• **RECFORM**  
  Specifies one of the following record formats for the named file as follows:
  
  - FIXBLK -- Fixed blocked
  
  - FIXUNB -- Fixed unblocked

  **Note:** This parameter is not valid for DI devices.

• **FILABL**  
  (MT devices) Specifies one of the following label types for a tape file:
  
  - NO -- No labels.
  
  - STD -- Standard labels.
  
  - NSTD -- Nonstandard labels. You also must specify the LABADDR parameter (described below).

• **LABADDR**  
  (MT and SD devices) Specifies the entry point name of the routine used to process user labels.

• **REWIND**  
  (MT devices) Specifies the disposition of the tape file as follows:
  
  - UNLOAD -- The tape is rewound and unloaded.
  
  - NORWD -- The tape is not rewound.

  If the REWIND parameter is omitted, the tape is rewound but not unloaded.

**Valid values for CD**

The following table lists valid #DVFILE specifications for the CD type:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVADDR</td>
<td>SYSnnn</td>
</tr>
<tr>
<td></td>
<td>SYSIPT</td>
</tr>
<tr>
<td></td>
<td>SYSPCH</td>
</tr>
<tr>
<td></td>
<td>SYSRDR</td>
</tr>
<tr>
<td>BLKSIZE</td>
<td>80</td>
</tr>
<tr>
<td>RECSIZE</td>
<td>--</td>
</tr>
<tr>
<td>DEVICE</td>
<td>A device number is required</td>
</tr>
<tr>
<td>TYPEFLE</td>
<td>INPUT or OUTPUT</td>
</tr>
<tr>
<td>RECFORM</td>
<td>FIXUNB</td>
</tr>
<tr>
<td>FILABL</td>
<td>--</td>
</tr>
</tbody>
</table>
### Valid values for DI

The following table lists valid DVFILE specifications for the DI type:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVADDR</td>
<td>SYSIPT, SYSLST, SYSPCH, SYSRDR</td>
</tr>
<tr>
<td>BLKSIZE</td>
<td>--</td>
</tr>
</tbody>
</table>
| RECSIZE   | 80 for SYSOPT, SYSRDR units  
            81 for SYSPCH units  
            121 for SYSLST units |
| DEVICE    | -- |
| TYPEFILE  | -- |
| RECFORM   | -- |
| FILABL    | -- |
| LABADDR   | -- |
| REWIND    | -- |

### Valid values for MT

The following table lists valid DVFILE specifications for the MT type:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVADDR</td>
<td>SYSnnn, SYSIPT, SYSLST, SYSPCH, SYSRDR</td>
</tr>
</tbody>
</table>
| BLKSIZE   | Input files:  
            80  
            A multiple of 80 for fixed-block files  
            Output files:  
            121  
            A multiple of 121 for fixed-block files |
| RECSIZE   | 80 for input files  
            121 for output files |
<p>| DEVICE    | -- |
| TYPEFILE  | INPUT or OUTPUT |
| RECFORM   | FIXUNB or FIXBLK |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILABL</td>
<td>NO, STD, or</td>
</tr>
<tr>
<td>LABADDR</td>
<td>Entry point name</td>
</tr>
<tr>
<td>REWIND</td>
<td>UNLOAD or NORWD</td>
</tr>
</tbody>
</table>

**Valid values for PR**

The following table lists valid `#DVFILE` specifications for the PR type:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVADDR</td>
<td>SYSnnn</td>
</tr>
<tr>
<td></td>
<td>SYSLST</td>
</tr>
<tr>
<td></td>
<td>SYSLOG</td>
</tr>
<tr>
<td>BLKSIZE</td>
<td>121</td>
</tr>
<tr>
<td>RECSIZE</td>
<td>--</td>
</tr>
<tr>
<td>DEVICE</td>
<td>A device number is required</td>
</tr>
<tr>
<td>TYPEFILE</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>RECFORM</td>
<td>FIXUNB</td>
</tr>
<tr>
<td>FILABL</td>
<td>--</td>
</tr>
<tr>
<td>LABADDR</td>
<td>--</td>
</tr>
<tr>
<td>REWIND</td>
<td>--</td>
</tr>
</tbody>
</table>

**Valid values for SD**

The following table lists valid `#DVFILE` specifications for the PR type:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVADDR</td>
<td>SYSnnn</td>
</tr>
<tr>
<td>BLKSIZE</td>
<td>Input files:</td>
</tr>
<tr>
<td></td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>A multiple of 80 for fixed-block files</td>
</tr>
<tr>
<td></td>
<td>Output files:</td>
</tr>
<tr>
<td></td>
<td>129</td>
</tr>
<tr>
<td></td>
<td>8 plus a multiple of 121 for fixed-block files</td>
</tr>
<tr>
<td>RECSIZE</td>
<td>80 for input files</td>
</tr>
<tr>
<td></td>
<td>121 for output files</td>
</tr>
<tr>
<td>DEVICE</td>
<td>A device number is required</td>
</tr>
<tr>
<td>TYPEFILE</td>
<td>INPUT or OUTPUT</td>
</tr>
<tr>
<td>RECFORM</td>
<td>FIXUNB or FIXBLK</td>
</tr>
<tr>
<td>FILABL</td>
<td>--</td>
</tr>
<tr>
<td>LABADDR</td>
<td>Entry point name</td>
</tr>
<tr>
<td>REWIND</td>
<td>--</td>
</tr>
</tbody>
</table>
Format 2 DVFILE Macro

Format 2 #DVFILE macros must be coded for a file whenever the file's DTF macro must be manually coded. Format 2 is typically used for the following sequential output files:

- Batch simulator output files
- SYSOUT files
- Sequential log files

Format 2 considerations

The following table provides considerations for coding format 2 #DVFILE macros:

<table>
<thead>
<tr>
<th>File type</th>
<th>Macro definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch simulator input files</td>
<td>#DVFILE macro:&lt;br&gt;Define as input files with 80-character, fixed-length records. For example:&lt;br&gt;BLKSIZE=80,TYPEFLE=INPUT,RECFORM=FIXUNB&lt;br&gt;DTF macro:&lt;br&gt;Specify EOFADDR=EORADDR for all input files.</td>
</tr>
<tr>
<td>SYSIN files</td>
<td></td>
</tr>
<tr>
<td>Batch simulator output files</td>
<td>#DVFILE macro:&lt;br&gt;Define as output files with 133-character, fixed-length records. For example:&lt;br&gt;BLKSIZE=133,RECFORM=FIXUNB,TYPEFLE=OUTPUT&lt;br&gt;DTF macro:&lt;br&gt;For files assigned to printers, specify CTLCHR=ASA.</td>
</tr>
<tr>
<td>Sequential log files</td>
<td></td>
</tr>
<tr>
<td>(Define each sequential log file in its own #DVFILE macro)</td>
<td></td>
</tr>
<tr>
<td>SYSOUT files</td>
<td>#DVFILE macro:&lt;br&gt;Define as output files with 133-character, variable-length records. To accommodate variable lengths, add 4 bytes to the blocksize (in this case, 133 + 4 = 137). For example:&lt;br&gt;BLKSIZE=137,RECFORM=VARUNB,TYPEFLE=OUTPUT&lt;br&gt;DTF macro:&lt;br&gt;For files assigned to printers, specify CTLCHR=ASA.</td>
</tr>
</tbody>
</table>

DTF macro considerations

The DTF macro you code for a file must immediately follow the associated #DVFILE macro. The following considerations apply when coding a DTF macro:

- If the file format is fixed blocked or variable blocked/unblocked, the DTF macro must specify WORKA=YES.
- If error exit option ERROPT=ERRADR applies to the file being defined, this specification must be included in the DTF macro for the file. For details, see the the operating system supervisor and I/O macros documentation.

Format 2 DVFILE Syntax

#DVFILE FILENAM=filename
Format 2 DVFILE Parameter

- **FILENAME**
  Specifies the name of the file. The specified file name in the #DVFILE macro must be identical to the file name specified for the file at system generation time.

- **DEVTYPE**
  Specifies one of the following device types for the named file:
  - CD -- Card reader/punch
  - DI -- Device independent (system logical unit)
  - MT -- Magnetic tape
  - PR -- Printer
  - SD -- Sequential disk
  Considerations: Valid values for remaining clauses depend on the device type specified for DEVTYPE. Valid #DVFILE specifications for different device types follow these parameter descriptions.

- **BLKSIZE**
  Specifies the block size, in bytes, for the named file. Not valid for DI devices. For valid block-size values, see operating system supervisor and I/O macro documentation.
  Required for the following devices:
  - CD
  - MT
  - PR
  - SD
- **RECSIZE**
  Specifies the record size, in bytes, for the named file. For valid record-size values, see operating system supervisor and I/O macro documentation.
  Required for the following:

  - DI devices
  - MT fixed-block files
  - SD fixed-block files

- **DTFNAME**
  Specifies the name of the user-coded DTF macro. For DTF macro coding instructions, see operating system supervisor and I/O documentation.

- **TYPEFILE**
  Specifies whether the named file is an input file or an output file as follows:

  - CD devices: INPUT or OUTPUT
  - MT devices: INPUT or OUTPUT
  - PR devices: OUTPUT
  - SD devices: INPUT or OUTPUT

  **Note:** This parameter is not valid for DI devices.

- **RECFORM**
  Specifies one of the following record formats for the named file:

  - **FIXUNB** -- Fixed unblocked.
  - **FIXBLK** -- Fixed blocked. The DTF macro for the file must specify WORKA=YES.
  - **VARUNB** -- Variable unblocked. The DTF macro for the file must specify WORKA=YES.
  - **VARBLK** -- Variable blocked. The DTF macro for the file must specify WORKA=YES.

  **Note:** This parameter is not valid for DI devices.

### Valid values for CD

The following table lists valid #DVFILE specifications for the CD type:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLKSIZE</td>
<td>A block must be specified</td>
</tr>
</tbody>
</table>
RECSIZE
DTFNAME  DTF macro name (when DTF macro is hand-coded)
TYPEFLE  INPUT or OUTPUT
RECFORM  FIXUNB or VARUNB

Valid values for DI

The following table lists valid #DVFILE specifications for the DI type:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLKSIZE</td>
<td>--</td>
</tr>
<tr>
<td>RECSIZE</td>
<td>A block must be specified</td>
</tr>
<tr>
<td>DTFNAME</td>
<td>DTF macro name (when DTF macro is hand-coded)</td>
</tr>
<tr>
<td>TYPEFLE</td>
<td>--</td>
</tr>
<tr>
<td>RECFORM</td>
<td>--</td>
</tr>
</tbody>
</table>

Valid values for MT

The following table lists valid #DVFILE specifications for the MT type:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLKSIZE</td>
<td>A block must be specified</td>
</tr>
<tr>
<td>RECSIZE</td>
<td>A record size must be specified only for fixed un-blocked devices</td>
</tr>
<tr>
<td>DTFNAME</td>
<td>DTF macro name (when DTF macro is hand-coded)</td>
</tr>
<tr>
<td>TYPEFLE</td>
<td>INPUT or OUTPUT</td>
</tr>
<tr>
<td>RECFORM</td>
<td>FIXUNB, FIXBLK, VARUNB, or VARBLK</td>
</tr>
</tbody>
</table>

Valid values for PR

The following table lists valid #DVFILE specifications for the PR type:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLKSIZE</td>
<td>A block must be specified</td>
</tr>
<tr>
<td>RECSIZE</td>
<td>--</td>
</tr>
<tr>
<td>DTFNAME</td>
<td>DTF macro name (when DTF macro is hand-coded)</td>
</tr>
<tr>
<td>TYPEFLE</td>
<td>OUTPUT</td>
</tr>
<tr>
<td>RECFORM</td>
<td>FIXUNB or VARUNB</td>
</tr>
</tbody>
</table>

Valid values for SD

The following table lists valid #DVFILE specifications for the SD type:
### Format 3 DVFILE Macro

Format 3 of the `#DVFILE` macro indicates the end of the `#DVFILE` macro calls. This macro must be the last macro in the source file for RHDCFTAB. `#DVFILE END` does not replace the Assembler END, which must be coded as the last statement in the source file.

```
#DVFILE END
```

### Step 2 Create an RHDCFTAB Module

You assemble and link edit an RHDCFTAB module to a load library to make z/VSE file definitions in `#DVFILE` macros available at runtime.

**To create a z/VSE RHDCFTAB module:**

1. Assemble and catalog the module using the sample JCL in z/VSE Assemble JCL (see page 476). Modify the JCL by substituting the following in place of the Assembler input statements:

   ```
PUNCH 'CATALOG rhdcftab.OBJ REPLACE=YES'
   #DVFILE macros and DTF macros
   END
   ```

2. Link the RHDCFTAB program using the sample JCL in z/VSE Link JCL. Modify the JCL by substituting the following statements in place of the Linkage editor control statements:

   ```
PHASE RHDCFTAB, *
INCLUDE rhdcftab
ENTRY RHDCFTAB

   rhdcftab
   Specifies the name of the object module containing the assembly output.
   ```

### Step 3 Execute the Startup Routine

JCL to execute the DC/UCF startup routine must include definitions of all user database files and the terminal network. To execute the DC/UCF system, use JCL statements based on the sample JCL statements in z/VSE DC/UCF Startup JCL (see page 476).
The IDMSLBS Procedure

IDMSLBS is a procedure provided during a CA IDMS z/VSE installation. It contains file definitions for the following CA IDMS components:

- Dictionaries
- Sample databases
- Disk journal files
- SYSIDMS file
- SYSCTL file

Tailor the IDMSLBS procedure to reflect the filenames and definitions at your site and include it in z/VSE JCL job streams.

Note: For more information on IDMS files, see Library, Dataset, and File Name References in z/VSE JCL (see page 470).

Specify Runtime Options

You can use the PARM parameter of the EXEC statement to specify certain runtime options.

In the following example, the EXEC IDMSDC statement includes a PARM parameter that overrides the DMCL to be used by the DC/UCF system:

```
// EXEC IDMSDC, SIZE=40K, PARM='DMCL=CVDMCL'
```

For a complete description of the runtime options that can be specified, see Specifying Runtime Options.

What Happens During Startup

Contents

- How the Startup Routine Acquires Storage (see page 46)
- How the System is Built (see page 48)
- The DC/UCF Region/Partition Layout (see page 53)

To begin the DC/UCF startup process, execute the DC/UCF startup routine. The startup routine does the following:

1. Acquires storage for the DC/UCF system
2. Builds the system by passing control to a series of startup modules
When these steps are finished, the DC/UCF system is executing. Each of these steps is discussed below, followed by a description of the DC/UCF region/partition layout that is built during the startup procedure.

How the Startup Routine Acquires Storage

To build the DC/UCF system, the startup routine acquires as much storage as possible for the DC/UCF region/partition. Because the operating system requires execution space, the startup routine immediately returns storage space to the operating system. The amount of storage returned to the operating system is determined by the FREESTG runtime option.

The following illustration shows how the startup routine acquires and returns storage. Details on how this is done at startup are provided below for each of the following operating systems:

- **z/OS**
- **z/VSE - GETVIS**
- **z/VM**

1. The startup routine acquires as much contiguous storage as it can.

```
Low storage   DC/UCF startup address
- - - - - - - - - - - -
```

2. The DC/UCF startup routine returns to the operating system the amount of storage specified by FREESTG.

**z/OS systems**

Under z/OS, the DC/UCF startup routine does the following:

1. Acquires contiguous storage in the region by issuing a GETMAIN command. The GETMAIN command acquires as much contiguous storage as possible. The maximum amount of storage that can be obtained is determined by the EXEC statement REGION parameter in the DC/UCF startup JCL.
2. Returns storage to the operating system by issuing a FREEMAIN command. The amount of storage returned is specified in the FREESTG runtime option. An operating system abend due to insufficient memory during startup is most likely the result of specifying insufficient storage in the FREESTG parameter.

3. Allocates storage for the DC/UCF nucleus, control blocks, storage pools, and program pools as specified at system generation.

4. Returns any unused storage to the operating system by issuing a second FREEMAIN command.

**z/VSE systems**

Under z/VSE, the DC/UCF startup routine acquires GETVIS storage. The amount of storage returned to the operating system is given by the FREESTG runtime option.

The startup routine does the following:

- Calculates the size amount of GETVIS storage available in the partition up to the 16 Megabyte address line or to the end of the partition, whichever is less.
- Subtracts the FREESTG value and issues one GETVIS request for the calculated size.
- Calculates the amount of XA storage required and issues one GETVIS request for that amount.
- When startup completes any storage from below the line that is not used is released to the partition.

**z/VM systems**

Under z/VM, the DC/UCF startup routine does the following:

1. Acquires contiguous storage in the virtual machine by issuing a GETMAIN command. The GETMAIN command acquires as much contiguous storage as possible. Make sure that the system does not try to acquire storage that contains shared segments.

2. Returns storage to the operating system by issuing a FREEMAIN command. The amount of storage returned is specified in the FREESTG runtime option or #DCPARM parameter. Be sure to return at least 256K bytes of storage to the operating system. An operating system abend during startup indicates that FREESTG does not reserve enough storage.

3. Allocates storage for the DC/UCF nucleus, control blocks, storage pools, and program pools as specified at system generation.

4. Returns any unused storage to the operating system by issuing a second FREEMAIN command.

**Note:** For more information on defining the z/VM virtual machine in which the DC/UCF system is to execute, see the *Installing section -- z/VM.*
How the System is Built

The DC/UCF system is built based on system generation definitions, runtime options, and operator startup override values. The startup routine coordinates building of the DC/UCF region/partition by passing control to other system modules that do most of the work. The resulting DC/UCF region /partition layout is shown in The DC/UCF Region/Partition Layout.

During the startup process, DC/UCF issues messages to inform the operator of the system's activities. The following output shows a portion of a sample DC/UCF log file that contains startup messages:

090908 09.45.26 IDMS DC013002 V73 T0 ATTACHING DATABASE RESOURCE CONTROLLER
090908 09.45.26 IDMS DC200131 V73 T1 Lock Manager Initialization Complete
090908 09.45.26 IDMS DC200023 V73 T1 CV Change Tracking is not used
090908 09.45.26 IDMS DC200245 V73 T1 Serializing member startup
090908 09.45.26 IDMS DC200023 V73 T1 CA IDMS/DB: 73 Started
090908 09.45.26 IDMS DC329012 V73 T1 Startup resynchronization initiated
090908 09.45.26 IDMS DC013003 V73 T0 OPENING SYSTEM RUN UNITS
090908 09.45.26 IDMS DC013014 V73 T0 ATTACHING TASK FOR SERVICE DRIVER RHDCRUSD
090908 09.45.26 IDMS DC013014 V73 T0 ATTACHING TASK FOR SERVICE DRIVER RHDCRUSD
090908 09.45.26 IDMS DC013014 V73 T0 ATTACHING TASK FOR SERVICE DRIVER RHDCRUSD
090908 09.45.26 IDMS DC013014 V73 T0 ATTACHING TASK FOR SERVICE DRIVER RHDCRUSD
090908 09.45.26 IDMS DC013014 V73 T0 ATTACHING TASK FOR SERVICE DRIVER RHDCRUSD
090908 09.45.26 IDMS DC013014 V73 T0 ATTACHING TASK FOR SERVICE DRIVER RHDCLGSD
090908 09.45.26 IDMS DC013014 V73 T0 ATTACHING TASK FOR SERVICE DRIVER RHDCLGSD
090908 09.45.26 IDMS DC056001 V73 T0 DLOG IS 00% FULL
090908 09.45.26 IDMS DC013014 V73 T0 ATTACHING TASK FOR SERVICE DRIVER RHDCRUSD
090908 09.45.26 IDMS DC013014 V73 T0 ATTACHING TASK FOR SERVICE DRIVER RHDCRUSD
090908 09.45.26 IDMS DC013014 V73 T0 ATTACHING TASK FOR SERVICE DRIVER IDMSDBSD
090908 09.45.26 IDMS DC205101 V73 T16 IDMSDBSD Started
090908 09.45.26 IDMS DC256004 V73 T0 SECURITY SYSTEM INITIALIZATION BEGINNING
090908 09.45.26 IDMS DC088071 V73 T0 Initialization TCP/IP environment started.
090908 09.45.27 IDMS DC256004 V73 T0 Starting checking status of TCP/IP stack TCPIP31
090908 09.45.27 IDMS DC088025 V73 T0 Status of TCP/IP stack TCPIP31 is active
090908 09.45.27 IDMS DC256002 V73 T0 Initialization TCP/IP environment completed. TCP/IP is active.
090908 09.45.27 IDMS DC256002 V73 T0 *** QUEUE STATUS IN DDDLDCRUN ***
090908 09.45.27 IDMS DC256003 V73 T0 QUEUE NAME INV TASK RECORDS CREATE RET EXP
090908 09.45.27 IDMS DC256004 V73 T0 $ADCTEST 1 0000004 03.178 255 PRM
090908 09.45.27 IDMS DC256004 V73 T0 $ADSCIDX NO
090908 09.45.27 IDMS DC256004 V73 T0 JPD1 NO
090908 09.45.27 IDMS DC256004 V73 T0 JPD1 NO
090908 09.45.27 IDMS DC256004 V73 T0 JPD2 NO
090908 09.45.27 IDMS DC256004 V73 T0 KJMQUE1 NO
090908 09.45.27 IDMS DC256004 V73 T0 KJMQUE2 NO
090908 09.45.27 IDMS DC256004 V73 T0 OLQONOTE OLQONOTE 0000000 0.000 001 000
KEPT       NO
090908 09.45.27 IDMS DC256004 V73 T0 RTSVQ 0000002 07.219 255 PRM
KEPT       NO
090908 09.45.27 IDMS DC256004 V73 T0 TASK_ANALYZER_12 0000001 03.315 255 PRM
KEPT       NO
090908 09.45.28 IDMS DC256015 V73 T0 Queue initialization complete
090908 09.45.28 SYSTEM INITIALIZED ON 08253 AT 09:45:28.01
090908 09.45.28 MAP OF REGION
090908 09.45.28 SCAAREA 000296E0 CCE 0002B280 NLT 000398E0 STGPOOL 0005 4000
090908 09.45.28 KEPT       NO
090908 09.45.28 IDMS DC256004 V73 T0 TASK_ANALYZER_12 0000001 03.315 255 PRM
090908 09.45.28 MAP OF NUCLEUS
090908 09.45.28 MODNAME CSECT ASM DATE LOADADR ENTRY POINTS
090908 09.45.28 IDMSDBIO 39B29000 DBIOEP1 39B29070 DBIOEP2 39B291B0 DBIOEP3 39B29238
090908 09.45.28 IDMSHLDB 39B67000 HLDBEP1 39B67070 HLDBEP2 39B72A08
090908 09.45.28 IDMSDBMS 39B81800 DBMSEP1 39B81870 DBMSEP2 39B81974
090908 09.45.28 IDMSLRBK 39B8D200 LRBKEP1 39B8D270 LRBKEP2 39B8DC20
090908 09.45.28 IDMSAREC 39B89400 ARECEP1 39B89470 ARECEP2 39B89810
090908 09.45.28 IDMSLRF 39B89900 LRFEP1 39B89970 LRFEP2 39B8A120
090908 09.45.28 IDMSEXP 39B8A400 EXPEP1 39B8A470 EXPEP2 39B8B100
090908 09.45.28 IDMSBRBK 39B8B500 BRBKEP1 39B8B570 BRBKEP2 39B8B820
090908 09.45.28 IDMSCONN 39B8B900 CONNEP1 39B8B970 CONNEP2 39B8C100
090908 09.45.28 IDMSAREC 39B8D200 ARECEP1 39B8D270 ARECEP2 39B8D900
090908 09.45.28 IDMSDBMS 39B8E180 CURSEP1 39B8E170 CURSEP2 39B8E1974
090908 09.45.28 IDMSDBIO 39B8F200 DBIOEP1 39B8F270 DBIOEP2 39B8F400
090908 09.45.28 IDMSLRBK 39B8F500 LRBKEP1 39B8F570 LRBKEP2 39B8F820
090908 09.45.28 IDMSAREC 39B8F900 ARECEP1 39B8FA00 ARECEP2 39B8FA70

23-Dec-2017
49/524
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>IDMS Component</th>
<th>Time</th>
<th>Event</th>
<th>IDMS Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>IDMS_CONN</td>
<td>09:09:08</td>
<td>09:45:28</td>
<td>IDMS_CONN</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>IDMS_XTRA</td>
<td>09:09:08</td>
<td>09:45:28</td>
<td>IDMS_XTRA</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>IDMS_BLDLR</td>
<td>09:09:08</td>
<td>09:45:28</td>
<td>IDMS_BLDLR</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_MDMTD</td>
<td>09:08:09</td>
<td>15:21</td>
<td>RHDC_MDMTD</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_RCUAL</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_RCUAL</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_XTRA</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_XTRA</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_BLDR</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_BLDR</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_MODT</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_MODT</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_RUALSE</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_RUALSE</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_CRMGR</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_CRMGR</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_SQL</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_SQL</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_QMEM</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_QMEM</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_MISC</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_MISC</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDCTABL</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDCTABL</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDCCXIT</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDCCXIT</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_SNAP</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_SNAP</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_RNFR</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_RNFR</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_PCTL</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_PCTL</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_LDR</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_LDR</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_TIRH</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_TIRH</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_MAPR</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_MAPR</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_COBI</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_COBI</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_PLIB</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_PLIB</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_IMP</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_IMP</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_WTLE</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_WTLE</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_EDQ</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_EDQ</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_DBCO</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_DBCO</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_MSTR</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_MSTR</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_SKI</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_SKI</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_CALC</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_CALC</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_SVC</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_SVC</td>
</tr>
<tr>
<td>09:09:08</td>
<td>09:45:28</td>
<td>RHDC_TASK</td>
<td>09:05:05</td>
<td>15:18</td>
<td>RHDC_TASK</td>
</tr>
</tbody>
</table>
The following are the events that occur during the DC/UCF startup process:

1. The startup routine acquires storage for the DC/UCF region/partition and then frees a predefined amount of storage to the operating system. This process is described in How the Startup Routine Acquires Storage.

2. The startup routine loads the main startup module (RHDCNTRY) at the end of the acquired region and passes control to the module. RHDCNTRY allocates XA storage according to SYSIDMS parameter CV_STARTUP_XA_REGION_MB and builds the DC/UCF system in the acquired space. If data sharing is in effect, it joins the XCF group and connects to the XES list and lock structures. If necessary, RHDCNTRY performs a warmstart. The DC/UCF system definition is read from the dictionary named SYSTEM as identified by a DBNAME or a segment name. Then RHDCNTRY builds the rest of the DC/UCF system. RHDCNTRY loads the system components sequentially, starting at the low end (the low storage address) of the DC/UCF region/partition.

   RHDCNTRY defines the MASTER task as the first task, T0 (task zero), and then returns control to the startup routine.

3. The startup routine returns any unused space in the originally acquired region/partition to the operating system.

4. The startup routine passes control to the DC/UCF dispatcher (RHDCWAIT).

5. RHDCWAIT dispatches the MASTER task (task T0). At this point, the DC/UCF startup process is complete and DC/UCF system execution is in progress. The MASTER task attaches the Database Resource Controller (DBRC) task, T1, which manages the database. During startup, DBRC does the following:
- Opens the journaling system
- Processes unrecovered transactions
- Processes incomplete (distributed) transactions
- Coordinates data sharing member startup, if applicable
- Initiates operator communications through the console
- Starts up the external request-unit service (ERUS) facility
- Initiates startup resynchronization for distributed transactions
- Posts the MASTER task when initialization is complete

The MASTER task also does the following:

- Initiates a task that erases all expired queues
- Writes a map of the region/partition and a map of the DC/UCF nucleus to the DC/UCF log
- Attaches driver modules
- Attaches the print control task (if applicable).
- Invokes the CLOD task to erase all logically deleted load modules from load area
- Invokes the QUED task. This is done for compatibility only, since processing of expired queues was already done

⚠️ **Note:** You can also invoke the CLOD and QUED tasks after startup, as described in the *CA IDMS System Tasks and Operator Reference section*.

MASTER then attaches any startup autotasks defined at DC/UCF system generation time.

---

**The DC/UCF Region/Partition Layout**

The DC/UCF region/partition layout differs depending on your system’s configuration.

To examine the layout for your DC/UCF system, issue the following DCMT command:

```
DCMT DISPLAY MEMORY MAP
```

⚠️ **Note:** For more information on this command, see the *System Tasks and Operator Reference section*.
System nucleus modules

System nucleus modules are loaded at the beginning of the appropriate program and reentrant pools. The size of a program pool is automatically extended to accommodate these modules. The nucleus load table (NLT) resides in the DC/UCF region/partition to identify the system nucleus modules to the system. To display a map of the DC/UCF nucleus modules, issue the following DCMT command:

DCMT DISPLAY MEMORY NUCLEUS

Note: For more information on this command, see the System Tasks and Operator Reference section.

Most system nucleus modules can be reloaded at runtime, as described in Dynamically Reloading Nucleus Modules.

DC/UCF region components

The following table lists the components of the DC/UCF region/partition in alphabetical order:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abend storage</td>
<td>Storage available for processing runtime abends. The amount of available abend storage is specified by the ABEND STORAGE parameter of the system generation SYSTEM statement.</td>
</tr>
<tr>
<td>CCE</td>
<td>The CA IDMS central control element, which contains information used by CA IDMS. This table is built at startup time from values stored in records in the dictionary by database-related parameters defined on the SYSTEM system generation statement.</td>
</tr>
<tr>
<td>CSA</td>
<td>The common system area (CSA) vector table and system-wide fields (for example, statistics fields).</td>
</tr>
<tr>
<td>CSVCAREA</td>
<td>The system SVC parameter area, which is used by DC/UCF as a general work area.</td>
</tr>
<tr>
<td>DC/UCF startup routine</td>
<td>The routine that begins the DC/UCF system startup process. The module name is determined during the link-edit job that creates the load module. Typical names are DCUCF.SYS, EXECDC, DCUCF.GO, and DCEEXEC. The module contents depend on the operating system in use. For details, see information presented earlier in this section about DC/UCF system startup under your operating system.</td>
</tr>
<tr>
<td>Central version DMCL</td>
<td>The actual DMCL used by the DC/UCF system.</td>
</tr>
<tr>
<td>DCEAREA</td>
<td>The dispatch control element area.</td>
</tr>
<tr>
<td>DDT</td>
<td>The destination definition table, which contains destination definition elements (DDEs) built from values stored in the dictionary by system generation DESTINATION statements.</td>
</tr>
<tr>
<td>DPEAREA</td>
<td>The deadlock prevention element area.</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>ECBLIST</td>
<td>The operating system’s event control block (ECB) list.</td>
</tr>
<tr>
<td>EREAREA</td>
<td>The external request unit area.</td>
</tr>
<tr>
<td>ESE</td>
<td>External service element.</td>
</tr>
<tr>
<td>HIADDR</td>
<td>The high 24-bit storage address of the DC/UCF system after the startup routine returns the amount of storage specified in the FREESTG runtime option.</td>
</tr>
<tr>
<td>IDMSDBIO</td>
<td>The CA IDMS database I/O module.</td>
</tr>
<tr>
<td>IDMSDBMS</td>
<td>The CA IDMS database services module.</td>
</tr>
<tr>
<td>ILEAREA</td>
<td>Internal lock element area.</td>
</tr>
<tr>
<td>Line driver modules</td>
<td>Line driver modules for DC/UCF. These modules are loaded at the beginning of the appropriate program or reentrant pool. The pools are increased automatically to accommodate these modules.</td>
</tr>
<tr>
<td>LKM</td>
<td>Locking control blocks, which are a series of internally managed control blocks, accessible only by the Lock Manager, used to serialize resources such as database key locks. These control blocks are allocated at startup time before the first 24-bit storage pool in a non-XA system and before the first 31-bit storage pool in an XA system. Any additional storage required by the Lock Manager at runtime is allocated from the storage pools. The amount of storage allocated is dependent on the value specified in the SYSLOCKS clause of the system generation SYSTEM statement.</td>
</tr>
<tr>
<td>LTT</td>
<td>The logical terminal definition table. This table contains logical terminal elements (LTEs) for terminals that can execute online applications. This table is built at startup time from values stored in the dictionary by system generation LTERM statements.</td>
</tr>
<tr>
<td>MPMODTB</td>
<td>The MPMODE table.</td>
</tr>
<tr>
<td>NLT</td>
<td>The nucleus load table, which identifies the system nucleus modules to the DC/UCF system.</td>
</tr>
<tr>
<td>Nucleus modules</td>
<td>System nucleus modules that perform general system services, such as program loading and storage management. Nucleus modules are loaded at the beginning of the appropriate program or reentrant pool. The pools are increased automatically to accommodate these modules. To display a map of the nucleus modules, issue a DCMT DISPLAY MEMORY NUCLEUS command.</td>
</tr>
<tr>
<td>OPT</td>
<td>The startup options table, which contains DC/UCF system-wide information. This table is built at startup time from values stored in the dictionary during system generation by the SYSTEM statement. The startup options table also contains the DC/UCF region map.</td>
</tr>
<tr>
<td>PDT</td>
<td>The program definition table, which contains program definition elements (PDEs) for programs in all program pools. This table is built at startup from values stored in the dictionary by system generation PROGRAM statements.</td>
</tr>
<tr>
<td>Program pool (24 bit)</td>
<td>The 24-bit program pool is defined by the PROGRAM POOL parameter of the system generation SYSTEM statement. (XA systems only) This optional program pool is defined by the XA PROGRAM POOL parameter of the system generation SYSTEM statement.</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Program pool (31 bit)</td>
<td>The physical terminal definition table. This table contains information used to support online applications that use physical terminals. This table is built from values stored in the data dictionary by system generation LINE and PTERM statements.</td>
</tr>
<tr>
<td>PTT</td>
<td>The queue definition table, which is built from values stored in the dictionary by system generation QUEUE statements. The QDT contains: Queue definition elements (QDEs) Queue control elements (QCEs) Queue wait elements (QWEs)</td>
</tr>
<tr>
<td>QDT</td>
<td>The resource control area, which contains elements used to control system resources: Resource link elements (RLEs) Resource control elements (RCEs) Deadlock prevention elements (DPEs) Internal lock elements (ILEs)</td>
</tr>
<tr>
<td>RCA</td>
<td>Resource control element area</td>
</tr>
<tr>
<td>RCEAREA</td>
<td>Resource control element area</td>
</tr>
<tr>
<td>Reentrant pool (24 bit)</td>
<td>A pool for 24-bit reentrant programs, which is defined by the REENTRANT POOL parameter of the system generation SYSTEM statement.</td>
</tr>
<tr>
<td>Reentrant pool (31 bit)</td>
<td>(XA systems only) This optional reentrant program pool is defined by the XA REENTRANT POOL parameter of the system generation SYSTEM statement.</td>
</tr>
<tr>
<td>Resident programs</td>
<td>An area that contains programs defined as resident at system generation time by means of PROGRAM statements. Resident programs are loaded at the beginning of the program pools immediately after the nucleus modules.</td>
</tr>
<tr>
<td>RHDCDEAD</td>
<td>The deadlock manager.</td>
</tr>
<tr>
<td>RLEAREA</td>
<td>Resource link element area</td>
</tr>
<tr>
<td>RUA</td>
<td>The internal run-unit allocation table contains the following: A run-unit header (RUH) for each run-unit type A run-unit element (RUE) for each run unit</td>
</tr>
<tr>
<td>SCAAREA</td>
<td>The subtask control area.</td>
</tr>
<tr>
<td>SCT</td>
<td>The storage control table area, which contains storage control elements (SCEs) for all storage pools.</td>
</tr>
<tr>
<td>Service driver modules</td>
<td>Service driver modules for DC/UCF. These include the following: RHDCRUSD (run unit service driver) RHDCLGSD (log service driver) Service driver modules perform journal and log I/O operations for other tasks, thus freeing those tasks to continue execution. If data sharing is in effect, the following additional service drivers are loaded: RHDCCFSD (Coupling Facility service driver)</td>
</tr>
</tbody>
</table>
Component Description

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDMSLMSD</td>
<td>Lock Manager service driver</td>
</tr>
<tr>
<td>IDMSBDSD</td>
<td>DBIO service driver</td>
</tr>
<tr>
<td></td>
<td>These modules are loaded at the beginning of the appropriate program or reentrant pool. The pools are increased automatically to accommodate these modules.</td>
</tr>
<tr>
<td>Storage pools (24 bit)</td>
<td>The primary storage pool is defined by the STORAGE POOL clause of the system generation SYSTEM statement. Additional 24-bit storage pools are optionally defined by system generation STORAGE POOL statements.</td>
</tr>
<tr>
<td>Storage pools (31 bit)</td>
<td>(XA systems only) These optional storage pools are defined by system generation XA STORAGE POOL statements (pools 128 to 254) and the XA STORAGE POOL parameter of the SYSTEM statement (pool 255).</td>
</tr>
<tr>
<td>SVC module</td>
<td>Actual SVC used by the DC/UCF system.</td>
</tr>
<tr>
<td>TCA</td>
<td>The task control area contains the following:</td>
</tr>
<tr>
<td></td>
<td>Dispatch control elements (DCEs)</td>
</tr>
<tr>
<td></td>
<td>Task control elements (TCEs)</td>
</tr>
<tr>
<td></td>
<td>Task statistics area (TSA)</td>
</tr>
<tr>
<td></td>
<td>This table is built from values established by system generation TASK statements.</td>
</tr>
<tr>
<td>TCEAREA</td>
<td>Task control element area.</td>
</tr>
<tr>
<td>TDT</td>
<td>The task definition table. This table contains task definition elements (TDEs) for tasks that can be used to invoke online components (for example, the subschema compiler). This table is built at startup time from values stored in the dictionary by system generation TASK statements.</td>
</tr>
<tr>
<td>TRC</td>
<td>The system trace area, which contains the system trace buffers. The size of this area is defined at system generation time by the SYSTRACE ENTRIES parameter of the SYSTEM statement. Each entry in this area requires 64 bytes.</td>
</tr>
</tbody>
</table>

Dynamically Reloading Nucleus Modules

You can reload DC nucleus modules without having to bring down the entire DC system. For example, you do not need to bring the system down to apply maintenance. You need only apply the changes to the load library and then reload the affected modules.

You can dynamically reload any module with a PDE type of NUCLEUS or DRIVER, *except for* the following modules:

- IDMSDBIO
- IDMSUXIT
- RHDCDBRC
- RHDCMSTR
- RHDCSCRN
• RHDCOS00

If one of the modules being reloaded is a driver module that controls your terminal (for example, a VTAM driver), you are logged off of DC/UCF when the driver module is recycled. You can sign back on after the driver module is reloaded.

Considerations

When dynamically reloading nucleus modules, be sure to:

• Include all interdependent modules in the list of modules to be reloaded. For example, if a nucleus module also requires a patch in the common system area (CSA), your list must include both the modified nucleus module and the RHDCCSA module.

• Bring DC to an inactive state before beginning the reload operation by performing the following steps:

  1. Disable long-running conversational tasks, such as the dynamic system monitor (OPER), the online debugger, and the Realtime Monitor at sites where the CA IDMS/Performance Monitor is installed.

  2. Cancel any long-running jobs that are already active.

The list of modules to be reloaded is associated with the logical terminal element (LTE) on which the DCMT VARY NUCLEUS commands are executed. If you sign off from your terminal, the list is deleted. You can determine whether a program is a nucleus module by examining its PDE type. The PDE type is shown by the DCMT DISPLAY PROGRAM program-name command.

Setting Up Interpartition Communication and the SVC

DC/UCF resides in its own region/partition. Online application programs executed by using DC/UCF execute within the DC/UCF region/partition. Additionally, DC/UCF online compilers (such as the online system generation compiler), development tools (such as ADSA and OLM), and database procedures execute within the DC/UCF region/partition.

Certain functions of the online DC/UCF system require the use of a special SVC which is supplied with DC/UCF software. In order to use these functions, the SVC must be installed from a release level at least as high as the online DC/UCF system. An SVC from a higher release level may be used to access a DC/UCF system at a lower release.

Online applications executing under other DC/UCF systems, batch application programs, and programs executed under TP monitors other than DC execute in regions/partitions external to the DC/UCF region/partition. Since CA IDMS resides in the DC/UCF region/partition, all CA IDMS database requests issued by these programs and all responses must be passed across regions/partitions.

The remainder of this section describes the common communications architecture used to pass database requests from one region/partition to another and provides specific information on the following communications:
Communications Architecture

The DC/UCF communications architecture was designed to separate applications from the communications environment. That is, your application does not need to include communications requirements. For example, the application does not need to designate the node in which a database resides. The architecture is based on the client/server model as follows:

- Client -- Requests information or services
- Server -- Provides information or services

The mandate of this model is that clients and servers must be connected only by a well-defined message protocol. One client or server can have no explicit information on other clients/servers.

**Data request processing:** When an application issues a request for database services, each request is routed through four communications layers on both the client side and the server side, as shown in the following diagram:
**Communication Architecture**

**Communication layers:** The following table describes each layer of the communications architecture:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Services Interface (DSI) layer</td>
<td>Provides a service-specific interface between an application and particular service (for example, SQL database services). The DSI creates a data transfer services block (DTSB) that contains routing information, the name of the target resource, the data, and data format descriptors and forwards the DTSB to the data transfer services layer.</td>
</tr>
<tr>
<td>Data Transfer Services (DTS)</td>
<td>Provides the service-independent application program interface (API) for client/server processing. This layer uses a name server table to determine where to pass the requested information.</td>
</tr>
</tbody>
</table>
| Distributed Node Services (DNS) | Determines how the information is to be sent; that is, using:  
- An SVC (used to communicate between the client and a DC/UCF system within the same operating system image)  
- DDS (used to communicate between DC/UCF systems in different CPUs)  |
| Communications drivers | Send the information across to the following:  
- The SVC  
- The UCF line driver  
- DDS using CCI, TCP/IP, or VTAM |

**The name server table:** The name server table is used by the data transfer services layer to determine where to route information. The table contains information on the location of the database to be accessed. Depending on the environment in which your application is executing, the name of the database to be accessed is provided in the following ways:
Environment | How a database name is provided
--- | ---
DC | Through the NODE and RESOURCE TABLE system generation statements
Batch, CICS, and other TP-monitors | Through the IDMSOPTI module, SYSCTL file, and SYSSIDMS parameter file

### DC-to-DC Communications

You use DC-to-DC communication when a program executing online in one DC/UCF system needs access to a database controlled by another DC/UCF system.

**Note:** This section describes how database requests are transferred between DC/UCF systems. It does not describe distributed applications using UCF or APPC. For information on distributed applications, see "Distributed Applications Using UCF, APPC, or TCP/IP."

**Communicating between DC/UCF systems at different release levels:** It is valid to communicate between a front-end DC/UCF system and back-end DC/UCF system that are at different release levels. Either the front-end system or the back-end system can be at a higher release level. For SVC access between systems, the SVC should be at a release level at least as high as the higher of the systems.

**System generation table definitions:** To implement DC-to-DC communication, you must define two types of tables using the system generation compiler as follows:

<table>
<thead>
<tr>
<th>Type of table</th>
<th>Sysgen statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE table</td>
<td>NODE</td>
<td>Describes what DC/UCF systems can communicate with this DC/UCF system and what communication method to use (for example, an SVC or DDS using CCI, TCP/IP, or VTAM); define one NODE statement for every node with which a given DC/UCF system can communicate</td>
</tr>
<tr>
<td>RESOURCE NAME table</td>
<td>RESOURCETABLE</td>
<td>Identifies on what node a database resides; define one resource entry for each remote database and for each local database</td>
</tr>
</tbody>
</table>

**Database request processing:** When an application issues a request for database services, the request is passed to the data services interface (DSI). If the request cannot be serviced on the local node, the request is passed to the data transfer services layer of the communications architecture. This layer uses the name server table, which contains the information provided by the system generation parameters listed above. The request is transferred from node to node (using the distributed node services layer), until it reaches the node that controls the database.

In some cases, a direct connection cannot be made between the DC/UCF system on which the database request originates and the DC/UCF system that controls the database. In these cases, a request can be routed across one or more intermediate nodes to reach the final destination.
The system that controls the database must, however, be defined as a resource in the originating system. This definition is used to perform resynchronization between the two systems and is required even if no actual two-phase commit processing is done. For more information on this requirement, see the section Resource Name Table Requirements.

**Distributed processing:** You can distribute database and application processing across DC/UCF systems in one of the following ways:

- You can define one DC/UCF system to manage the front-end processing (terminal management, etc.) of applications and another DC/UCF system to manage the CA IDMS database processing

- You can distribute the front-end processing of applications across multiple DC/UCF systems and define another DC/UCF system to manage the CA IDMS database processing for those applications. You might distribute applications based on geographic location, type of application, or size of application.

You decide the best way to set up multiple DC/UCF systems to meet the goals of your organization and maximize the use of your computing resources. The following diagram shows the components of multiple DC/UCF regions:

---

By distributing application and database processing across multiple DC/UCF systems, you minimize the constraints on system resources such as CPU and virtual storage. Additionally, by isolating database or application processing on separate DC/UCF systems, you can do the following:

- Exercise more control over the processing of strategic applications
- Insulate database processing from application processing
- Insulate applications from each other
Minimize the effects of processing outages across the user environment

**Managing the tables dynamically:** You can make a new copy of the NODE table or RESOURCE NAME table available to the DC/UCF system dynamically, by following these steps:

1. Re-generate the system with the new NODE or RESOURCE NAME table parameters
2. Dynamically reload the NODE table or RESOURCE NAME table by entering DCMT VARY RESOURCE TABLE NEW COPY

⚠️ **Note:** For more information on this DCMT command, see the System Tasks and Operator Reference section.

**Batch and Non-DC/UCF Programs**

**Contents**
- Generating IDMSOPTI and Defining SYSCTL (see page 66)

Batch programs and non-DC/UCF programs pass CA IDMS central version database requests and receive responses through the CA IDMS SVC (supervisor control routine). The following diagram shows how the SVC passes requests and data between programs executed in other regions/partitions and DC/UCF. In this diagram, the CA IDMS SVC passes program requests for database services and DC/UCF responses across regions/partitions. Program A is an online program executed without DC/UCF services. Program B is a batch program.
Batch and Non-DC/UCF Programs

In z/OS systems, an actual SVC is used. In z/VSE and z/VM systems, equivalent modules perform CA IDMS SVC communication services. The remainder of this overview uses the term SVC to refer to both types of definitions.

**Required interface modules:** Batch programs also require the following interface modules to communicate with the SVC:

- The batch interface allows batch programs to communicate with the SVC.
- TP-interface modules tailored to each supported TP monitor allow (non-DC/UCF) online programs to communicate with the SVC.
- These interface modules are installed at DC/UCF system installation time.
- To make the optional 10.2 services batch interface available, programs that were not linked in a 10.2 system must be linked with the IDML load module, and the IDMSB102 load module must be in a STEPLIB specified in the batch job JCL.

**Note:** For more information on using the optional 10.2 services batch interface, see the *CA IDMS Navigational DML Administrating section*.

Because batch programs and non-DC/UCF programs execute outside of DC/UCF, the interface modules for these types of programs must provide DC/UCF access information when they pass program requests for CA IDMS database services to the SVC. Basically, they must specify the following:

- The type of database services (central version or local mode) that the program uses.
The database that the program accesses

The DC/UCF system the program uses when using central version services

The DC/UCF system to which the program passes database requests when using central version services

**Database specifications:** When batch programs execute, the batch interface derives this information from a combination of sources as follows:

- From the IDMSOPTI module (if any) link edited with the program (for non-SQL access only).
- From the SYSCTL file specified in execution JCL.
- From alternative database and node specifications made within the SYSIDMS parameter file or within the program itself.
- From user exit 23 specifications. Exit 23 can intercept database specifications for the program during run unit initiation (non-SQL access only)

**Note:** For more information on the SYSIDMS parameter file, see the Common Facilities Guide.

**TP monitor overrides:** For non-DC/UCF online programs, the TP front-end interface module passes program information to the SVC. This information is specified when the TP front-end module is generated. At runtime, this information can be overridden, when appropriate, by any of the following specifications:

- Program database and node specifications
- Information passed by user exit 23 (non-SQL access only)

**CV Retry Processing:** If a batch job or UCFTSO session attempts to connect to a Central Version and that Central Version is not up, message DC208002 is normally issued and the batch job will await an operator response to the query: CV cv-number NOT ACTIVE REPLY RETRY OR CANCEL. A SYSIDMS parameter, CVRETRY=OFF, allows this message to be suppressed so that control is immediately returned to the application program with an appropriate ERROR-STATUS.

Under z/OS, the SYSIDMS parameter, CVRETRY_MSG_CODES=descriptor-route-codes, gives you control over the destination where message DC208002 is routed. Control over the routing of the DC208002 message facilitates the use of data center automation tools to respond to the message without operator intervention.

For more information on SYSIDMS parameters see the Common Facilities Guide.

The following SYSIDMS parameter lets you control the routing of the DC208002 message.

**CVRETRY_MSG_CODES=descriptor-route-codes**

Specifies the descriptor and route codes to be used for batch message DC208002 (CV cv-number NOT ACTIVE. REPLY RETRY OR CANCEL). descriptor-route-codes must be an eight-digit hexadecimal value.

The first four digits of descriptor-route-codes represent the descriptor codes and the last four digits represent the route codes. Each bit within the descriptor or route codes represents a code value. The first bit (x’8000’) represents code value 1 and the last bit (x’0001’) represents code value 16.
value 16. Multiple bits can be on in each set of codes so that x’8101’ represents code values 1, 8 and 16.

**Default:** x’00004000’ (representing descriptor code zero (0) and route code two)

**Database access:** The following table summarizes how DC/UCF decides which database to access given a DBNAME specification in the program or a SYSIDMS parameter, in IDMSOPTI, and in the SYSTEM statement SYSCTL clause. The same rules apply to databases that reside on remote nodes.

<table>
<thead>
<tr>
<th>Program/SYSIDMS DBNAME</th>
<th>IDMSOPTI DBNAME</th>
<th>SYSCTL DBNAME</th>
<th>DC/UCF uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specified</td>
<td>ALWAYS</td>
<td>ALWAYS</td>
<td>SYSCTL parameters</td>
</tr>
<tr>
<td>Specified</td>
<td>ALWAYS</td>
<td>DEFAULT</td>
<td>IDMSOPTI parameters</td>
</tr>
<tr>
<td>Specified</td>
<td>ALWAYS</td>
<td>NULL</td>
<td>IDMSOPTI parameters</td>
</tr>
<tr>
<td>Specified</td>
<td>DEFAULT</td>
<td>ALWAYS</td>
<td>SYSCTL parameters</td>
</tr>
<tr>
<td>Specified</td>
<td>DEFAULT</td>
<td>DEFAULT</td>
<td>Program specification</td>
</tr>
<tr>
<td>Specified</td>
<td>DEFAULT</td>
<td>NULL</td>
<td>Program specification</td>
</tr>
<tr>
<td>Specified</td>
<td>NULL</td>
<td>ALWAYS</td>
<td>SYSCTL parameters</td>
</tr>
<tr>
<td>Specified</td>
<td>NULL</td>
<td>DEFAULT</td>
<td>Program specification</td>
</tr>
<tr>
<td>Not Specified</td>
<td>ALWAYS</td>
<td>DEFAULT</td>
<td>IDMSOPTI parameters</td>
</tr>
<tr>
<td>Not Specified</td>
<td>ALWAYS</td>
<td>NULL</td>
<td>IDMSOPTI parameters</td>
</tr>
<tr>
<td>Not Specified</td>
<td>ALWAYS</td>
<td>ALWAYS</td>
<td>SYSCTL parameters</td>
</tr>
<tr>
<td>Not Specified</td>
<td>DEFAULT</td>
<td>DEFAULT</td>
<td>IDMSOPTI parameters</td>
</tr>
<tr>
<td>Not Specified</td>
<td>DEFAULT</td>
<td>NULL</td>
<td>IDMSOPTI parameters</td>
</tr>
<tr>
<td>Not Specified</td>
<td>NULL</td>
<td>ALWAYS</td>
<td>SYSCTL parameters</td>
</tr>
<tr>
<td>Not Specified</td>
<td>NULL</td>
<td>DEFAULT</td>
<td>SYSCTL parameters</td>
</tr>
<tr>
<td>Not Specified</td>
<td>NULL</td>
<td>NULL</td>
<td>Program’s subschema</td>
</tr>
</tbody>
</table>

**Note:** * If both the program and SYSIDMS parameter file specify a DBNAME parameter, the program specification overrides the SYSIDMS parameter file specification.

**Generating IDMSOPTI and Defining SYSCTL**

The following section discusses the IDMSOPTI module and the SYSCTL file for each operating system.

**z/OS: IDMSOPTI and SYSCTL**

This section describes how to do the following in z/OS:

- Generating an IDMSOPTI Module (see page 67)
Generating an IDMSOPTI Module

You can define IDMSOPTI modules for batch programs that require non-SQL database services. To generate an IDMSOPTI module, do the following:

1. Code an IDMSOPTI macro

2. Assemble and link edit the IDMSOPTI module
   To make IDMSOPTI specifications available to a batch program, link edit the IDMSOPTI module with the program. The IDMSOPTI module specifies whether the program uses CA IDMS central version or local mode services. When the program uses central version services, IDMSOPTI can provide parameters that further determine which DC/UCF system is to be accessed.

IDMSOPTI Syntax

```
IDMSOPTI
   SVC= svc-number
   CVNUM= cv-number
   CENTRAL= NO | ONLY | YES
   SYSCTL=sysctl-ddname
   NODENAM=(nodename)
   DBNAME=(database-name)
```

To generate an IDMSOPTI module:

1. Code an IDMSOPTI macro

2. Assemble and link edit the IDMSOPTI module
   To make IDMSOPTI specifications available to a batch program, link edit the IDMSOPTI module with the program. The IDMSOPTI module specifies whether the program uses CA IDMS central version or local mode services. When the program uses central version services, IDMSOPTI can provide parameters that further determine which DC/UCF system is to be accessed.
IDMSOPTI Parameters

- **SVC**
  Specifies the number of the CA IDMS SVC through which the program communicates with the DC/UCF system. svc-number must be an integer in the range 0 through 255.
  **Considerations:** To allow users to utilize the CA IDMS SVC number in a SYSCTL file at program execution, include the SYSCTL parameter (below) in the IDMSOPTI macro.

- **CVNUM**
  Identifies the DC/UCF system to the CA IDMS SVC.
  **Considerations:** cv-number must be the same value (0 through 255) specified by the CVNUMBER parameter of the system generation SYSTEM statement for the system. For more information on the SYSTEM statement, see the CA IDMS Administrating section.

- **CENTRAL**
  Specifies whether the program uses CA IDMS central version services.
  - **NO**
    The program will never use CA IDMS central version services.
  - **ONLY**
    The program will always require CA IDMS central version services.
  - **YES**
    The program can use either central version or local mode services:
    - The program uses central version services when the execution JCL for the application specifies a SYSCTL file.
    - The program uses local mode services when JCL that executes the application does not specify a SYSCTL file
    If you code CENTRAL=YES, be sure to include the SYSCTL parameter (below) in the IDMSOPTI module. At runtime, the SYSCTL ddname specified in JCL for the application must match the ddname specified in the IDMSOPTI module.

- **SYSCTL**
  Specifies the ddname of a SYSCTL file.
  **Considerations:** You should include the SYSCTL parameter when either of the following conditions applies:
  - Users can execute the program using either central version or local mode services (that is, CENTRAL=YES). For this to occur, the SVC parameter must either be omitted or set to 0.
  - Users can execute the program on different DC/UCF systems. You can optionally specify a default DC/UCF system in the IDMSOPTI CVNUM parameter. At runtime, a user can override the default system, using the system specified in the SYSCTL file. The user must specify the SYSCTL ddname named in the IDMSOPTI module for the program.

**Note:** For more information on SYSCTL files, see the considerations listed in the section Defining a SYSCTL File (https://docops.ca.com/pages/viewpage.action?pageId=328583418#:~:text=Defining%20a%20SYSCTL%20File).
- **NODENAM**
  Specifies the one- to eight-character name of a system defined to your DC/UCF communications network.
  ALWAYS/DEFAULT specifies conditions under which programs link edited with this IDMSOPTI module will pass database requests to the named system node.

  - **ALWAYS**
    The program passes database requests to the named system unless the named system is overridden by a SYSCTL file.

  - **DEFAULT**
    The program passes database requests to the named system only if both of the following are true:
    - The program does not name a remote system.
    - The SYSCTL file either does not name a remote system or specifies DEFAULT along with the remote system that it names.

- **DBNAME**
  Specifies a database name.
  This specification may be overridden. For override conditions see the section Batch and Non-DC/UCF Programs (see page 63).

  - **ALWAYS**
    The program uses the named database unless the named database is overridden by a SYSCTL file.

  - **DEFAULT**
    The program uses the named database only if both of the following are true:
    - The program does not name a database.
    - The SYSCTL file either does not name a database or specifies DEFAULT along with the database that it names.

**IDMSOPTI usage:** The following table shows the IDMSOPTI macro parameters appropriate for various programs that request CA IDMS database services from another region:

<table>
<thead>
<tr>
<th>Program execution</th>
<th>IDMSOPTI strategy</th>
<th>IDMSOPTI syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always uses central version database services on the same DC/UCF system</td>
<td>Assemble an IDMSOPTI module for programs that use the DC/UCF system.</td>
<td>IDMSOPTI SVC=svc-number,CVNUM=cv-number,CENTRAL=ONLY</td>
</tr>
<tr>
<td>Uses central version database services on various DC/UCF systems</td>
<td>Optionally assemble an IDMSOPTI module.*</td>
<td>IDMSOPTI SYSCTL=sysctl-ddname,CENTRAL=ONLY</td>
</tr>
<tr>
<td>Uses central version services on a default DC/UCF system that users can override with a SYSCTL file.</td>
<td>Assemble an IDMSOPTI module.</td>
<td></td>
</tr>
</tbody>
</table>
## Program execution

<table>
<thead>
<tr>
<th>IDMSOPTI strategy</th>
<th>IDMSOPTI syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always uses local mode database services</td>
<td>IDMSOPTI SYSTCL= sysctl-ddname ,SVC=svc-number ,CVNUM=cv-number ,CENTRAL=ONLY</td>
</tr>
<tr>
<td>Sometimes uses central version and sometimes uses local mode database services</td>
<td>Optionally assemble an IDMSOPTI module.</td>
</tr>
</tbody>
</table>

* A SYSCTL data set can provide the necessary communication information at program execution time.

### Assembling and link editing the IDMSOPTI module:

You can define several different IDMSOPTI modules for a given system. You must give each IDMSOPTI module a unique name.

**To assemble and link edit an IDMSOPTI module:**

1. Create an IDMSOPTI source module as follows:
   
   ```
   IDMSOPTI opti-parameters
   END
   ```

2. Save the IDMSOPTI source module in your custom source library.

3. Assemble and link it into your custom load library by executing the *z/OS Assemble and Link-Edit JCL (see page 465).*
   
   Substitute the name of your IDMSOPTI source member and insert the following binder statement:
   
   ```
   NAME idmsopti (R)
   ```

### Defining a SYSCTL File

You can define one or more SYSCTL files to provide alternative information for programs that require CA IDMS database services while executing in another region. This information includes the number of the CA IDMS SVC to be used. SYSCTL specifications override those given by batch and TP-monitor interface modules and IDMSOPTI modules.

You define the ddname for a SYSCTL file and determine the specifications to be contained in the SYSCTL file by using the DC/UCF system generation SYSTEM statement. At system startup time, DC/UCF copies the specifications made in the SYSTEM statement to the SYSCTL file.

At sites where more than one DC/UCF system runs concurrently, each SYSCTL file must have a unique data set name. However, all systems can use the same ddname for the SYSCTL file.

**Note:** For more information on the SYSTEM statement, see the Administrating section.
Accessing a SYSCTL file: To use a SYSCTL file in conjunction with programs executed in other regions, the user must either:

- Specify a ddname of SYSCTL in the batch application's execution JCL. The SYSCTL DD statement should point to a data set with the same data set name used by the DC/UCF system for its SYSCTL data set.
- If the program is linked with IDMSOPTI, the ddname used for the SYSCTL data set in the batch application's execution JCL must match the ddname specified for the SYSCTL in the IDMSOPTI module. The data set name pointed to by this ddname must match the data set name used by the DC/UCF systems for its SYSCTL data set.

IDMS System Operations z/VSE

This section describes how to do the following in z/VSE:

- Generating an IDMSOPTI Module (see page 71)
  - IDMSOPTI Syntax (see page 71)
  - IDMSOPTI Parameters (see page 71)
- Defining a SYSCTL File (see page 74)

Generating an IDMSOPTI Module

You can define an IDMSOPTI module for batch programs that require non-SQL database services. To generate an IDMSOPTI module, do the following:

1. Code an IDMSOPTI macro
2. Assemble and catalog the IDMSOPTI module

Note: For more information on IDMSOPTI modules, see the introduction to the System Startup section.

IDMSOPTI Syntax

IDMSOPTI Parameters

- **SVC**
  Specifies the number of the CA IDMS SVC through which the program communicates with the DC/UCF system. The value specified for svc-number should be an integer in the range 114 through 255.
  **Considerations:** To allow users to utilize the CA IDMS SVC number in a SYSCTL file at program execution, include the SYSCTL parameter (below) in the IDMSOPTI macro.

- **CVNUM**
  Identifies the DC/UCF system to the CA IDMS SVC.
  **Considerations:** CVNUM must specify the number (0 through 255) given by the CVNUMBER parameter of the system generation SYSTEM statement for the system. For more information on the SYSTEM statement, see the Administrating section.
SYSCTL
Specifies the filename of a SYSCTL file. Sysctl-filename is a seven-character filename.
Considerations: If you use a SYSCTL file to specify SVC information, you won't need to relink programs that use that SVC when you subsequently modify the SVC.
Because the SYSCTL file is used system-wide, the SYSCTL filename must represent a logical unit assignment that is supported by all partitions in the system.
When executing a batch application, the user must specify the SYSCTL filename by using the DLBL statement in the batch application's JCL. If this parameter is omitted from the JCL, the system default SYSCTL filename (specified in the system generation SYSTEM statement) is used at runtime.

UPSI
Defines which bit switch in the UPSI byte allows batch programs to use CA IDMS central version services at runtime.
Upsi-bit-switch-number specifies a bit string. Up to eight bit values can be specified in the string.
The position where you code a single 1 (one) in this bit string determines the location of the central version bit switch. Positions before the 1 (if 1 is not the first bit specified) must be coded as either zeros or blanks. Trailing blanks are treated as zeros.
For example, to specify that the fifth position in the UPSI byte is the central version switch, you would make the following specification in the IDMSOPTI UPSI parameter:

```
UPSI=00001
```

In this example, the application at runtime can use central version services if a 1 (one) is coded in the fifth position in the UPSI byte of the program's JCL. Otherwise, the application uses CA IDMS local mode services.
Considerations:

- When CENTRAL=YES (see below) is coded for the IDMSOPTI module, it is particularly useful to code an UPSI value. Runtime users can then use the UPSI switch to determine whether the program uses central version or local mode services.
- When a SYSCTL file is specified in the program JCL, the runtime UPSI setting is ignored and the program uses central version services, based on information from the SYSCTL file.

CENTRAL
Specifies whether the program uses CA IDMS central version services.

- NO
  The program will never use CA IDMS central version services.
- ONLY
  The program will always require CA IDMS central version services.
- YES
  The program can use either central version or local mode services, depending on specifications made in the JCL for the batch application:
    - The program uses central version services when either a SYSCTL file or the appropriate UPSI byte is specified in the JCL.
    - The program uses local mode services when neither of these
- **NODENAM**
  Specifies a one- to eight-character name of a system defined to your DC/UCF communications network.

- **ALWAYS**
  The program passes database requests to the named system unless the named system is overridden by a SYSCTL file.

- **DEFAULT**
  The program passes database requests to the named system only if both of the following are true:
  - The program does not name a remote system.
  - No database is specified by a SYSCTL file, or a SYSCTL file specifies DEFAULT along with the database that it names.

**IDMSOPTI usage:** The following table shows IDMSOPTI macro parameters appropriate for various programs that request CA IDMS database services from another partition:

<table>
<thead>
<tr>
<th>Program execution</th>
<th>IDMSOPTI strategy</th>
<th>IDMSOPTI syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always uses central version database services on the same DC/UCF system</td>
<td>Assemble an IDMSOPTI module for programs that use the DC/UCF system.</td>
<td>IDMSOPTI SVC=svc-number, CVNUM=cv-number, CENTRAL=ONLY</td>
</tr>
<tr>
<td>Uses central version database services on various DC/UCF systems*2</td>
<td>Optionally assemble an IDMSOPTI module.*1</td>
<td>IDMSOPTI SYSCTL=sysctl-ddname, CENTRAL=ONLY</td>
</tr>
<tr>
<td>Uses central version services on a default DC/UCF system that users can override with a SYSCTL file*2</td>
<td>Assemble an IDMSOPTI module.</td>
<td>IDMSOPTI SYSCTL=sysctl-ddname,SVC=svc-number, CVNUM=cv-number, CENTRAL=ONLY</td>
</tr>
</tbody>
</table>
### Program execution

<table>
<thead>
<tr>
<th>Always uses local mode database services</th>
<th>IDMSOPTI strategy</th>
<th>IDMSOPTI syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optionally assemble an IDMSOPTI module.</td>
<td>IDMSOPTI CENTRAL= NO</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sometimes uses central version and sometimes uses local mode database services</th>
<th>IDMSOPTI strategy</th>
<th>IDMSOPTI syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assemble an IDMSOPTI module.</td>
<td>IDMSOPTI SYSCTL= sysctl- filename</td>
<td></td>
</tr>
</tbody>
</table>

### Notes:

*1 - A SYSCTL data set can provide the necessary communication information at program execution time.

*2 - The SYSCTL statement can be omitted from the IDMSOPTI module if the system-wide default is preferred.

- **Assembling and cataloging the IDMSOPTI module:** You can define several different IDMSOPTI modules for a given system. You must give each IDMSOPTI module a unique name.

  **Note:** Any modifications to CA IDMS load libraries should be applied using MSHP. For instructions about how to assemble and link edit using MSHP, see the *Installing section -- z/VSE*.

  **To create a z/VSE IDMSOPTI module:**

  Assemble and catalog the module using the sample JCL in z/VSE Assemble JCL (see page 476). Modify the JCL by substituting the following in place of the Assembler input statements:

  ```
  PUNCH 'CATALOG idmsopti.OBJ REPLACE=YES'
  IDMSOPTI macro
  END
  ```

  - **idmsopti**
    Specifies the name of the object module containing the assembly output

---

### Defining a SYSCTL File

You can define one or more SYSCTL files to provide alternative information for programs that require CA IDMS database services while executing in another partition. This information includes the number of the CA IDMS SVC to be used. SYSCTL specifications override those given by batch and TP-monitor interface modules and IDMSOPTI modules.

You define the filename for a SYSCTL file and determine the specifications to be contained in the SYSCTL file by using the DC/UCF system generation SYSTEM statement. At system startup time, DC/UCF copies the specifications made in the SYSTEM statement to the SYSCTL file.

At sites where more than one DC/UCF system runs concurrently, each SYSCTL file must have a unique file ID. However, all systems can use the same filename for the SYSCTL file.

**Note:** For more information on the SYSTEM statement, see the Administrating section.
Accessing a SYSCTL file: To use a SYSCTL file in conjunction with programs executed in other regions, the user must either:

- Specify a filename or SYSCTL in the batch application's execution JCL. The SYSCTL filename should point to a file ID with the same file ID used by the DC/UCF system for its' SYSCTL file.
- If the program is linked with IDMSOPTI, the filename used for the SYSCTL file in the batch application's execution JCL must match the filename specified for the SYSCTL in the IDMSOPTI module. The file ID pointed to by this filename must match the file ID used by the DC/UCF systems for its SYSCTL file.

IDMS Systems Operations - z/VM

This section describes how to do the following in z/VM:

- Generating an IDMSOPTI Module (see page 75)
- Defining a SYSCTL File (see page 75)
- Using SYSIDMS to Access the Correct CV (see page 76)

Generating an IDMSOPTI Module

You can define IDMSOPTI modules for batch programs that require non-SQL database services. For more information, see the introduction to the section System Startup.

Note: For more information on IDMSOPTI with batch programs under z/VM and dynamic IDMSOPTI generation, see the Installing section -- z/VM.

To prepare an IDMSOPTI module, do the following:

1. Code an IDMSOPTI macro and save the macro statements in a file with a file type of ASSEMBLE.
2. Assemble the IDMSOPTI module and put it in a user TXTLIB library.

Multiple IDMSOPTI macros: You can code several different IDMSOPTI macros for a given system. Each IDMSOPTI macro must have a unique name. However, when you add the IDMSOPTI module to a TXTLIB library, you must give the library member the name IDMSOPTI. Therefore, each IDMSOPTI module must reside in its own TXTLIB library.

Defining a SYSCTL File

You can define one or more SYSCTL files to provide alternate information for programs that require CA IDMS database services while executing in another virtual machine. This information includes the number of the CA IDMS SVC to be used. SYSCTL specifications override those given by batch and TP-monitor interface modules and IDMSOPTI modules.
You define a ddname for a SYSCTL file and determine the specifications to be contained in the SYSCTL file by using the DC/UCF system generation SYSTEM statement. At system startup time, DC/UCF copies the specifications made in the SYSTEM statement to the SYSCTL file.

At sites where more than one DC/UCF system runs concurrently, each SYSCTL file must have a unique file ID. However, all systems can use the same ddname for the SYSCTL file.

**Note:** For more information on the SYSTEM statement, see the Administrating section.

**Accessing a SYSCTL file:** To use a SYSCTL file in conjunction with programs executed in other virtual machines, the user must either:

- Specify a ddname of SYSCTL in the batch application's execution JCL. The SYSCTL ddname should point to a file with the same file ID used by the DC/UCF system for its SYSCTL file.

- If the program is linked with IDMSOPTI, the ddname used for the SYSCTL file in the batch application's execution JCL must match the ddname specified for the SYSCTL in the IDMSOPTI module. The file ID pointed to by this ddname must match the file ID used by the DC/UCF systems for its SYSCTL file.

Previously existing batch programs that were link edited with an IDMSOPTI module continues to work as before unless a SYSCTL file is included in the runtime execution JCL. SYSCTL specifications override those in an IDMSOPTI module.

### Using SYSIDMS to Access the Correct CV

Another way of directing a batch job to a DC/UCF system is by coding the following SYSIDMS parameters in a file and pointing the SYSIDMS DD statement to that file:

- **CVMACH**
  Specifies the user id of the virtual machine in which the DC/UCF system is executing.

- **CVNUM**
  Identifies the DC/UCF system (within the named virtual machine) to the CA IDMS SVC.
  **Considerations:**
  - For z/VM batch programs communicating with DC/UCF in a z/VM machine, the CVNUM should not be coded or the value should be set to zero.
  - For z/VM batch programs communicating with DC/UCF in a guest operating system, the CVNUM value must be the CVNUMBER of the DC/UCF system.

- **NODENAM**
  Specifies the one- to eight-character name of a system defined to your DC/UCF communications network.

- **DBNAME**
  Specifies the database name on the target DC/UCF system.
  **Note:** For more information on using SYSIDMS parameters, see the Common Facilities Guide.
Generating the SVC

Contents

- z/OS (see page 77)
  - SVCOPT Syntax (see page 78)
  - SVCOPT Parameters (see page 79)

- z/VSE (see page 82)
  - DEFSVC Syntax (see page 83)
  - DEFSVC Parameters (see page 84)

- z/VM (see page 85)

An SVC is used to facilitate interpartition communication for the following:

- IDMS batch programs
- Non-DC/UCF online programs
- UCF programs
- DC/UCF to DC/UCF programs within the same operating system image

Certain online functions also use the SVC. These functions include storage protection and detection of runaway programs, for example, user-mode programs in an endless loop.

In z/OS systems, an actual SVC is used. In z/VM and z/VSE systems, equivalent modules perform CA IDMS SVC communication services. The remainder of this overview uses the term SVC to refer to both types of definitions.

Information for generating an SVC appears below for each operating system.

z/OS

In z/OS operating systems, the CA IDMS SVC provides a means of communication between DC/UCF and programs executing in another region within the same CPU.

⚠️ **Note:** For more information on SVCs, see the introduction to the section System Startup (https://docops.ca.com/display/IDMS19/SystemStartup). During installation, you generated a CA IDMS SVC. This section explains the steps to take if you want to generate a new CA IDMS SVC module and make it available for use.

⚠️ **Note:** The SVC parameters CVKEY and AUTHREQ can be used to help ensure system integrity. For more information on system integrity, see the section Installation Considerations in the current Installing section for z/OS.
Considerations: You define both the SVC number and a SYSCTL file for the DC/UCF system in the system generation SYSTEM statement. When you start up DC/UCF in a z/OS system, the following occurs:

- DC/UCF signs on to a CA IDMS SVC if an SVC number is specified at system generation time.
- DC/UCF places the SVC number in the specified SYSCTL file.

Note: For more information on specifying the SVC number and SYSCTL file for a DC/UCF system, see the Administrating section.

SVCOPT Syntax

```
#SVCOPT SVCNO=svc-number

,ENVIRON=
  MVS
  (MVS, VMCF)
  (DOS, VMCF)
  (CMS, VMCF)

,SVCXLEN=ere-extension-length

 Expansion of mvs-only-options

,AUTHREQ=
  YES
  NO

,CVKEY=dc-primary-protect-key

,SMF=
  NO
  YES

 Expansion of mvs-vmcf-options

,LAP=
  NO
  YES

,MAXTCB=max-tcb-count

,VMCFCPU=vm-mvs-cpu-number
  NO

,VMCFLOC=vmcf-address-space
  X'230'

,VMCFLOC=VMFsch=MASTER

 Expansion of vmcf-options
```
SVCOPT Parameters

- **SVCNO**
  Gives the number of the CA IDMS SVC being generated. Svc-number must be an integer in the range 0 through 255.
  **Considerations:**
  - Be sure to assign an SVC number that is higher than any SVC interrupt numbers used by your operating system. Consult your operating system documentation for SVC values that your operating system reserves.
  - SVC numbers 172, 173, and 174 are reserved SVC number for CA IDMS for use by CAIRIM. Values 200 through 255 are standard user SVC numbers.

- **ENVIRON**
  Specifies the environment in which you are operating.
  - **MVS**
    Specifies the z/OS operating system.
  - **(MVS,VMCF)**
    Specifies the z/OS operating system and activates VMCF support. The Virtual Machine Command Facility (VMCF) is only supported when the CA IDMS CMS Option is installed and when z/OS is running as a guest machine under z/VM. Batch jobs running under CMS can communicate with CVs running under z/OS using VMCF. Only one IDMS SVC in a z/OS image can use VMCF at a time.
    **Note:** Do not code this parameter merely because DC/UCF is running on a guest machine. This parameter applies only if VMCF is being used to communicate with another virtual machine (for example, UCFCMS).
  - **(CMS,VMCF)**
    Specifies the z/VM operating system and activates VMCF support. This Virtual Machine Command Facility (VMCF) option is required when running under z/VM.
  - **(DOS,VMCF)**
    Specifies the z/VSE operating system and activates VMCF support. The Virtual Machine Command Facility (VMCF) is only supported when the CA IDMS CMS Option is installed and when z/VSE is running as a guest machine under z/VM. Batch jobs running under CMS can communicate with CVs running under z/VSE using VMCF. Only one IDMS SVC in an z/VSE image can use VMCF at a time.
  - **SVCXLEN**
    Specifies the number of bytes in the ERE extension to be used by the IDMSSVCX user exit.
    **Limits:** ERE-extension-length may be a positive integer in the range 1 through 32767.

Expansion of mvs-only-options parameters
AUTHREQ
Specifies if CV startup modules must run as authorized. If AUTHREQ=YES, all CV startup modules must be linked as authorized (SETCODE AC(1)) and reside in an authorized library. This allows the startup modules to establish PC routines to perform certain necessary system functions. Once these PC routines are established, the startup module unauthorizes itself. This prevents application programs and nucleus modules, such as user exits, from exploiting operating system privileges. Specifying AUTHREQ=YES disables certain SVC functions that could be used invalidly by the following:

- A batch program running in the protect key specified by CVKEY
- A program or user exit running without storage protect in the online Environment

NO is the default.

CVKEY (z/OS)
This mandatory parameter identifies the primary protect key for DC/UCF. Valid values are 1 through 15 and *. It is recommended that 8 and 9 should not be used. A value of * permits the intentional creation of the unsecured version of the SVC. This protect key must also be specified for DC/UCF system routines defined in the z/OS program properties table (PPT) for the system. If a numeric CVKEY value is specified and a routine invokes a function reserved for use only by DC/UCF, the CA IDMS SVC verifies that the invoking routine's job or address space has the specified primary protect key. If the caller does not have the proper primary protect key, the SVC will abend the caller.

SMF
Specifies whether system monitor facility (SMF) records are required.

Expansion of mvs-vmcf-options parameters

LAP
Specifies whether code should be generated for this CA IDMS SVC to allow the system to turn LAP (low address protection) on and off as needed. If YES is specified, the necessary code is generated. Low address protection controls the use of low core storage. The SVC needs to be able to use low core storage when the VMCF option has been enabled. Therefore, if you specify (z/OS,VMCF) on the ENVIRON parameter, you should also specify LAP=YES.

MAXTCB=max-tcb-count
max-tcb-count specifies the number of save areas that are allocated to support CPU affinity code under z/OS. If DC multitasking support is enabled this value should be the same as the maximum number of TCBs that can run in all DCs using this SVC at the same time. If multitasking support is not enabled in the CV, then this should be set to 1, the default. Specifying too large a number wastes some CSA storage. Currently about 88 bytes per save area is allocated.

Note: DC/UCF recognizes this parameter only if you code (MVS,VMCF) on the ENVIRON parameter, described above.

VMCFCPU
Establishes CPU affinity for the CA IDMS SVC when issuing VMCF commands. CPU affinity is required when z/OS is running concurrently on multiple processors and is running under z/VM when VMCF is running only on one of the processors. VMCFCPU specifies the number of the
processor on which affinity is desired. Valid values are 0-15 and NO. If z/OS runs under z/VM 2.2 and above, CPU affinity should not be required.

**Note:** DC/UCF recognizes this parameter only if you code (MVS,VMCF) on the ENVIRON parameter, described above.

- **VMCFLOC**
  Identifies the low core address for the LPSW (program status word). vmcf-address-space is a hex value. DC/UCF recognizes this parameter only if you code (MVS,VMCF) on the ENVIRON parameter.
  **Note:** For a valid address space, consult with your system programmer.

- **VMCFSCH**
  Indicates that the VMCF service request block (SRB) is scheduled to the MASTER schedule address space control block (ASCB). MASTER is the only valid value.
  **Note:** DC/UCF recognizes this parameter only if you code (MVS,VMCF) on the ENVIRON parameter, described above.

**Expansion of vmcf-options parameters**

- **DBGCMES=YES|NO**
  This parameter specifies whether to allocate a 4096 byte storage area and trace VMCF external interrupts.
  Although the default is NO, YES is recommended as the overhead is slight, and the trace buffer provides an invaluable tool for investigating problems.
  **Note:** DC/UCF recognizes this parameter only if you code (...VMCF) on the ENVIRON parameter, described above.

- **PSWMODE=EC|BC**
  Specifies whether the operating system running under z/VM is doing its own paging. EC indicates that paging is being done, and virtual addresses must be converted to "real" before passing them to VMCF. BC indicates that paging is not being done, and addresses passed to VMCF do not need to be converted. The recommended values are as follows:
  - Under z/OS, EC is probably specified.
  - Under z/VM, BC should be specified.
  - z/VSE running z/VM mode should use BC.
  - z/VSE running ESA or 370 mode should use EC.

  **Note:** DC/UCF recognizes this parameter only if you code (...VMCF) on the ENVIRON parameter, described above.

- **VMBUFCT=vmcf-buffer-count**
  vmcf-buffer-count: parameter specifies the number of buffers allocated for VMCF communication. Each buffer is 4096 bytes long, page-fixed, and page aligned. The default number is 10.
  On the CV side these buffers are shared by all CVs using this SVC. The number of buffers allocated should be enough to support the number of concurrent inbound and outbound requests that could be active at one time taking into account that requests larger than 4K are split and will use one buffer for each chunk.
  On the z/VM front-end the number of buffers should be twice the number needed to handle one
ERE data packet plus overhead. It is possible for a CV to receive a packet, process it, and return it, before the front-end has a chance to free its buffer. These buffers are used to send and receive ERE packets between the front-end z/VM machines and the back-end CV machine. If a data packet is too large to fit into one buffer, it is split into two or more chunks. Each chunk is sent separately and reassembled on the target machine. For VMCF communications there are 40 bytes of overhead in each 4K buffer, so the largest chunk of data that may be sent is 4056 bytes. If the application that is using this communication also splits data packets, a size should be chosen so that either the CA IDMS CMS Option does no splitting (that is, keep the packet sizes to 4056 or less.) or the CA IDMS CMS Option does all the splitting (that is, specify a packet size to be as large as needed.) The latter option may be more efficient since the splitting takes place at a lower level in the code path.

In z/OS and z/VSE, these buffers are allocated when the first CV using the VMCF SVC is started and remains until the last CV is shutdown. Only one set of buffers is allocated for the entire system. In the z/VM front-end they are allocated on the first SVC call and are not released until z/VM clears storage during program termination. Each z/VM CV allocates these buffers at startup.

Note: DC/UCF recognizes this parameter only if you code (,VMCF) on the ENVIRON parameter, described above.

Assembling and link editing the #SVCOPT macro:

1. Create a SVCOPT source module as follows:

   ```
   #SVCOPT svcopt-parameters
   END
   ```

2. Save the SVCOPT source module in your custom source library.

3. Assemble and link it into your custom load library by executing the z/OS Assemble and Link-Edit JCL (see page 465).
   Substitute the name of your SVCOPT source member and insert the following binder statements:

   ```
   INCLUDE CAGJLOAD(IDMSMSCVC)
   INCLUDE CAGJLOAD(IDMSMSCVM) __only if ENVIRON=(MVS,VMCF)
   INCLUDE CAGJLOAD(IDMSCMO) __only if ENVIRON=(MVS,VMCF)
   INCLUDE CUSTLOAD(idmssvcx) __only if you have an IDMSSVCX exit
   ENTRY IGCnnn
   NAME IGCnnn(R)
   ```

   - idmssvcx
     (Optional) Specifies the name of your IDMSSVCX exit. For more information on creating an IDMSSVCX exit, see IDMSSVCX (see page 221).

Installing the CA IDMS SVC: You install the CA IDMS SVC module using CAIRIM. For complete information, refer to the CA IDMS installation section for your operating system.

z/VSE

You define a CA IDMS SVC to provide a means of communication between programs executing in another partition and DC/UCF. A CA IDMS SVC must be present if central version database operations are required on a z/VSE system. DC/UCF signs on to the CA IDMS SVC at system startup. For more information on SVCs, see the introduction to the section System Startup.
If the CA IDMS CMS Option is installed, you must code the #SVCOPT macro and link the resulting object file to create an IDMSDSVM phase. #SVCOPT is described in 3.5.1, "z/OS" on page 123. The link for IDMSDSVM is described later in this section.

To generate an SVC, you use the #DEFSVC macro call which creates the IDMSVCTB phase. The IDMSVCTB can define up to three SVCS for different tape release or gen levels. You may define one SVC for the most recent release and tape level, and use that SVC for previous releases.

You initialize the SVCS at z/VSE IPL by executing the CA System Adapter utility program CASAUTIL. When a DC/UCF system starts up, a phase named IDMSDSVC is loaded, which acts as an extension of the SVC.

If you have created an SVC user exit routine, it is loaded into the SVA by IDMSDSVC at CV startup time.

You generate an IDMSVCTB phase using #DEFSVC macro calls. The two types of #DEFSVC calls are the following:

- **TYPE=ENTRY** (the default) defines the SVC number and other attributes of the SVC. You can include up to three #DEFSVC macro calls of this type; each call defines one CA IDMS SVC.
- **TYPE=GEN** creates the SVC table and the punched JCL needed for the IDMSVCTB assembly and link edit. One #DEFSVC macro call of this type must appear last in the IDMSVCTB assembly.

This section explains how to do the following:

- Code the #DEFSVC macro
- Generate IDMSVCTB

**Considerations**

You define both the SVC number and a SYSCTL file for the DC/UCF system in the system generation SYSTEM statement. When you start up DC/UCF in a z/VSE system, the following occurs:

- DC/UCF signs on to a CA IDMS SVC if an SVC number is specified at system generation time.
- DC/UCF places the SVC number in the specified SYSCTL file.

**Note:** For more information on specifying the SVC number and SYSCTL file for a DC/UCF system, see the Administrating section.

**DEFSVC Syntax**

```
#DEFSVC SVC=svc-number
      .MAXCV= maximum-cv-count
      .SVCXNAME= name of the user written SVC exit module
      .SVCXLEN= svc-extension-length
```

23-Dec-2017
DEFSVC Parameters

- **SVC**
  Identifies the number of the CA IDMS SVC being generated. svc-number is an integer in the range 0 through 255.

- **MAXCV**
  Specifies the maximum number of DC/UCF regions that can signon to this SVC. maximum-cv-count is an integer in the range 1 through 256. The default is 5.

- **SVCXNAME**
  Specifies the name of the user written SVC exit module, if it exists. The default is blank.

- **SVCXLEN**
  Specifies the length, in bytes, of additional user area to be added to the ERE extension area; you must code this parameter if you use the IDMSSVCX user exit. svc-extension-length is an integer in the range 0 to 32767. The default is 0.

- **TYPE**
  Specifies the type of the #DEFSVC macro call.
  - **ENTRY**
    Indicates the #DEFSVC macro call defines an SVC.
  - **GEN**
    Creates the SVC table for the IDMSVCTB phase assembly and link-edit. A call of this type must be the last call in the IDMSVCTB assembly.

- **MAXPART**
  Specifies the maximum number of partitions allowed to run DC/UCF regions for any SVC in the z/VSE system. maximum-cv-partition-count is an integer in the range 1 to 256. The default is 3. You can specify this parameter only if you also specify TYPE=GEN.

#DEFSVC macro statement examples: The following example shows three #DEFSVC macro calls used to create the IDMSVCTB phase. The first two calls define SVCs 250 and 232; the last call creates the SVC table and controls the destination of punched JCL for the IDMSVCTB assembly. Note that you can omit all library options if the CA IDMS libraries are part of the permanent LIBDEF definitions for the partition used to generate IDMSVCTB:

```
#DEFSVC SVC=250,MAXCV=4,SVCXLEN=4,SVCXNAME=IDMSSVCX
#DEFSVC SVC=232,MAXCV=2
#DEFSVC TYPE=GEN,MAXPART=5
```

Assembling and link editing IDMSVCTB

z/VSE IDMSVCTB assembly and link edit

To create a z/VSE IDMSVCTB module:
1. Assemble and catalog the module using the sample JCL in z/VSE Assemble JCL (see page 476). Modify the JCL by substituting the following in place of the Assembler input statements:

```
PUNCH 'CATALOG idmsvctb.OBJ REPLACE=YES'
SVTB TITLE 'CA IDMS SVC TABLE'
#DEFSVC macro statement (required)
#DEFSVC macro statement (optional)
#DEFSVC macro statement (optional)
#DEFSVC TYPE=GEN (required)
END
```

2. Link the IDMSVCTB program using the sample JCL in z/VSE Link JCL. Modify the JCL by substituting the following statements in place of the Linkage editor control statements:

```
PHASE IDMSVCTB,*,SVA
INCLUDE idmsvctb
ENTRY SVTBEPI
```

   - `idmsvctb`
     Specifies the name of the object module containing the assembly output.

**Linking IDMSDSVM:** IDMSDSVM is a phase that is used when the CA IDMS CMS Option is installed. It is loaded by system adapter when IDMS is initialized. This is usually done when z/VSE is IPL'ed. It may also be reloaded using the System Adapter utility CASAUTIL RELOAD function, provided all CVs using the CA IDMS CMS Option are shutdown. IDMSDSVM is relinked when a #SVCOPT macro is compiled specifying runtime options. It must be loaded into page-fixed, shared storage.

z/VSE IDMSDSVM link edit

Link the IDMSDSVM program using the sample JCL in z/VSE Link JCL (see page 476). Modify the JCL by substituting the following statements in place of the Linkage editor control statements:

```
PHASE IDMSDSVM,_,NOAUTO,SVAPFIX
MODE AMODE(31),RMODE(ANY)
INCLUDE dsvcopt
INCLUDE IDMSDSVM
INCLUDE IDMSCMSO
ENTRY DSVMEP
```

   - `dsvcopt`
     Specifies the name of the object file generated by #SVCOPT assembly.

**z/VM**

Communication between DC/UCF and programs running in another z/VM machine requires that an #SVCOPT be created for both the DC/UCF back-end machine and the z/VM front-end machine.

For DC/UCF, the #SVCOPT is linked as part of the IDMSVMCF module. A different version of #SVCOPT is generated for DC/UCF than for a batch job as the requirements for each are different from the other.

Similarly, to enable a program executing in a z/VM virtual machine to access DC/UCF system executing under a z/OS or z/VSE guest operating system, an #SVCOPT macro also needs to be generated that specifies VMCF as part of the ENVIRON= parameter.
Under z/OS, the generated #SVCOPT module is linked with the CA IDMS SVC.

In z/VSE, the equivalent #SVCOPT module is linked as part of module IDMSDSVM and is loaded as part of CA-System-Adapter initialization.

**Note:** For more information on CA IDMS central version operations for programs in a z/VM virtual machine, see Appendix C, “z/VM Considerations.”

Under z/VM, CA IDMS SVC functions are performed by both the IDMSVMCF and IDMSUSVC modules as follows:

IDMSVMCF intercepts program calls for CA IDMS database services and routes the calls to the VMCF.

- IDMSUSVC provides the VMCF with an entry point into DC/UCF and routes to the VMCF the calls that are issued by DC/UCF to batch application programs.

The IDMSUSVC and IDMSVMCF modules are provided as an object deck at installation time.

**Note:** For more information on these modules, see the Installing section -- z/VM. For more information on CA IDMS SVC operations, the introduction to the section System Startup.

You define both the SVC number and a SYSCTL file for the DC/UCF system in the system generation SYSTEM statement. When you start up DC/UCF in a z/VM system, the following occurs:

- DC/UCF signs on to a CA IDMS SVC if an SVC number is specified at system generation time.
- DC/UCF places the SVC number in the specified SYSCTL file.

**Note:** For more information on specifying the SVC number and SYSCTL file for a DC/UCF system, see the Administrating section.

**IDMSSVCX support:** z/VM users can install user exit IDMSSVCX at their site. This exit captures user and system information whenever an external request unit attempts to sign on to DC/UCF under z/VM. To install this exit in your system, modify the IDMSSVCX macro for the system so that:

- VM specifies YES, indicating that the macro runs under z/VM.
- SVCXLEN specifies the length of the ERE SVC extension to be used by IDMSSVCX. Specify the length in bytes. The valid range is 1 through 32767, and the default is 0.

**Note:** When the contents of the ERE SVC extension are written to the task statistics record in the DC/UCF log, only the first 40 bytes of the ERE SVC extension are written.
Distributed Applications Using UCF, APPC, or TCP/IP

When the program is executed by a terminal attached to a TP monitor supported by UCF, the CA IDMS SVC passes input/output (I/O) requests and other information between the DC/UCF region/partition and the region/partition in which the TP monitor executes.

UCF front- and back-ends

To enable the TP monitor to communicate with the SVC, the site must install a UCF front-end in the TP-monitor region/partition. The UCF back-end must be installed in the DC/UCF region/partition. The UCF runtime environment is described in Introduction. For information on how to generate a UCF system, see UCF System Generation Statements (see page 161).

Under UCF, an application can be distributed between the front-end system and the back-end system in an environment that consists of one of the following:

- Two DC systems
- A DC back-end and a front-end on a system using one of the TP-monitors supported by DC.

For more information, see the following topics:
- UCF Programs (see page 87)
- Advanced-Program-to-Program Communications (see page 109)
- TCP/IP Support (see page 113)

UCF Programs

The distribution of applications is possible only for tasks that are associated with terminals. The task must be initiated from the front-end system that owns the terminal. During the execution of distributed applications, control of the terminal is transferred from one program to another. All involved programs can issue terminal I/O requests.

See the following topics for more information:
- Terminating Task Data Transfer (see page 88)
- Non-terminating Task Data Transfer (see page 94)
Terminating Task Data Transfer

Terminating task data transfer from program storage to a terminal or device uses UCF to pass data and control *serially* between a UCF front-end and back-end. The front-end initiates a task on the back-end system and waits until the task is complete to return the requested information. The information usually includes a request to begin a new task on the front-end. This process can be repeated as many times as necessary to accomplish the needs of the application. Using industry-wide terminology, the UCF back-end is an example of a user-written connection-less server.

The following diagram shows a sample distributed application. In this example, Program A running on a UCF front-end (for example, CICS) passes a part number to program B and terminates. Program B retrieves the requested information and passes it back to a new task, Program C:

Terminating Task Data Transfer

The next sections explain:

- How data and control transfer between programs on the front-end and programs on the back-end.
- How the DC back-end transfers control and data to the front-end.
- How the front-end system transfers control and data to the DC back-end. The following front-end systems are presented:
  - DC
  - CICS
  - BATCH
How UCF Transfers Control and Data

Applications distributed between UCF front- and back-ends use UCF facilities to pass data and control. General information on transferring data and control is presented below.

For details about passing data and control in a distributed application, see the information presented later in DC Back-end (see page 90) and in the specific description of your back-end system.

Passing data

When you pass data between systems in a UCF distributed application, the data must be passed as a single contiguous piece of data.

Front-end to back-end transfers

When a program transfers control from the front-end to the back-end, the following occurs:

1. The transferring program specifies the next task or transaction to be executed on the front-end. This task or transaction must be defined to invoke the appropriate UCF front-end program.

2. The UCF front-end program does the following:
   - Establishes the UCF connection to the back-end.
   - Passes data to the back-end if any was prepared by the transferring program.
   - Initiates on the back-end system a task that has the same name as the related front-end task or transaction. The back-end task must be defined to invoke the program to which control is being transferred.

The following diagram shows what happens when control transfers from the front-end to the back-end of a distributed application. In this example, front-end program A transfers data and control to back-end program B. To do this, program A executes TSKB, which in turn invokes the UCF front-end program. The UCF front-end program passes data and transfers control to the back-end.

```
Operating System | Data and Control | SVC
Supervisor

Program A
TSKB

UCF Front-End System

Program B

UCF Front-End System
DC Back-End System
```

Back-end to front-end transfers
When a program transfers control from the back-end to the front end, the following occurs:

1. The transferring program specifies the next task or transaction to be executed on the back-end. This task or transaction must be defined to invoke program RHDCUXFT.

2. Program RHDCUXFT, the UCF program that handles transfers to the front-end, does the following:
   - Initiates on the front-end system a task or transaction that has the same name as was specified by the back-end program. The front-end task or transaction must be defined to invoke the program to which control is being transferred.
   - Passes data to the front-end, as necessary.
   - Terminates the UCF connection.

Program RHDCUXFT must be defined at system generation time with the following attributes:

```
LANGUAGE ASSEMBLER REENTRANT NOPROTECT
```

The following diagram shows what happens when control transfers from the back-end to the front-end of a distributed application. In this diagram, back-end program B transfers data and control to front-end program C. To do this, program B executes task TSKC, which in turn invokes program RHDCUXFT. Program RHDCUXFT passes data and transfers control to the front-end.

```
Operating System Supervisor

SVC Data and Control

Task C

Program RHDCUXFT

Program C

Program B TSKC

UCF Front-End System

DC Back-End System
```

**DC Back-end**

The following information is presented below for the DC back-end:

- How back-end DC programs transfer control and data
- How to define tasks and programs for use on the back-end

**Note:** For information on front-end programs, see the discussion later in this section about the front-end system in use at your site.
Back-end DC programs

The following considerations apply to programs executing on the DC back end:

- A DC program transferring control to a program on the front-end system does the following:
  - Passes data in storage that is USER KEEP LONG with a STGID of PDAT. The data being passed must begin with a record descriptor word (RDW). The RDW is the standard four-byte prefix for variable-length records. The first two bytes contain the binary length of the data plus 4; the second two bytes contain binary zeros.
  - The following example illustrates the storage used to pass the character string ABCDE:

    DC    H'9', H'0'    RDW
    DC    C'ABCDE'      DATA

  - Transfers control to the program on the front-end system by returning to DC and specifying the next task to be executed for the terminal. The specified task, which must be defined as NOINPUT, indirectly causes the next program to execute.
    - In the following example, the DC program specifies that task TSKC is the next task to be executed:
      
      #RETURN NXTASK='TSKC'

- A DC program receiving control from a program on the front-end system retrieves passed data by obtaining USER KEEP LONG storage with a STGID of PDAT. The first four bytes of the passed data comprise the RDW.

Defining tasks and programs

When defining tasks and programs on the back-end DC system, keep in mind that to transfer control from the back-end to the front-end, the transferring program specifies the next task to be executed. The specified back-end task must be defined to invoke program RHDCUXFT with the NOINPUT option (see the example below).

Before terminating the UCF connection, program RHDCUXFT moves the passed data to the front-end and initiates on the front-end system a task or transaction that has the same name as the specified back-end task. The front-end task or transaction must be defined to invoke the program to which control is being transferred.

For example, if program B executes on a DC back-end system and program C executes on a front-end system, the following back-end definitions are required:

- DC task definition:
  
  TASK TSKC INVOKES RHDCUXFT NOINPUT.

- DC program definition:
  
  PROGRAM RHDCUXFT LANGUAGE ASSEMBLER REENTRANT NOPROTECT.

Example of a back-end program

The following example of distributed applications shows the relevant sections of the DC back-end program (B). Program B performs the following operations:

1. Obtains the PDAT storage containing the passed data from Program A on the front-end:
1. Extracts the part number from the passed data:
   \[
   \text{MVC PARTNUM(10), 4(R2)}
   \]

2. Frees the 'PDAT' storage:
   \[
   \text{#FREESTG ADDR=(R2)}
   \]

3. Obtains new PDAT storage to pass the part information to program C on the front-end:
   \[
   \text{#GETSTG TYPE=(USER, LONG, KEEP), LEN=20, STGID='PDAT', ADDR=(R2), INIT='00'}
   \]

4. Copies the part information into the new PDAT storage:
   \[
   \text{MVC 4(8,R2), PARTPRIC PRICE}
   \text{MVC 12(8,R2), PARTQTY QUANTITY ON HAND}
   \text{MVC 0(2,R2), =H'20' SET LENGTH IN RDW}
   \]

5. Transfers control to program C on the front-end and returns to DC:
   \[
   \text{#RETURN NXTTASK='TSKC'}
   \]

### DC Front-end

In an environment that consists of two DC systems, the communication between those systems is achieved by the SVC (if both systems are on the same CPU) or by DDS (using the CCI, TCP/IP, or VTAM access method). Both systems own terminals, and each system can have both a UCF DC front-end program and the appropriated line for the communication. In this case, each system can be thought of as a front-end or a back-end, depending on the terminal to which the user is assigned.

Distributed applications that are executed in such an environment should always transfer control by means of the UCF DC front-end program. The UCF DC front-end program determines whether the associated terminal is already a UCF or a DDS terminal. If it is, the program branches to RHDCUXFT, and the transfer is handled as a back-end to front-end transfer.

The methods for transferring control and data used by programs on the DC front-end are the same as those used by back-end programs. These methods are discussed in DC Back-end.

### CICS Front-end

This section contains the following information for applications distributed between a DC back-end and a CICS front-end:

- How front-end CICS programs transfer control and data
- How to define transactions and programs for use on the CICS front end
- Example of an application distributed between CICS and DC
CICS programs

Programs that execute on the CICS front-end system transfer control and data as described below:

- A CICS command-level program transferring control to a program on the DC/UCF system queues the next transaction to be executed for the terminal and specifies the location and length of the data to be passed. The specified transaction indirectly causes the next program to execute.

  In the following example, the next transaction to be executed is TSKB, and the data to be passed is located at PARTNUM:

  \[
  \text{EXEC CICS START TRANSID('TSKB') TERMID(EIBTRMID) FROM(PARTNUM) LENGTH(10)}
  \]

  When transferring control to a program on the DC/UCF system, a CICS command-level program returns to CICS:

  \[
  \text{EXEC CICS RETURN}
  \]

- A CICS command-level program receiving control from a program on the DC/UCF system retrieves passed data:

  \[
  \text{EXEC CICS RETRIEVE INTO(data-area) LENGTH(data-length-area)}
  \]

Defining transactions and programs

When defining transactions and programs for the CICS front-end system, keep in mind that to transfer control from the front-end to the back-end, the transferring program specifies the next task or transaction to be executed. The specified task or transaction must be defined to invoke the UCF front-end program.

When the UCF connection is established, the UCF front-end program moves the passed data to the back-end and initiates on the back-end system a task that has the same name as the specified front-end task or transaction. The back-end task must be defined to invoke the program to which control is being transferred.

For example, if program A executes on a CICS front-end system and program B executes on a DC back-end system, the following front-end definitions are required:

**CICS program and transaction definitions:**

```plaintext
DEFINE TRANSACTION(TSKA) PROGRAM(A) ...
DEFINE TRANSACTION(TSKB) PROGRAM(UCFCICS) ...
DEFINE PROGRAM(A) ....
DEFINE PROGRAM(UCFCICS) LANGUAGE ASSEMBLER CEDF(NO)
EXECKEY(CICS) RESIDENT
```

**Note:** For information on defining DC back-end tasks and programs, see DC Back-end.

Example of a distributed application
The following example of distributed applications shows the relevant section of programs A and C on a CICS front-end system. The programs pass to and receive data from Program B on the DC back-end system. An example of Program B appears in DC Back-end.

Program A (CICS) performs the following operations:

- Passes data and transfers control:
  
  EXEC CICS START TRANSID('TSKB') TERMID(EIBTRMID)
  FROM(PARTNUM) LENGTH(10)

- Returns to CICS:
  
  EXEC CICS RETURN

Program B on the DC back-end passes the part information and control to program C.

Program C (CICS) performs the following operations:

- Sets the length of the input area for the passed data:
  
  MVC DATALEN,=H'16' SET LENGTH OF INPUT AREA

- Retrieves the passed price and quantity information:
  
  EXEC CICS RETRIEVE INTO(PARTPRIC) LENGTH(DATALEN)

Non-terminating Task Data Transfer

Non-terminating task data transfer allows a front-end program in one region to exchange multiple packets of information with a back-end program running in a DC/UCF region. Each packet can contain up to 32000 bytes of data. This provides a very efficient method for an application running in one region (such as a WEB access program running in a batch or CICS region) to access and/or update information in a CA IDMS database.

For example, in the following diagram, control returns to the front-end task (program A) after the back-end passes the requested information:
Non-terminating Task Data Transfer

Using industry-wide terminology, the UCF back-end is an example of a user-written connection-oriented server. This is accomplished as follows:

1. The front-end application links to the UCF distributed application support (UDAS) synchronous front-end program passing parameters that contain a buffer address and a buffer size. The UDAS front-end program emulates a UCF terminal to exchange buffers with the back-end.

2. The back-end UCF system "thinks" it is communicating with a terminal with an unusually large screen size (up to 31K). The back-end IDMS application is coded as if it were a conversational task communicating with a terminal.

Front-ends supported

UDAS synchronous supports CICS and BATCH front-ends.

Flow of Control

The "conversation" between the back-end and front-end applications always originates on the front-end application by passing the following parameter list to the UDAS front-end program on all requests:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERPARM DSECT</td>
<td>Address of send/receive buffer</td>
</tr>
<tr>
<td>USERBUF DS A</td>
<td>Length of send/receive buffer</td>
</tr>
<tr>
<td>USERBUFL DS F</td>
<td>Session ID</td>
</tr>
<tr>
<td>USERETCD DS F</td>
<td>Return code</td>
</tr>
</tbody>
</table>

Establishing a session with the back-end system

1. The front-end application links to the UDAS front-end program, specifying USERSES=0 (all other parameters are ignored)
2. After the UDAS front-end program establishes a session with the back-end system, it returns control to the front-end application passing a session ID in USERSES and a return code in USERETCD.

The session ID must be passed to the UDAS front-end program on all subsequent requests. You should check the return code after all requests. A list of valid Return codes appears later in this section.

Allocating a SEND buffer

Next the front-end application links to the UDAS front-end program with the address and length of a SEND buffer. The first SEND buffer must contain the task code that is to be invoked on the back-end. Optionally, the buffer may also contain data to satisfy the first read done by the back-end application.

The back-end task whose name is specified in the first SEND buffer should be defined as an INPUT task in the back-end CA IDMS system.

Passing information back and forth

After a session is established and a back-end application has been invoked, the front-end and back-end application programs are in a SEND/RECEIVE "conversation". The back-end application has control of the conversation. The front-end application always links to the UDAS front-end program with a buffer address.

If the back-end application does a write (#TREQ PUT), then the UDAS front-end program will move the data that the back-end application has written into the front-end application's buffer. The USERBUFL parameter indicates the maximum buffer length that the front-end application can receive. After the UDAS front-end program has moved data into the buffer, USERBUFL will contain the actual length of the data received.

If the back-end application does a read (#TREQ GET), then the UDAS front-end program will get data from the front-end application's buffer and send it to the back-end.

Ending a conversation

To terminate the conversation, the back-end program must link to the program RHDCBYE. RHDCBYE terminates the session and frees resources so that the session can be used by other front-end programs.
and input
Signon complete <--- Return with USERSES
Return code 4
USERBUF=taskcode+input

(SEND) ------------ Send data to back end Start task ------- #TREQ GET
First <-------- (READ)
input already received ---- Read complete
*************
Process
*************
Send complete <--- Return <----------------------------- #TREQ PUT
Return code 8 (WRITE)
USERBUF=buffer

USERBUFL=max ------- Move data to buffer
Put length in USERBUFL
Write complete ------------------------ Write complete
Repeat write,
if needed.
Read for next request.
Receive com- <--- Return <----------------------------- #TREQ GET
plete.

Return code 4.
No more requests.
Send ‘All Done’ — Send data to back end ---------------- Read complete.
Finish session <--- RHDCBYE <------ Link to BYE
Send complete. <-- Return -------------------------------- #RETURN
Return code 0.
End

Return codes

When the front-end application gets control back from UDAS, the USERETCD field will be filled in with an appropriate return code as follows:

<table>
<thead>
<tr>
<th>Return code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The SEND/RECEIVE communication between the front-end and the back-end is closed and finished normally.</td>
</tr>
<tr>
<td>4</td>
<td>A &quot;READ&quot; is expected (SEND). The next call to UDAS will send data in USERBUF to the back-end application.</td>
</tr>
<tr>
<td>8</td>
<td>A &quot;WRITE&quot; is expected (RECEIVE). The next call to UDAS will receive data from the back-end and put data in USERBUF.</td>
</tr>
<tr>
<td>Greater than 8</td>
<td>Any other value in USERETCD is abnormal and is typically an abend condition. In most cases, any other code will be a 4-character alphabetic abend code, such as A101.</td>
</tr>
</tbody>
</table>

**Note:** For more information on these codes, see the *Messages section.*
Return codes are useful in keeping the front-end and back-end applications in sync with each other regarding reads (SENDS) and writes (RECEIVES). For example, if the front-end application gets control back from UDAS with a return code of 8, but the application was expecting to do a read, then the two applications are out of sync and should be debugged. As another example, if the back-end application abends, a "FINISH" request is sent to the front-end (RC = 0) when the application was probably expecting a read or write request.

Considerations for the back-end program

- If data is sent with the task code on the first SEND from the front end to the back-end, the back-end application will receive blanks the length of the task code before it's data.

- The back-end application must always link to RHDCBYE to terminate a conversation and clean up the session. The following are some recommendations to guarantee that RHDCBYE is invoked:
  1. Code all read and write macros (#TREQ) with COND=ALL, so that the back-end application will get control if there is an I/O error and then can link to RHDCBYE.
  2. Resource time out should be generated to invoke RHDCBYE. This is the default.
  3. The back-end application should establish a STAE exit that will get control and link to RHDCBYE in case it terminates abnormally.

Examples of a UDAS DC back-end application

Up-to-date sample Assembler and COBOL programs can be found in members UDASCVSA and UDASCVSC in sample JCL. For your convenience, a version of each of the sample programs follows.

Sample UDAS DC back-end Assembler program

UCFTEST1 TITLE 'UDAS TEST SEND RECEIVE'
* SAMPLE UDAS IDMS/DC BACK-END APPLICATION
* UCFTEST1 EP=UCFEP1
    #MOPT CSECT=UCFTEST1,ENV=USER
UCFTEST1 CSECT
ENTRY UCFEP1
UCFEP1 DS 0H
LR R12,R15
USING UCFEP1,R12
SPACE
    #GETSTG LEN=WORKLEN,PLIST=*,ADDR=(R4),TYPE=(USER,SHORT), INIT=X'00'
    USING WORKD,R4
    #STAE PGM='RHDCBYE'
* GET REQUEST
GETREQ DS 0H
    #TREQ GET,INAREA=INAREA,MAXIN=INLEN,COND=ALL,OPTNS=(UPLOW)
    LR R3,R15 SAVE RETURN CODE FROM READ.
    #SNAP TITLE='UCFTEST1, INAREA/PARTIAL OUTAREA', AREA=(INAREA,INLEN,OUTAREA,100)
    LTR R3,R3 ANY ERRORS ON READ?
    BNZ ERROR YES, HANDLE ERROR.
    * CHECK FOR LAST INDICATION FROM FRONT-END
    CLC INAREA(8),=C'ALL DONE' FRONT-END FINISHED?
    BE DISCON YES, DISCONNECT.
    CLC INAREA(8),=C'SUSPEND' FRONT-END FINISHED FOR NOW?
    BE RETURN YES, SUSPEND.
SPACE
******************************************************************************
** TALK TO DATA BASE TO CREATE RESPONDE **
******************************************************************************
Sample UDAS DC back-end COBOL program

*RETRIEVAL
ID DIVISION.
PROGRAM-ID. UCFTESTC.
* THIS SAMPLE COBOL PROGRAM HAS SAME FUNCTIONALITY
* AS ASSEMBLER PROGRAM UCFTEST1. IT IS AN EXAMPLE OF
* A BACK-END UDAS COBOL APPLICATION PROGRAM.
ENVIRONMENT DIVISION.
IDMS-CONTROL SECTION.
PROTOCOL MODE IS IDMS-DC.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-START PIC X(50) VALUE '***** UCFTESTC WORKING STORAGE STARTS HERE *****'.
01 OUTAREA PIC X(31) VALUE 'FIRST WRITE ********##END DATA'.
01 OUTLEN EQU *-OUTAREA
SPACE
01 OUTARE2 PIC X(31) VALUE '2ND WRITE - REPEAT ##########'
05 FILLER PIC X VALUE 'B' OCCURS 30000.
05 FILLER PIC X VALUE '##END HERE'.
01 OUT ARE2-END PIC X.
01 INAREA.
05 INAREA1 PIC X(8) VALUE SPACES.
05 INAREA2 PIC X(10) VALUE 'ALL DONE'.
05 INAREA2-END PIC X.
01 INLEN PIC S9(8) USAGE COMP.
01 PROCESSING-INDICATOR PIC X VALUE 'C'.
01 SNAP-TITLE PIC X(134) VALUE 'UCFTESTC, INAREA/PARTIAL OUTAREA'.
EJECT
01 COPY IDMS SUBSCHEMA-CTRL.
EJECT
PROCEDURE DIVISION.
100-MAIN-Routine SECTION.

* SET UP ABEND EXIT. IF PROGRAM TERMINATES ABNORMALLY,
* RHDCBYE WILL BE CALLED. THIS WILL NOTIFY FRONT-END
* THAT BACK-END HAS TERMINATED (RETURN CODE OF 0 WILL
* BE RETURNED TO FRONT-END UDAS APPLICATION PROGRAM).
* FRONT-END PROGRAM SHOULD TAKE APPROPRIATE ACTION.
* SET ABEND EXIT ON PROGRAM 'RHDCBYE'.

PERFORM 200-SEND-RECEIVE UNTIL NOT CONTINUE-REQUESTED.

IF BYE-REQUESTED
THEN TRANSFER CONTROL TO 'RHDCBYE' RETURN.
GOBACK.

200-SEND-RECEIVE SECTION.

* RECEIVE REQUEST FROM FRONT-END. PROCESS REQUEST.
* IF FRONT-END SENDS AN INDICATOR TO TERMINATE, SET
* APPROPRIATE FLAG.
* TEST PROGRAM PROCESSING JUST CONSISTS OF SNAPPPING
* A PORTION OF THE INPUT/OUTPUT BUFFERS AND SENDING
* TWO FIXED BUFFERS TO FRONT-END FOR EACH FRONT-END
* REQUEST.
READ TERMINAL INTO INAREA MAX LENGTH 80
RETURN LENGTH INTO INLEN.

PERFORM IDMS-STATUS.

SNAP TITLE SNAP-TITLE FROM INAREA LENGTH INLEN
FROM OUTAREA LENGTH 100.

PERFORM IDMS-STATUS.

IF ALL-DONE
THEN MOVE 'B' TO PROCESSING-INDICATOR
GO TO SEND-RECEIVE-EXIT.

IF SUSPEND
THEN MOVE 'S' TO PROCESSING-INDICATOR
GO TO SEND-RECEIVE-EXIT.

* AT THIS POINT, A REAL APPLICATION WOULD MAKE
* DATABASE CALLS AND/OR PERFORM OTHER PROCESSING
* BASED ON THE MESSAGE FROM THE FRONT-END.
* FOR TEST PURPOSES, JUST SEND TWO BUFFERS,
* THEN ISSUE READ TO GET NEXT MESSAGE FROM FRONT-END.
WRITE TERMINAL FROM OUTAREA LENGTH 31.
PERFORM IDMS-STATUS.
WRITE TERMINAL FROM OUTAREA2 TO OUTAREA2-END.
PERFORM IDMS-STATUS.

SEND-RECEIVE-EXIT.
EXIT.

COPY IDMS IDMS-STATUS.
IDMS-ABORT SECTION.
IDMS-ABORT-EXIT.
EXIT.
CICS Front-end

You create the CICS UDAS front-end using the same assembly and link procedures as a normal UCF CICS front-end except that you use macro #UDASCIC instead of #UCFCICS and you use different parameters on the #UCFUTD macro as shown below in the Sample UDAS CICS Front End Assembly. You do not need the UCFCICZ module for #UDASCIC.

UDAS CICS front-end programs contain an options table and a small stub module that calls the main processing routines that reside in separate modules. To create a UDAS CICS front-end program, assemble the #UDASCIC macro and link the resulting object as shown in the following section.

UDASCIC Syntax

label #UDASCIC

    OS = OS
        DOS

    ,FREESTG= YES
        NO

    ,PRINT = print-option
        OFF

UDASCIC Parameters

- **FREESTG=YES|NO**
  Specifies whether CICS GETMAIN storage should be freed at the end of a conversation. If FREESTG=YES is coded, then all the storage allocated by UDASCIC via CICS GETMAINs will be freed when a conversation terminates (that is, at the time USERSESS is set to zero). If FREESTG=NO is coded, then the allocated storage will be held until task termination. If multiple conversations are started and ended in the same CICS task, then a storage build-up may result.

- **OS=OS|DOS**
  Specifies the operating system under which the CICS front-end system executes. OS applies to all supported z/OS versions of CICS. DOS applies to all supported z/VSE versions of CICS. The default is OS.

- **PRINT=print-option**
  Specifies what print option the Assembler is to use while processing the COPY commands which copy in the DSECTs used in the UCF front-end module. Valid options are ON/GEN/NOGEN. See the documentation for your Assembler product for details. The default is OFF.

**Note:** For more information on the #UCFCICS macro, see UCF Operations. For more information on IDMSINTC, IDMSCINT, and IDMSTRUE, see CICS Considerations.

The following are the limitations under CICS:
No terminal-less tasks are allowed. However, a UDASCIC program can be invoked from a Bridge Facility.

SEND/RECEIVE conversations cannot extend over a pseudo-converse.

Only CICS command level programming is allowed for the front-end application.

A CICS UDAS application program can be coded in Assembler or COBOL. The UDASCIC program should be called using an EXEC CICS LINK with a COMMAREA as shown in the sample programs later in this section.

The field UDAS-BUFFER-ADDRESS must be set to ADDRESS OF UDAS-USER-BUFFER before the first call to UDASCIC as shown in the sample COBOL program later in this section.

Creating a UDAS CICS front-end program

Create a sample z/OS or z/VSE UDAS front-end program by following the example in member UDASCICI in sample JCL.

Note: The front-end assembly specifies the largest possible buffer size in the front-end table. It also specifies only one #UCFUTD macro. These are the recommended values.

Sample assembly JCL can be found in member UDASCICI in sample JCL.

Each UDAS CICS front-end program must be defined in the CICS CSD. If you need a template, see the definition of PROGRAM(UDASCIC) in member CICSCSD of the CAGJSRC library. The program must be defined as RESIDENT.

Creating a UDAS CICS front-end application program

Up-to-date sample Assembler and COBOL programs can be found in members UDASCICA and UDASCICC in sample JCL. For your convenience, a version of each of the sample programs follows.

Sample CICS front-end COBOL application:

```
TITLE 'TESTCICS -- UDAS TEST PROGRAM'
DFHEISTG DSECT
EYECATCH DS CL8
  DS 0F
COMREG DS 0XL14 PARAMETERS TO PASS
USERBUF DS A ADDRESS OF READ/WRITE BUFFER
USERBUFL DS F LENGTH OF BUFFER/DATA
USERSES DS F SESSION ID
USERETCD DS F RETURN CODE
COMLEN EQU *-COMREG LENGTH OF LINK AREA
SPACE 2
SNAPLEN DS H'00' 31K BUFFER
BUFFER DS XL31000'00' 31K BUFFER
BUFLEN EQU *-BUFFER
ENDDATA EQU *-COMREG
TESTCICS CSECT
  LA R12,4095(R3)
  LA R12,1(R12)
  USING TESTCICS,R3,R12 2ND BASE REG
```
MVC EYECATCH(8),='CL8'PARMPARM'
LA R1,COMLEN
STH R1,SNAPLEN LENGTH TO SNAP OUR STORAGE

* SIGNON TO UDAS (#UDASCIC)
SPACE
XC USERSES,USERSES NO SESSION ID = SIGNON
SPACE
EXEC CICS LINK PROGRAM ('UDASCIC')
SPACE

* SESSION ID SHOULD HAVE BEEN RETURNED TO US. LETS SNAP IT TO SEE.
* LA R1,COMLEN
* STH R1,SNAPLEN LENGTH TO SNAP OUR STORAGE
* EXEC CICS DUMP FROM(COMREG) LENGTH(SNAPLEN) DUMPCODE(DMP1)
SPACE
* TEST RETURN CODE FROM SIGNON. WE EXPECT SEND FIRST.
CLC USERETCD,='F'4' RC=4 (SEND)?
BNE ERROR NO, WE HAVE AN ERROR.
SPACE 2

* OUR FIRST SEND WILL CONTAIN DC TASK CODE AND DATA.
MVC BUFFER(8),='CL8'UCFTEST1' TASK CODE
MVC BUFFER+8(20),='CL20'SAMPLE DATA 12345678'
SPACE
LA R1,BUFFER ST R1,USERBUF GIVE ADDRESS OF BUFFER TO UDAS
LA R1,28 LENGTH OF TASK CODE + DATA
ST R1,USERBUFL GIVE TO UDAS
SPACE
EXEC CICS LINK PROGRAM ('UDASCIC')
SPACE

* EXEC CICS DUMP FROM(COMREG) LENGTH(SNAPLEN) DUMPCODE(DMP2)
* TEST RETURN CODE -- WE EXPECT A RECEIVE NEXT.
CLC USERETCD,='F'8' RC=8 (RECEIVE)
BNE ERROR NO, WE HAVE AN ERROR.
SPACE 2

* NOW WE WILL RECEIVE TO GET OUR RESPONSE. FOR THIS EXAMPLE
* WE'LL JUST KEEP READING UNTIL WE GET A RETURN CODE NOT EQUAL TO 8.
* LH R1,=AL2(ENDDATA) WE'LL SNAP ALL OF OUR STORAGE
* STH R1,SNAPLEN LENGTH TO SNAP
T100 LH R1,=AL2(BUFLEN) ST R1,USERBUFL TELL UDAS MAXIMUM LENGTH TO REC
SPACE
XC BUFFER(200),BUFFER
SPACE
* LINK TO UDAS TO RECEIVE
EXEC CICS LINK PROGRAM ('UDASCIC')
SPACE

* SNAP STORAGE TO SEE WHAT WE RECEIVED, ACTUAL LENGTH IN USERBUFL.
* EXEC CICS DUMP FROM(COMREG) LENGTH(SNAPLEN) DUMPCODE(DMP3)
* TEST RETURN CODE
CLC USERETCD,='F'8' RC=8 ? (RECEIVE)
BE T100 YES, KEEP RECEIVING.
CLC USERETCD,='F'4' RC=4 ? (SEND)
BNE ERROR NO, SOMETHING WENT WRONG>
SPACE 2

* WE COULD SEND ANOTHER DATA REQUEST AT THIS POINT, BUT THIS IS
* ENOUGH FOR THIS TEST.
* WE MUST SEND BACKEND DC TASK AN INDICATION THAT WE HAVE NO
* MORE REQUESTS SO IT CAN SIGNOFF OUR SESSION.
SPACE
MVC BUFFER(8),='CL8'ALL DONE' TELL BACKEND TASK
LA R1,8 LENGTH OF MESSAGE
ST R1,USERBUFL GIVE TO UDAS
* LA R1,BUFFER
* ST R1,USERBUFL GIVE ADDRESS OF BUFFER TO UDAS
SPACE
* LINK TO UDAS TO SEND
EXEC CICS LINK PROGRAM ('UDASCIC')
COMMAREA(COMREG) LENGTH(COMLEN)
SPACE
EXEC CICS DUMP FROM(COMREG) LENGTH(SNAPLEN) DUMPCODE(DMP2)
* TEST RETURN CODE
  OC USERETCD,USERETCD RC=0? -- WE EXPECT FINISH.
  BNZ ERROR NO, WE HAVE AN ERROR.
SPACE 2
* WE'RE ALL DONE
ERROR DS 0H
FINISH DS 0H
B RETURN
SPACE 3
DMP0 DC 'IMUP'
DMP1 DC 'SIGN'
DMP2 DC 'SEND'
DMP3 DC 'RECV'
DMPY DC 'YESS'
DMPN DC 'NONO'
SPACE
R0 EQU 0
R1 EQU 1
RETURN DS 0H
EXEC CICS DUMP FROM(COMREG) LENGTH(SNAPLEN) DUMPCODE(DMP0)
EXEC CICS RETURN
END

Sample CICS front-end COBOL application

IDENTIFICATION DIVISION.
  PROGRAM-ID. UDASCOBC.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
    01 WS-TEXT PIC X(30) VALUE 'WORKING STORAGE STARTS HERE'.
    01 ABEND-FIELDS.
      05 FORCE-ABEND PIC S9(8) USAGE COMP VALUE +1.
      05 FULL-ZERO PIC S9(8) USAGE COMP VALUE +0.

    01 UDAS-PARAMETERS.
      05 UDAS-BUFFER-ADDRESS USAGE POINTER.
      05 UDAS-BUFFER-LENGTH PIC S9(8) COMP.
      05 UDAS-SESSION-CODE PIC S9(8) COMP.
      05 UDAS-RETURN-CODE PIC S9(8) COMP.
      05 UDAS-RETURN1 REDEFINES UDAS-RETURN-CODE.
        10 UDAS-RETURN-ALPHA PIC X(4).
      05 UDAS-RETURN2 REDEFINES UDAS-RETURN-CODE.
        10 UDAS-RETURN-FIRST PIC X(1).
        10 FILLER PIC X(3).

    01 UDAS-USER-BUFFER.
      05 BUFFER-1 PIC X(40).
      05 BUFFER-2 PIC X(10000).
      05 BUFFER-3 PIC X(10000).
      05 BUFFER-4 PIC X(1960).

    01 FIRST-BUFFER.
      05 TASK-CODE PIC X(9) VALUE 'UCFTEST1 '.
      05 SAMPLE-DATA PIC X(20) VALUE 'SAMPLE DATA 12345678'.
      05 FILLER PIC X(11) VALUE SPACES.

    01 ALL-DONE.
      05 FILLER PIC X(8) VALUE 'ALL DONE'.
      05 FILLER PIC X(32) VALUE SPACES.

    01 COUNTERS
      * LOOP-COUNTER CAN BE ADJUSTED TO SEQUENTIALLY
      * START AND END MULTIPLE BACK-END TASKS.
      * EJECT
        05 LOOP-MAX PIC 9(4) VALUE 2.
        05 LOOP-COUNTER PIC 9(4) VALUE 0.
      MESSAGE
        01 MESSAGES
          * LOOP-COUNTER CAN BE ADJUSTED TO SEQUENTIALLY
* START AND END MULTIPLE BACK-END TASKS.

05 OUTPUT-MESSAGE PIC X(70) VALUE SPACES.
05 MSGLEN PIC S9(4) USAGE COMP VALUE +70.
05 SUCCESS-MESSAGE.
  10 LOOP-COUNTER-SUCC PIC 9(4) VALUE 0.
  10 FILLER PIC X(66) VALUE
      ' SUCCESSFUL UDASCOBC CONVERSATIONS COMPLETED'.
05 BEGIN-MESSAGE.
  10 FILLER PIC X(26) VALUE
      ' BEGINNING CONVERSATION #'.
  10 LOOP-COUNTER-BEGIN PIC 9(4) VALUE 0.
05 BAD-SIGNON-MESSAGE.
  10 FILLER PIC X(50) VALUE
      ' BAD RETURN CODE FROM SIGNON'.
05 BAD-SEND-MESSAGE.
  10 FILLER PIC X(50) VALUE
      ' BAD RETURN CODE FROM SEND'.
05 BAD-RECEIVE-MESSAGE.
  10 FILLER PIC X(50) VALUE
      ' BAD RETURN CODE FROM RECEIVE'.
05 BAD-SIGNOFF-MESSAGE.
  10 FILLER PIC X(50) VALUE
      ' BAD RETURN CODE FROM SIGNOFF REQUEST'.
05 BUFFER-LENGTH-MESSAGE.
  10 FILLER PIC X(14) VALUE 'BUFFER LENGTH '.
  10 BEFORE-AFTER PIC X(6).
  10 FILLER PIC X(7) VALUE ' CALL: '.
  10 UDAS-BUFFER-LENGTH-DISPLAY PIC S9(4).
  10 FILLER PIC X(39) VALUE SPACES.
05 BUFFER-CONTENTS-MESSAGE.
  10 FILLER PIC X(26) VALUE 'PARTIAL BUFFER CONTENTS : '.
  10 BUFFER-1-DISPLAY PIC X(40).
  10 FILLER PIC X(4) VALUE SPACES.
05 UDAS-RETURN-MESSAGE.
  10 FILLER PIC X(23) VALUE 'RETURN CODE FROM UDAS : '.
  10 UDAS-DISPLAY-NUMERIC PIC 9(8).
  10 UDAS-DISP-COMPOSITE REDEFINES UDAS-DISPLAY-NUMERIC.
    15 UDAS-DISPLAY-ALPHA PIC X(4).
    15 FILLER PIC X(4).
  10 FILLER PIC X(31) VALUE SPACES.

EJECT

PROCEDURE DIVISION.
MAIN SECTION.
* In this sample program, UDAS-PARAMETERS and
* UDAS-USER-BUFFER are coded in the WORKING STORAGE
* SECTION. Alternatively, either or both 01 level
* fields could be in the LINKAGE SECTION if
* a) The storage is allocated and passed to
*    UDASCOBC from a higher level program or
* b) The storage is allocated via an EXEC CICS
*    GETMAIN with the SET(ADDRESS OF ...) parameter.
* Note that some compilers do not allow the ADDRESS OF
* special register for WORKING STORAGE fields.
* One of the techniques above is necessary if your compiler
* does not allow the following statement:
*    SET UDAS-BUFFER-ADDRESS TO ADDRESS OF UDAS-USER-BUFFER.
*    PERFORM DO-CONVERSATIONS UNTIL
*    LOOP-COUNTER = LOOP-MAX.
*    MOVE LOOP-COUNTER TO LOOP-COUNTER-SUCC
*    MOVE SUCCESS-MESSAGE TO OUTPUT-MESSAGE
*    PERFORM DISPLAY-MESSAGE.
*    PERFORM SEND-MESSAGE.
*    GOBACK.
* DO-CONVERSATIONS SECTION.
  ADD 1 TO LOOP-COUNTER.
  MOVE LOOP-COUNTER TO LOOP-COUNTER-BEGIN
  MOVE BEGIN-MESSAGE TO OUTPUT-MESSAGE
  PERFORM DISPLAY-MESSAGE.
* ESTABLISH COMMUNICATION WITH UDAS.
MOVE 0 TO UDAS-SESSION-CODE.
PERFORM CALL-UDAS.
IF UDAS-RETURN-CODE NOT = 4
  MOVE BAD-SIGNON-MESSAGE TO OUTPUT-MESSAGE
  PERFORM DISPLAY-MESSAGE
  PERFORM ABORT.
*  SEND FIRST BUFFER WITH TASK CODE AND SOME SAMPLE DATA.
  MOVE FIRST-BUFFER TO BUFFER-1.
  MOVE 29 TO UDAS-BUFFER-LENGTH.
  PERFORM CALL-UDAS.
  *  WE EXPECT THE BACK-END TO RETURN DATA SO WE SHOULD HAVE
  *  A "RECEIVE" REQUEST FROM UDAS. KEEP RECEIVING DATA
  *  AS LONG AS BACK-END SENDS IT.
  IF UDAS-RETURN-CODE NOT = 8
    MOVE BAD-SEND-MESSAGE TO OUTPUT-MESSAGE
    PERFORM DISPLAY-MESSAGE
    PERFORM ABORT.
  PERFORM RECEIVE-DATA UNTIL UDAS-RETURN-CODE NOT = 8.
  *  IF WE DON'T GET A RECEIVE REQUEST, WE SHOULD GET A
  *  SEND REQUEST, I.E., THE BACK-END IS TRYING TO "READ"
  *  DATA.
  IF UDAS-RETURN-CODE NOT = 4
    MOVE BAD-RECEIVE-MESSAGE TO OUTPUT-MESSAGE
    PERFORM DISPLAY-MESSAGE
    PERFORM ABORT.
*  WE COULD SEND MORE DATA AT THIS POINT, BUT THIS IS
*  ENOUGH FOR TEST PURPOSES. JUST SEND AN INDICATOR
*  TO THE BACK-END THAT IS TIME TO CLEAN UP.
  MOVE ALL-DONE TO BUFFER-1.
  MOVE 8 TO UDAS-BUFFER-LENGTH.
  PERFORM CALL-UDAS.
  IF UDAS-RETURN-CODE NOT = 0
    MOVE BAD-SIGNOFF-MESSAGE TO OUTPUT-MESSAGE
    PERFORM DISPLAY-MESSAGE
    PERFORM ABORT.
ABORT SECTION.
*  APPROPRIATE ERROR HANDLING CAN BE PERFORMED HERE.
*  FOR TEST PURPOSES WE WILL ABEND THE TASK WITH
*  A CODE OF 'UDAS'.
  EXEC CICS ABEND ABCODE('UDAS') CANCEL END-EXEC.
RECEIVE-DATA SECTION.
*  RECEIVE A BUFFER SENT FROM THE BACK-END.
  MOVE SPACES TO BUFFER-1.
  MOVE 32000 TO UDAS-BUFFER-LENGTH.
  PERFORM CALL-UDAS.
CALL-UDAS SECTION.
*  CALL UDASCIC AFTER PUTTING ADDRESS OF BUFFER IN
*  UDAS PARAMETER BLOCK.
  MOVE UDAS-BUFFER-LENGTH TO UDAS-BUFFER-LENGTH-DISPLAY.
  MOVE 'BEFORE' TO BEFORE-AFTER.
  MOVE BUFFER-LENGTH-MESSAGE TO OUTPUT-MESSAGE.
  PERFORM DISPLAY-MESSAGE.
  MOVE BUFFER-1 TO BUFFER-1-DISPLAY
  MOVE BUFFER-CONTENTS-MESSAGE TO OUTPUT-MESSAGE.
  PERFORM DISPLAY-MESSAGE.
EXEC CICS LINK PROGRAM ('UDASCIC')
COMMAREA (UDAS-PARAMETERS)
LENGTH (LENGTH OF UDAS-PARAMETERS) END-EXEC.
  MOVE SPACES TO UDAS-DISP-COMPOSITE.
IF UDAS-RETURN-FIRST = LOW-VALUES
  MOVE UDAS-RETURN-CODE TO UDAS-DISPLAY-NUMERIC
  MOVE UDAS-RETURN-MESSAGE TO OUTPUT-MESSAGE
  PERFORM DISPLAY-MESSAGE
ELSE
  MOVE UDAS-RETURN-ALPHA TO UDAS-DISPLAY-ALPHA
  MOVE UDAS-RETURN-MESSAGE TO OUTPUT-MESSAGE
  PERFORM DISPLAY-MESSAGE.
  MOVE UDAS-BUFFER-LENGTH TO UDAS-BUFFER-LENGTH-DISPLAY.
  MOVE 'AFTER ' TO BEFORE-AFTER.
  MOVE BUFFER-LENGTH-MESSAGE TO OUTPUT-MESSAGE.
  PERFORM DISPLAY-MESSAGE.
Move Buffer-1 to Buffer-1-Display
Move Buffer-Contents-Message to Output-Message.
Perform Display-Message.

Display-Message Section.
* Following command will display info to the
* CSMT (MSGUSR) file.
EXEC CICS WRITEQ TD QUEUE('CSMT')
FROM (OUTPUT-MESSAGE) LENGTH (MSGLEN) END-EXEC.

Send-Message Section.
* Following command will display info to the
* terminal. It can be commented out for
* stress testing purposes.
EXEC CICS SEND FROM (OUTPUT-MESSAGE) LENGTH (MSGLEN)
ERASE WAIT END-EXEC.

Batch Front-ends

The interfaces for BATCH front-end UDAS applications use the #UDASBCH macro to create UDAS front-end programs.

Calling conventions to UDAS

The following calling conventions are used to link to UDAS from the front-end application program:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>User parameter block</td>
</tr>
<tr>
<td>R13</td>
<td>18-word save area</td>
</tr>
<tr>
<td>R14</td>
<td>Return address</td>
</tr>
<tr>
<td>R15</td>
<td>V(UDAS front-end entry point)</td>
</tr>
</tbody>
</table>

Calling the UDAS front-end from an Assembly language program

To call the UDAS front-end for a request, use the Assembly language instructions below:

LA R1,USERPARM
L R15,=V(udasb)
BALR R14,R15

where *udasb* is the entry point created when assembling #UDASBCH.

⚠️ **Note:** R1 points directly to USERPARM when calling the standard #UDASBCH entry point. If calling the special entry point UDASBCHY, R1 points to a fullword that contains the address of USERPARM.

⚠️ For sample JCL and a complete sample program, see member UDASBCHA in sample JCL.

Calling the UDAS front-end from a COBOL program

To call the UDAS front-end for a request, use the COBOL instructions below:

CALL ‘UDASBCHY’ USING *udas-parameters*
where *udas-parameters* is the 01 level structure that matches the USERPARM DSECT which is shown in Flow of Control (see page 95).

**Note:** For sample JCL and a complete sample program, see member UDASBCHC in sample JCL.

### How to create a front-end module

To create the UDAS BATCH front-end module, assemble a #UDASBCH macro and link the resulting object as described in the following sections.

#### UDASBCH Syntax

To create the UDAS BATCH front-end module, assemble a #UDASBCH macro and link the resulting object as described.

```
label #UDASBCH

SYSTEM  =  BATCH
          front-end-system-id

OS=  OS  BUFSIZ=  8000
     DOS

buffer-size
```

#### UDASBCH Parameters

- **label**
  The macro label provides the entry point name for the assembled module. The default is UDASBCH

- **SYSTEM**
  The rules for the front-end system name are the same as those described in UCF Operations for macro #UCFUFT.

- **OS=OS|DOS**
  Defines the operating system environment. Use OS for z/OS operating systems and DOS for z/VSE operating systems.

- **BUFSIZ=buffer-size**
  Specifies the size of the buffer that will be allocated to pass data between the front-end and the back-end. Buffer-size must be at least as big as the largest output buffer length specified by either the front-end or back-end programs. The default is 8000.

### Creating a UDAS Batch front-end interface program

Use the sample JCL in member UDASBCHI in sample JCL.
Advanced-Program-to-Program Communications

Contents

- Real APPC (see page 110)
- Emulated APPC (see page 112)

You can achieve non-terminating task data transfer, as described above for UCF programs, using APPC (advanced-program-to-program communications). APPC is a protocol that allows user-written programs on different systems to communicate with one another. Real APPC is implemented through the Systems Network Architecture (SNA) logical unit (LU) 6.2 protocol. DC/UCF uses the VTAMLU line type to support real APPC.

CA IDMS support for APPC

Within the CA IDMS environment, APPC is supported through two languages:

- Assembler, using the #TREQ DML statement
- CA ADS

You can use real APPC for communications between the following programs:

- CA ADS to CA ADS programs on the mainframe
- DC to DC programs
- DC to CICS programs
- DC to IBM System 36 or System 38
- DC to any LU 6.2 physical unit
- DC to PC programs

⚠️ Note: For more information on APPC communications using Assembler, see the #TREQ statement in the CA IDMS DML Reference section for Assembler. For more information on APPC communications in CA ADS, see the CA ADS Reference section.

Emulated APPC

CA IDMS supplies emulation software for the PC that allows CA ADS dialogs executing on a PC with a 3270 emulator card to emulate APPC. DC/UCF supports emulated APPC through the LAPPCEMU line type.
Real APPC

To provide support for real APPC in a DC/UCF system, do the following:

- Add the appropriate physical and logical terminal definitions to the VTAMLU line definition in the DC/UCF system definition
- Have your VTAM system programmer code a VTAM mode table entry to define the SNA protocols (that is, the bind parameters) to be used for sessions with other type 6.2 logical units

More Information

- For the complete syntax for the LINE, PTERM, and LTERM statements used to define SNA logical units in a DC/UCF system, see the discussion of VTAMLU device definition in the CA IDMS Administrative section.
- For more information on VTAM mode table entries for SNA logical units, see the CA IDMS Administrative section.

PTERM statement

The PTERM statement used to define a physical terminal for real APPC must specify the following parameters:

- TYPE IS LU
- NOACQUIRE (default)
- NAME IS vtam-node-name
- INFMH IS NO (default)
- OUTFMH IS NO (default)
- CONTENTION IS WINNER
- NOHOLD (default)
- LIMIT ON INPUT IS 0 (default)
- MODEENT IS vtam-modeent-name
- NORELEASE
- SYNCLEVEL IS OFF (default)

Note:

- vtam-node-name must match the name of a VTAM minor node (logical unit) specified in the VTAM definition.
- vtam-modeent-name must match the name specified in the LOGMODE parameter of the VTAM mode table entry for the SNA logical unit used for APPC.

VTAM mode table entry

The VTAM mode table entry for the logical unit used for APPC must specify the following parameters:
LOGMODE=vtam-modeent-name
TYPE=0
PSNDPAC=X'03'
SSNDPAC=X'03'
SRCPVAC=X'03'
FMPROF=X'13'
TSPROF=X'07'
PRIPROT=X'B0'
SECPROT=X'B0'
COMPROT=X'50B1'
RUSIZE=X'8585'
PSERVIC=X'060200000000000000102F00'

Note: vtam-modeent-name must match the name specified in the MODEENT parameter of a PTERM statement used to define a physical terminal for APPC.

Sample system generation statements

The following DC/UCF system generation statements define two terminals to be used for real APPC:

ADD LINE SNALU1
  TYPE IS VTAMLU
  APPLICATION IDENTIFICATION IS IDMSSNA
  ENABLED.

ADD PTERM LU001
  TYPE IS LU
  NAME IS LU620001
  CONTENTION IS WINNER
  MODEENT IS SNAAPPC1
  ENABLED.

ADD LTERM LU001
  ENABLED.

ADD PTERM LU002
  TYPE IS LU
  NAME IS LU620001
  CONTENTION IS WINNER
  MODEENT IS SNAAPPC1
  ENABLED.

ADD LTERM LU002
  ENABLED.

Sample VTAM mode table entry

The following VTAM mode table entry defines the bind parameters for APPC:

MODEENT
  LOGMODE=SNAAPPC1,
  TYPE=0,
  PSNDPAC=X'03',
  SSNDPAC=X'03',
  SRCPVAC=X'03',
  FMPROF=X'13',
  TSPROF=X'07',
  PRIPROT=X'B0',
  SECPROT=X'B0',
  COMPROT=X'50B1',
  RUSIZE=X'8585',
  PSERVIC=X'060200000000000000102F00'
Emulated APPC

To provide support for emulated APPC in a DC/UCF system, add the appropriate line, physical terminal, logical terminal, program, and task definitions to the DC/UCF system definition.

Note: For complete LINE, LTERM, PROGRAM, PTERM, and TASK statement syntax, see the CA IDMS Administrating section.

LINE statement

The LAPPCEMU line type provides emulated APPC support. You should define only one such line in a DC/UCF system.

The following is the device-specific syntax for the LAPPCEMU line type:

```
TYPE is LAPPCEMU
```

PTERM statement

The physical terminal associated the LAPPCEMU line type must be defined as type PAPPCEMU. You should associate only one physical terminal with an LAPPCEMU line.

The following is the device-specific syntax for the PAPPCEMU physical terminal type:

```
TYPE is PAPPCEMU
```

LTERM statement

The LTERM statement used to define the logical terminal associated with a PAPPCEMU physical terminal must specify ULOW.

Sample line and terminal definitions

The following DC/UCF system generation statements define a line and associated physical terminal /logical terminal pair to support emulated APPC:

```
ADD LINE APPCLIN
   TYPE IS LAPPCEMU
   ENABLED.

ADD PTERM APPCPTE
   TYPE IS PAPPCEMU
   ENABLED.

ADD LTERM APPCLTE
   ULOW
   ENABLED.
```

PROGRAM and TASK statements

Include the following PROGRAM and TASK statements in the DC/UCF system definition to define the APPC emulation startup program and task:
TCP/IP Support

This section describes how to setup and manage TCP/IP support within CA IDMS. TCP/IP is supported in CA IDMS systems running in the z/OS, z/VSE, and z/VM environments.

The following topics, as well as the changes they require for their execution, are discussed in details in the next sections:

- DNS socket functions -- The GETHOSTBYADDR and GETHOSTBYNAME DNS socket functions supported by the operating system (z/OS and z/VSE) or by our own CA IDMS DNS Resolver (z/VSE and z/VM); z/VSE users can choose between the two options.

- Services socket functions -- The GETSERVBYNAME and GETSERVBYPORT services socket functions are supported by our own CA IDMS Services Resolver on all operating systems.

- DNS + Services socket functions -- The GETADDRINFO and GETNAMEINFO socket functions are supported by the operating system (z/OS) or by our own CA IDMS DNS and Services Resolvers (z/VSE and z/VM).

- Establishing TCP/IP Support (see page 114)
  - Prerequisites (see page 114)
  - Updating the Startup JCL (see page 115)
  - Modifying the Sysgen (see page 117)
- Managing TCP/IP Support (see page 117)
- Supporting DNS Functions Using the SYSTCPD File (see page 118)
  - CA IDMS DNS Resolver (see page 118)
- Supporting Services Functions Using a Services File (see page 120)
  - Port Number Independence (see page 120)
  - CA IDMS Services Resolver (see page 120)
  - Service Resolver Syntax (see page 121)
  - Service Resolver Parameter (see page 121)
Establishing TCP/IP Support

Setting up TCP/IP support within CA IDMS requires the following steps. Each step describes the actions required for each operating system.

- **Prerequisites**
- Changes needed to the system startup JCL
- Modifications needed to the sysgen dictionary

**Prerequisites**

This section contains the prerequisites for setting up TCP/IP support in z/OS, z/VSE, and z/VM operating systems.

**z/OS**

On z/OS, support for TCP/IP is implemented using UNIX System Services kernel functions. The userid assigned to central version should be granted access to these functions. Depending on your security environment, this might require defining an OMVS segment for the userid. For more information, refer to the appropriate security documentation.

**z/VSE**

On z/VSE, the CA IDMS TCP/IP implementation supports TCP/IP stacks from Connectivity Systems Incorporated (CSI) and Barnard Software Incorporated (BSI). For details regarding the required JCL for their respective BSD/C API support, refer to appropriate documentation.

**Link the RHDCT1IP module**

A DOST1IP.OBJ module is delivered with the product. It must be linked with object modules delivered by CSI or BSI as phase RHDCT1IP.

*Linking RHDCT1IP for CSI*

When using TCP/IP from CSI, the following INCLUDEs are required to link the RHDCT1IP module. IPNRxxxx modules should get autolinked.

```
// LIBDEF *,SEARCH=(tcpiplib.sublib,idsmslib.sublib)
// LIBDEF PHASE,CATALOG=idsmslib.sublib
// OPTION CATAL
PHASE RHDCT1IP,*
INCLUDE DOST1IP
ENTRY T1IPEP1
/*
// EXEC LNKEDT
/*
```

*Linking RHDCT1IP for BSI*


When using TCP/IP from BSI, the following INCLUDEs are required to link the RHDCT1IP module. The IPNRxxxx entry points should get resolved in BSI module BSTTENVR.

```
// LIBDEF *,SEARCH=(tcpiplib.sublib,idmslib.sublib)
// LIBDEF PHASE,CATALOG=idmslib.sublib
// OPTION CATAL
PHASE RHDCT1IP,*
INCLUDE DOST1IP
INCLUDE BSTTENVR
ENTRY T1IPEP1
/*
// EXEC LNKEDT
/*
```

**Updating the Startup JCL**

The CA IDMS startup JCL must be updated to include the definition of the files/data sets needed for the support of the DNS and services socket functions. The DNS socket functions require a SYSTCPD file. The support of the services socket functions in CA IDMS is optional. To enable services socket functions, define the filename/ddname associated with the services file in the SERVICES FILE IS services-file-name clause of the TCP/IP sysgen entity.

**More Information**

- For more information on the TCP/IP sysgen entity, see the *Administrating section*.

- For more information on the services file and its layout, see *Supporting Services Functions Using a Services File*.

- For more information on the SYSTCPD file and its layout, see *Supporting DNS Functions Using the SYSTCPD File*.

**z/OS**

- To support the DNS socket functions, a SYSTCPD card must be added to the JCL. This card is needed if either of the following is true:
  - The TCP/IP run time is from IBM and the TCP/IP installation specified a prefix different from the default ("TCP/IP") prefix:
    ```
    //SYSTCPD DD DISP=SHR,DSN=prefix.TCPIP.DATA
    ```
    where `prefix` should be replaced by the TCP/IP installation prefix.
  - The TCP/IP implementation is CA TCPaccess CS for z/OS. For more information, see the *CA TCPaccess Communications Server for z/OS Customization Guide*.

Contact your systems programmer to obtain this information. See the documentation from the TCP/IP vendor for information on the SYSTCPD content.

- To support the services socket functions, add the services file to the startup JCL as follows:
  ```
  //services-file-name DD DISP=SHR,DSN=services-dsname
  ```

  **services-file-name**
  Specifies the value assigned to the SERVICES FILE clause in sysgen.
services-dsname
Specifies the name of the data set containing the services file definitions. The default services data set is cataloged as <tcpip-hlq>.ETC.SERVICES.

z/VSE

Regardless of the TCP/IP stack implementation, if the default stack id is not '00', the following JCL statement is required:

```
// OPTION SYSPARM='nn'       Set default stack ID to nn
```

To support the DNS socket functions, you must choose one of the following implementations:

- CA IDMS -- An advantage of using the CA IDMS DNS Resolver is its ability to return all alias names and IP addresses that are defined to the host name or host IP address. The support of the GETADDRINFO and GETNAMEINFO socket functions requires the CA IDMS DNS Resolver.

- BSI or CSI -- An advantage of using the BSI/CSI DNS implementation is the GETHOSTBYNAME and GETHOSTBYADDR functions return names that are defined to the local TCP/IP stack. However, only the primary name for an IP address or primary IP address associated with the host name is returned.

To select the BSI/CSI DNS resolver, do not code any SYSTCPD file; to select the CA IDMS internal DNS resolver, code the following SYSTCPD file:

```
// DLBL SYSTCPD,'tcpip.tcpip.data',,SD
// EXTENT SYS001,vvvvv
// ASSGN SYS001,DISK,VOL=vvvvv,SHR
```

```
tcip.tcpip.data
```
Specifies the file-id of the file.

To support the services socket functions, add the definition for the services file as follows:

```
// DLBL 'services-file-name','services-file-id',,SD
// EXTENT SY5001,vvvvv:// ASSGN SY5001,DISK,VOL=vvvvv,SHR
```

- services-file-name
  Specifies the value assigned to the SERVICES FILE clause in sysgen.

- services-file-id
  Specifies the file-id of the file.

z/VM

To support the DNS socket functions, add the definition for the SYSTCP file to the startup EXEC as follows:

```
FILEDEF SYSTCPD DISK fn ft fm
```

```
fn ft fm
```
Specifies the FILE ID of the SYSTCPD file. The default value is TCPIP DATA.

To support the services socket functions, add the definition for the services file to the startup EXEC as follows:

```
FILEDEF services-file-name SYSTCPD DISK fn ft fm
```
services-file-name
Specifies the value assigned to the SERVICES FILE clause in sysgen.

fn ft fm
Specifies the FILE ID of the services file.

Modifying the Sysgen

The following are the modifications to make to the sysgen:

- Define the TCP/IP entity in sysgen. The DEFAULT STATUS clause must be defined to ON to enable TCP/IP support in CA IDMS.

- If generic listening or DDS using TCP/IP has to be used in the DC/UCF system, a SOCKET type line must be defined. For each generic listener, define a LTERM/PTERM pair and its associated task code and program. For each DDS destination, define a LTERM/PTERM pair and its associated parameters.

- If generic listening is used, define also a number of BULK PTERMs. The system generation syntax permits you to define one pair, and the REPEAT clause of the PTERM statement can be used to facilitate the definition of multiple PTERMs and LTERMs. You determine the number of BULK PTERM/LTERM pairs as follows:
  - Each connection accepted by a generic listener uses one BULK PTERM/LTERM pair.
  - A server task started by a generic listener can exploit pseudo-conversational programming. This means that the TCP/IP connection remains open, and the BULK PTERM/LTERM pair is without an associated active task.

Note: For more information on the system generation statements, see the Administrating section.

Managing TCP/IP Support

The following DCMT commands are enhanced to help you manage the TCP/IP environment:

- DCMT DISPLAY TCP/IP -- Displays information on the TCP/IP runtime environment of a DC/UCF system.

- DCMT VARY TCP/IP -- Allows you to dynamically enable and disable TCP/IP support or to dynamically alter the value of all the different TCP/IP parameters defined to sysgen.

- DCMT VARY PTERM -- Enables you to manage the generic listener service and the DDS definitions dynamically.
Supporting DNS Functions Using the SYSTCPD File

This section describes the support of the DNS functions using the SYSTCPD file for each of the following operating systems:

- On z/OS, the DNS socket functions (GETHOSTBYADDR and GETHOSTBYNAME) are directly supported in the operating system dependent TCP/IP interface used by the #SOCKET macro interface. The SYSTCPD file is used internally by the operating system to process the DNS requests.

- On z/VSE, the SYSTCPD file is optional. If the SYSTCPD file is coded, the DNS socket functions are resolved using the CA IDMS DNS Resolver.

- On z/VM, the DNS socket functions are not supported by the operating system interface. Therefore, an internal DNS Resolver is implemented in CA IDMS. The resolver communicates with a name server to retrieve the requested DNS information.

CA IDMS DNS Resolver

The following information applies to z/VM and optionally z/VSE.

When support of TCP/IP is enabled in the DC/UCF system (during startup or the execution of a DCMT VARY TCP/IP STATUS ON command), the following occurs:

1. The file associated with the SYSTCPD card is read.
2. The DNS specific parameters are parsed.
3. The CA IDMS Resolver is configured.

Coding the SYSTCPD file

The following syntax rules apply for this file:

- All records starting with a ';' character are treated as comments.
- Blanks and <end-of-line> characters delimit the tokens.
The format for each configuration statement is: Keyword Value.

The following table lists the SYSTCPD parameters:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Default Value</th>
<th>Range of values</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMAINORIGIN</td>
<td></td>
<td>Max 64 chars</td>
<td>Suffix that is appended to a hostname that doesn’t contain any dots</td>
</tr>
<tr>
<td>NSINTERADDR</td>
<td></td>
<td></td>
<td>Up to 4 different NSINTERADDR input lines pointing to different DNS servers</td>
</tr>
<tr>
<td>NSPORTADDR</td>
<td>53</td>
<td>1 to 65535</td>
<td>Port of the name server</td>
</tr>
<tr>
<td>RESOLVEVIA</td>
<td>UDP</td>
<td></td>
<td>Protocol to use for the communication. Only TCP is supported on z/VSE.</td>
</tr>
<tr>
<td>RESOLVERTIMEOUT</td>
<td>30</td>
<td>1 to 65535</td>
<td>Time in seconds that the resolver waits for a response from the server</td>
</tr>
<tr>
<td>RESOLVERUDPRETRIES</td>
<td>1</td>
<td>1 to 65535</td>
<td>Number of times the resolver tries to communicate with the name server (when RESOLVEVIA is UDP only)</td>
</tr>
<tr>
<td>TCPIPUSERID</td>
<td>TCPIP</td>
<td>Max 8 chars</td>
<td>Userid of the default TCP/IP virtual machine.</td>
</tr>
</tbody>
</table>

Note: z/VM systems only. If no INCLUDE list of TCP/IP stacks is provided in the TCP/IP sysgen entity, then this parameter defines the userid of the TCP/IP virtual machine that is used as the default TCP/IP stack by CA IDMS. If an INCLUDE list of stacks is provided, this parameter is ignored. For more information on INCLUDE stacks list, see the TCP/IP entity in the CA IDMS Administrating section.

Sample SYSTCPD

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMAINORIGIN</td>
<td>CA.COM</td>
</tr>
<tr>
<td>NSINTERADDR</td>
<td>172.24.255.255</td>
</tr>
<tr>
<td>NSPORTADDR</td>
<td>53</td>
</tr>
<tr>
<td>RESOLVEVIA</td>
<td>TCP</td>
</tr>
<tr>
<td>RESOLVERTIMEOUT</td>
<td>3</td>
</tr>
<tr>
<td>RESOLVERUDPRETRIES</td>
<td>1</td>
</tr>
</tbody>
</table>

z/VSE SYSTCPD File

The following rules apply for the SYSTCPD file:

- Must contain 80-byte records
- Can be blocked up to a block size to 32720

Important! Do not code the BLKSIZE= DLBL parameter.
z/VM SYSTCPD File

The following rules apply for the SYSTCPD file:

- Can be fixed or variable length
- Can use the default file, TCPIP DATA *, or the user can create his own file with specific definitions

Supporting Services Functions Using a Services File

This section describes translating service names to and from port numbers using the services file.

Port Number Independence

CA IDMS provides the ability for DC/UCF systems and applications that execute within those systems to use logical service names in place of port numbers. The use of logical names allows port numbers to be changed without impacting application code or system definitions.

Port number independence is provided through the following facilities:

- A CA IDMS Services Resolver that translates a service name into a port number or vice versa.
- The ability to specify a service name in place of a port number on LISTENER and DDSTCPIP PTERM SYSGEN statements.
- The ability to specify or change a service name using a DCMT VARY PTERM statement.
- The ability to display the service name for a PTERM using a DCMT DISPLAY PTERM statement.
- Socket functions (GETSERVBYNAME and GETSERVBYPORT) for returning a port number associated with a service name or the service name(s) associated with a port number.

The socket functions are described in the CA IDMS Reference section. The remainder of the facilities are described below.

CA IDMS Services Resolver

The CA IDMS Services Resolver is responsible for translating service names to and from port numbers. It does this using the contents of a services file that can be included in the execution JCL of a DC/UCF system. The DDNAME or filename of the services file is specified in the TCP/IP sysgen entity.

The services file contains a list of service names together with the port number, protocol and optional alias names associated with each one. This information is used when servicing a GETSERVBYNAME or GETSERVBYPORT socket function to retrieve the port number associated with a service name or the service name associated with a port number.

On z/OS, the default services data set is cataloged as <tcpip-hlq>ETC.SERVICES. You can use this data set, a customized data set containing the definitions of the services that you want to use, or a combination of the two by concatenating the two files together. To ensure that the customized entries take precedence, include the customized data set before the system default data set.
During the initialization of the TCP/IP environment in a CA IDMS system, the services file is read and its contents converted to an internal structure that is stored in memory for efficient access. The contents of the in-core structure can be refreshed from the services file using the DCMT VARY TCP/IP command.

Each record within a services file defines a service name entry in character format.

**Service Resolver Syntax**

The syntax of the services data set records accepted by the CA IDMS Services Resolver follows the general standards for such files. The format is as follows:

```
name - port#/protocol  alias  # comment
```

**Service Resolver Parameter**

- **name**
  Identifies the official Internet service name.

- **port#**
  Identifies the port number. Port number is a positive number between 1 and 65535.

- **protocol**
  Identifies the protocol used for the service.

- **alias**
  Identifies an unofficial name for the service.

- **# comment**
  Specifies comment text that continues until the end of the record.

**Services file**

The following rules apply for the services file:

- The record size of the services file is as follows:
  - On z/OS, it must be between 56 and 256.
  - On z/VSE, it must contain 80-byte records and can be blocked up to a block size to 32720. The BLKSIZE= DLBL parameter cannot be coded.
  - On z/VM, it can be fixed or variable length.

- Items in a record are separated by spaces or tabs.

- name, protocol, and alias are case sensitive.

- name must start in column 1.
Records with duplicate service names are ignored. The first definition of service name takes precedence.

- Maximum length for name and alias is normally 32 characters, but longer names are accepted on all systems.

- Comments begin with the number sign (#) character and continue until the end of record.

- The only protocols accepted in CA IDMS are TCP and UDP, in any combination of uppercase and lowercase.

The following illustrates a standard TCPIP.ETC.SERVICES data set delivered with the operating system:

```
#  
# Network services, Internet style  
#  
echo 7/tcp  
echo 7/udp  
discard 9/tcp sink null  
discard 9/udp sink null  
systat 11/tcp users  
daytime 13/tcp  
daytime 13/udp  
netstat 15/tcp  
qotd 17/tcp quote  
chargen 19/tcp ttytst source  
chargen 19/udp ttytst source  
ftp 21/tcp  
```

**Note:** To allow updating the services file while it is in use by a CV, make it a member of a partitioned data set rather than a sequential file.

**More Information:**

- For more information on the GETSERVBYNAME and GETSERVBYPORT socket functions, see the Reference section.

- For more information on the TCP/IP SYSGEN entity, see the Administrating section.

- For more information on the DCMT VARY TCP/IP command, see the System Tasks and Operator Reference section.

**Supporting the GETADDRINFO and GETNAMEINFO socket functions**

The GETADDRINFO and GETNAMEINFO socket functions implement a combination of DNS and services requests.

- On z/OS, these socket functions are supported by the operating system.
On z/VSE and z/VM, these socket functions are supported by the CA IDMS DNS and Services Resolvers. For more information on the CA IDMS DNS and Services Resolvers, see sections 4.3.3 and 4.3.4.

UCF Operations

This section describes the procedures for generating a UCF system. UCF enables a terminal connected to a host TP monitor to execute DC/UCF tasks. For the purposes of this discussion, a DC/UCF task can be a user task, a task associated with an online software component, or a system-supplied task.

Typically, UCF is used to do the following:

- Execute DC/UCF applications and host TP-monitor applications from the same terminal
- Execute DC/UCF applications under two or more DC/UCF systems at the same terminal (for example, in a production and test environment)
- Share data between an application executing under a DC/UCF system and an application executing under CICS or a different DC/UCF system.

TP-monitors supported

UCF supports the following TP monitors and terminal control facilities: CICS, CMS (z/VM), DC, and TSO. UCF also provides an interface to enable DC/UCF tasks to execute in batch mode.

UCF Operations Overview

UCF architecture

The UCF runtime system consists of a front-end system and a back-end system as follows:

- The UCF front-end is the host TP monitor. The front-end program executes as an application program within the host TP monitor and performs the following functions:
  - Communicates with the UCF terminals connected to it by using standard host TP-monitor terminal I/O facilities
  - Communicates with the back-end system by passing terminal requests to the back-end and receiving responses using the SVC
- The UCF back-end is a DC/UCF system that resides in its own region/partition. The back-end system performs the following functions:
- Receives terminal input from the UCF front-end and directs it to the appropriate application for processing
- Relays write-to-terminal requests from the application to the UCF front-end, which performs the write operation
- Handles requests for output to DC printers (printer support only)

**Line driver**

UCF is a line driver; for each DC/UCF system, there is one UCF line driver. When a terminal I/O request is issued, DC/UCF directs the request to the UCF line driver associated with the issuing terminal; the application program need not be concerned with the type of terminal on which it executes (unless basic mode terminal I/O is performed). Through UCF, the same DC/UCF program can execute under any of the host TP monitors supported (for example, CICS) and can communicate with the host system using terminals with any of the supported access methods (for example, BTAM or VTAM).

⚠️ **Note:** Sites with UCF and not DC can access terminals through the UCF line only. Sites with DC can access terminals through any supported access method, including UCF.

The following diagram illustrates UCF operations. In this diagram, the host TP monitor application database exists optionally. The DC terminals are available only to sites having DC; sites with UCF are permitted access through the UCF front-end terminals only.

**UCF Operations Overview**

**UCF security**
When the UCF front-end connects to the DC back-end, it automatically signs on to the DC back-end using the user ID supplied by the UCF front-end terminal. Therefore, you should define the user to the DC back-end system using the ID and password assigned on the UCF front-end system.

The remainder of this section presents procedures for creating UCF front-end and back-end systems and summarizes the DC/UCF system generation statements that must be included in the back-end system definition.

**UCF Front-End**

The UCF front-end system executes in the host TP-monitor region/partition and communicates with terminals through host TP-monitor I/O facilities. The front-end communicates with the back-end system through the external request-unit service (ERUS) facility.

UCF front-end systems can execute in dedicated or intermittent mode.

**Dedicated mode**

In dedicated mode, tasks execute as DC/UCF tasks and are invoked by the terminal operator in response to the ENTER NEXT TASK CODE prompt. While in dedicated mode, the terminal operator can execute DC/UCF tasks only; host TP-monitor tasks cannot be executed until dedicated mode is terminated.

To invoke dedicated mode, the terminal operator first enters the dedicated task code (or transaction id or verb, depending on the host TP monitor in use). This task code must be defined in the NTID parameter of the #UCFUFT macro (described in Front-end Table) as well as to the host TP monitor. Typically, only one dedicated task code exists (for example, DBDC). The terminal operator then signs on to the DC/UCF system.

To terminate dedicated mode and return to the host TP-monitor environment, the terminal operator enters the task code BYE or SUSPEND as follows:

- **BYE** terminates dedicated mode and disconnects the front-end terminal from the UCF back-end. This action frees all resources held by the back-end for the terminal (for example, logical and physical terminal elements, signon element). The next time the terminal operator invokes dedicated mode, the DC/UCF signon must be performed.

- **SUSPEND** terminates dedicated mode but does not disconnect the front-end terminal from the UCF back-end; all resources held by the back-end for the terminal are preserved. The terminal operator can invoke dedicated mode later in the session and execute DC/UCF tasks without signing on again. When using a DC front-end with a DDS access method (CCI, TCP/IP, or VTAM), the SUSPEND task code is not supported and is treated as a BYE task code.
Intermittent mode

In intermittent mode, tasks execute as host TP-monitor tasks and are invoked by the terminal operator in response to the host TP monitor’s equivalent of the ENTER NEXT TASK CODE prompt; DC/UCF is transparent. DC/UCF tasks that will be executed in intermittent mode must be defined to the host TP monitor and to the DC/UCF system with an intermittent task code. Any number of intermittent task codes can exist (for example, ADSG, DCMT, IDD, OLM).

When a DC/UCF task terminates on the back-end, the action taken depends on whether a back-end pseudo-converse exists and on the SUSPEND/DISCONNECT options in effect.

A back-end pseudo-conversation exists if the terminating task specifies a NEXT TASK CODE or an autotask is defined for the terminal. If a pseudo-conversation exists, the next task will be invoked in the normal manner (similar to dedicated mode).

If the terminating task does not specify a NEXT TASK CODE and no terminal autotask is defined, control will be returned to the front-end TP-monitor. The ENTER NEXT TASK CODE prompt is never written when running in intermittent mode.

By default, the DC/UCF system issues the equivalent of a SUSPEND when ENTER NEXT TASK CODE would have been written. The front-end terminal is not disconnected from the UCF back-end; all resources held by the back-end for the terminal are preserved. The terminal operator can invoke dedicated mode or intermittent mode again later in the TP-monitor session and execute DC/UCF tasks without signing on again.

Alternatively, the system can be directed to disconnect the terminal when the ENTER NEXT TASK CODE would have been written. If this option is desired, both of the following actions must be taken:

- Compile and link the DC/UCF options module RHDCOPTF specifying #DEFOPTF OPT00020.
- Compile and link the UCFCICS front-end module specifying #UCFCICS DISC=YES.

When this option is chosen, the DC/UCF system issues the equivalent of a BYE command when ENTER NEXT TASK CODE would have been written. The front-end terminal is disconnected from the UCF back-end. This action frees all resources held by the back-end for the terminal, for example, logical and physical terminal elements and signon element. The next time the terminal operator invokes a UCF task, a new DC/UCF signon will be performed.

**Note:** The UCF batch, z/VM, and TSO front-ends can execute in dedicated mode only.

UCF front-end components

The UCF front-end system consists of the following:

- The UCF front-end common module (RHDCUCFC) contains code common to all TP monitors and is distributed in object form with the UCF system.
The appropriate TP-monitor interface module is distributed in macro form with the UCF system and must be assembled at your site.

The front-end module contains TP-monitor-dependent procedures. The front-end module is created by assembling a macro appropriate to the TP monitor in use.

The front-end table defines the characteristics of the front-end system's terminals and supplies an identifier for the front-end system. The front-end table is created by assembling a series of macros.

The procedures for creating the front-end table and for defining and using each front-end module are presented separately below.

- **Front-end Table**
  - **Batch Front-end** (see page 133)
  - **CICS Front-end** (see page 138)
  - **CICS Abort Session Program** (see page 146)
  - **DC Front-end** (see page 151)
  - **TSO Front-end** (see page 153)

### Front-end Table

The front-end table defines global characteristics of a UCF front-end, including characteristics of the front-end's terminals. One front-end table must be created for each UCF front-end to be used, with the exception of the batch, z/VM, and TSO front-ends.

To create a UCF front-end table, assemble and link edit the following macros:

- **#UCFUFT** (see page 128) supplies the global characteristics for the front-end system (for example, system identifier).
- **#UCFUTD** (see page 129) identifies each terminal associated with the front-end system.
- **#UCFDEND** (see page 131) ends the front-end table definition.

### Front-end system and terminal identifiers

Each UCF front-end table contains an entry that identifies the front-end system and, optionally, an entry that identifies each terminal associated with the front-end system.

### System identifiers

Each front-end in a UCF system has an identifier that identifies to the DC/UCF system the TP monitor on which the front-end system task is executing. The system identifier (FESID) is used by the UCF back-end to determine if a front-end is authorized to access the back-end. DCMT functions use the FESID to display and vary information associated with a front-end.
When a host TP monitor (front-end) signs on to a DC/UCF system (back-end) through UCF, the front-end system passes the system identifier to DC/UCF. Before accepting the UCF signon, the DC/UCF system attempts to locate the identifier in its system table. If the identifier passed with the signon is not in the system table, the signon is rejected. See UCF Back-End for a discussion of the UCF system table.

Usually, the system identifier is specified in the front-end table with the #UCFUFT macro. However, because the batch, z/VM, and TSO front-ends have built-in front-end tables, the system identifier for these interfaces must be specified in the TP-monitor-dependent macro.

**Terminal identifiers**

Each terminal owned by a host TP monitor has a unique identifier. At runtime, the front-end task automatically passes the terminal identifier (FETID) to the DC/UCF system. The DC/UCF system uses the identifier with DCMT functions to display and vary information associated with that terminal.

The terminal identifier can be coded in the #UCFUTD macro for the terminal and/or in the PTERM statement for the terminal in the back-end system definition. At runtime, a terminal with no entry in the UCF front-end table (that is, one for which no identifier was coded in the #UCFUTD macro) will be assigned, by default, the characteristics of the prototype front-end terminal of the same type. If a prototype has not been defined for that terminal type, the terminal identifier must exist in the front-end table in order for a terminal to access UCF. Typically, you define prototypes for 3270- and TTY-type terminals; however, terminal identifiers must be supplied in the #UCFUTD macro for 3280-type terminals (printers).

The three macros used to create the UCF front-end table are described below, followed by an example of a front-end table, and the front-end table JCL for z/OS, z/VSE, and z/VM.

**#UCFUFT Macro**

The #UCFUFT macro defines global front-end characteristics. One #UCFUFT macro is required for each front-end table. #UCFUFT must be the first macro in the source file and must be labeled.

**Syntax**

```plaintext
label #UCFUFT

  MODE = CONV|PCONV

  .ENV = IBM

  .NTID = dedicated-task-code

  .PTID = print-task-code

  .SYSTEM = front-end-system-id
```

**Parameter**

- **MODE=CONV|PCONV**  
  (DC front-ends only) Specifies whether the front-end is to run in conversational or pseudo-conversational mode as follows:
- CONV -- The front-end will run in conversational mode.

- PCONV (default) -- The front-end will run in pseudo-conversational mode.

The MODE specification can be overridden for selected terminals with the #UCFUTD macro (described on the following pages).

- ENV=
  Defines the operating system environment. IBM is the default.

- NTID=
  Specifies the one- through eight-character task code used to invoke the front-end in dedicated mode. dedicated-task-code must match the task code specified in the host TP-monitor system definition.

- PTID
  (CICS and DC front-ends only) Specifies the one- through eight-character task code used to invoke the UCF Print Control Task. Print-task-code must match the task code specified in the CICS or DC system definition.

- SYSTEM
  Specifies the one- through eight-character identifier for the front-end system (FESID). front-end-system-id must be defined in the back-end system table and typically will be one of the following: CICS, DC, or TSO.

#UCFUTD Macro

The #UCFUTD macro defines terminal-specific characteristics. One #UCFUTD macro is required for each terminal associated with this front-end if no prototype terminals will be defined. If prototypes will be defined, one #UCFUTD macro must be coded for each type of terminal (that is, 3270 and TTY) to be used with the UCF front-end. Typically, only one prototype TTY terminal and one prototype terminal for each 3270-type model present need be defined. #UCFUTD macros must be unlabeled.

Syntax

```
#UCFUTD
  | BUFSIZ = buffer-size
  | .FORMFD = YES NO
  | .MODE = CONV PCONV
  | .MODEL = 2 model-number
  | .NCHARS = 80 line-length
  | .NLINES = 24 page-length
  | .PRTCLS = 1 printer-class
  | .SCS = YES
```
Parameter

- **TERM=**
  Specifies the unique identifier for the front-end terminal. Except when defining a 3280-type printer, this parameter is typically omitted and, therefore, defines a prototype terminal. `front-end-terminal-id` is a one- through eight-character alphanumeric value and must match the terminal identifier specified in the host TP-monitor system definition. This parameter is required if `TYPE=T3280` is specified.

- **TYPE=**
  Specifies the front-end terminal type as follows:
  - **CRT|TTY** -- The front-end terminal is a TTY-type terminal. TTY applies to all asynchronous ASCII terminals.
  - **T3277 (default)** -- The front-end terminal is a 3270-type terminal.
  - **T3279** -- The front-end terminal is a 3279-type color terminal with extended color support.
  - **T3280** -- The front-end terminal is a 3280-type terminal. If `T3280` is specified, the `TERM=` parameter (described above) must also be specified.

- **MODE=CONV|PCONV**
  (DC front-ends only) Specifies whether this terminal will execute in conversational or pseudo-conversational mode as follows:
  - **CONV** -- The terminal will execute in conversational mode.
  - **PCONV** -- The terminal will execute in pseudo-conversational mode.

  The `MODE=` parameter defaults to the `MODE` specification in the `#UCFUFT` macro.

- **MODEL=**
  (TYPE T3277 only) Specifies the 3270-type terminal model code. `model-number` is an integer in the range 1 through 5; the default is 2.

- **NCHARS=**
  Specifies the terminal's line length. `line-length` is an integer in the range 1 through 32767; the default is 80 characters.

- **NLINES=**
  Specifies the terminal's page length. `page-length` is an integer in the range 1 through 32767; the default is 24 lines. If `TYPE=TTY` is specified, a page length of 3 is recommended.
• **BUFSIZ=**
  Specifies the size, in bytes, of the I/O buffer allocated to the terminal. `buffer-size` can be any positive integer in the range of 1-32300. The value specified must be greater than or equal to the size of the longest data stream to be transferred between the DC/UCF system and the UCF front-end system. Typically, the default value (equal to the product of 1.5 times the `NCHARS` value times the `NLINES` value plus 800) is used.

• **FORMFD=YES|NO**
  (CICS and DC front-ends only) Specifies whether the printer front-end terminal has formfeed capabilities as follows:
  - **YES** -- The terminal has formfeed capabilities.
  - **NO (default)** -- The terminal does not have formfeed capabilities.

• **PRTCLS=**
  (CICS front-end only) Specifies a print class for `WRITE TO PRINTER` requests issued from the terminal. `print-class` is an integer in the range 1 through 64; the default is 1. This parameter is ignored if the terminal itself is a printer.

• **SCS IS NO|YES**
  Specifies whether the printer is an SNA character string device, also known as a type 1 logical unit:
  - **NO (default)** indicates that the printer is not an SNA character string device.
  - **YES** indicates that the printer is an SNA character string device.

### #UCFDEND Macro

The `#UCFDEND` macro specifies the end of the front-end table definition. One `#UCFDEND` macro must be coded for each front-end table. `#UCFDEND` must follow the last `#UCFUTD` macro and must be unlabeled.

**Syntax**

```
#UCEFEND

XA= YES NO
```

**Parameter**

- **XA**
  Determines whether the front-end table will have the capability to reside above the 16-megabyte line. The default, NO, indicates it does not have the capability; YES indicates it does.

### Sample Front-end Table Definition

The following macro statements define a sample CICS front-end system, consisting of a prototype TTY-type terminal and five 3270-type terminals:
Front-end Table JCL

This section contains the JCL to create the UCF front-end table for z/OS and z/VSE operating systems. Alternatively, you can assemble a front-end table in the same input stream that creates a particular front-end module. For example, a z/OS UCF CICS front-end object module can be created by assembling a #UCFCICS macro, a #UCFUFT macro, appropriate #UCFUTD macros, and a #UCFDEND macro. A particular UCF CICS front-end load module can be created by using the JCL shown in the section CICS Front-end JCL, and then omitting the INCLUDE statement for ucffet.

To create a stand-alone front-end table under z/OS:

1. Create a front-end table source module as follows:

   UCFFET  #UCFUFT  ucfuft-parameters
   #UCFUTD  ucfudt-parameters
   . . .
   #UCFDEND
   END

   2. Save the source module in your custom source library.

   3. Assemble and link it into your custom load library by executing the z/OS Assemble and Link-Edit (see page 465).

      Substitute the name of your front-end table source member and insert the following binder statement:

      - NAME ucffet(R)
      Name of the UCF front-end table load module.

To create a stand-alone front-end table under z/VSE:

1. Assemble and catalog the table using the sample JCL in z/VSE Assemble JCL (see page 476).

   Modify the JCL by substituting the following in place of the Assembler input statements:

   PUNCH 'CATALOG ucffet.OBJ REPLACE=YES'
   UCFFET  #UCFUFT  ucfuft-parameters
   #UCFUTD  ucfudt-parameters
   . . .
   #UCFDEND
   END

   - ucffet
   Specifies the name you choose for your UCF front-end table.
Batch Front-end

The UCF batch front-end enables the execution of DC/UCF tasks from a batch job.

To create the batch front-end module, assemble a #UCFBTCH macro and link edit the resulting object module with RHDCUCFC, CA IDMS and IDMSOPTI (optional). The batch front-end supports dedicated mode only and requires no front-end table definition.

The IDMSOPTI module identifies the back-end system with which the front-end communicates. The module is created by assembling an IDMSOPTI macro. If you omit IDMSOPTI from the link edit of the front-end system, the execution JCL for the front-end must include a SYSCTL file that identifies the back-end system. For more information on IDMSOPTI, see Setting Up Interpartition Communication and the SVC (see page 58).

#UCFBTCH Macro

The following is the syntax for the #UCFBTCH macro. The macro must be labeled; the label provides the module's entry point name.

Syntax

```
label #UCFBTCH

OS = OS
     DOS

.PRINT = YES
         NO

.RETCD = YES
         NO

.INPLEN = 80
         nn

.S015 = YES
        NO

.SYSTEM = BATCH
         front-end-system-id
```

Parameters

- **OS**
  Specifies the operating system under which the batch front-end system will execute. OS applies to all supported z/OS and z/VM operating systems. DOS applies to all supported z/VSE operating systems. The default is OS.

- **PRINT=**YES|NO
  Specifies whether reports queued to DC printers can be printed using UCF batch printer support. NO, the default, indicates that the system will create the batch front-end load module; printer support is not generated. YES directs the system to create a UCF print support module. UCF print support is discussed later under Front-end Table.
- **RETCD=YES|NO**
  If the CA IDMS system is shut down while a UCFBATCH job is running, the UCFBATCH job will end with a return code of 4 or 12. If RETCD=YES is specified, UCFBATCH jobs will continue to run as they currently do and will receive the non-zero return code. If RETCD=NO is specified, UCFBATCH jobs will not receive the return code of 4 or 12. The default is YES.

- **INPLEN=nn**
  Specifies the number of columns that UCF will process as input on each record within the input file. Columns following the specified length are ignored.
  **Default:** 80
  **Note:** If you use the default and the input file contains sequence numbers in columns 73 through 80, then you might receive syntax errors. To avoid the errors in this situation, specify INPLEN=72 instead of using the default.

- **S015=YES|NO**
  Specifying S015=YES or defaulting to S015=YES will not change the Current behavior of the UCF Batch front end. Coding S015=NO suppresses the following message:
  UCFBTCH S015 - NOTHING FOR PRINTER TO PRINT.

- **SYSTEM**
  Specifies the 1 through 8-character front-end system identifier. front-end-system-id must match the identifier specified in the FESID parameter of the #FESTENT macro used to define the batch front end in the UCF system table; see #FESTENT Macro for further information. The default of BATCH matches the FESID value for the batch front-end in the system table supplied with UCF.

**Front-end Load Module Assembly JCL**

This section contains the JCL to create the batch front-end load module for z/OS, z/VSE, and z/VM operating systems.

> **Note:** A default UCF batch interface module is provided during installation with the name RHDCUCFB. RHDCUCFB is created using macro input: #UCFBTCH SYSTEM=BATCH,OS=os

> If you want to create your own UCF batch front-end load module, use the following procedure.

**To create a z/OS front-end load module**

1. Create a UCFBTCH source module as follows:
   ```bash
   Ucfbttche #UCFBTCH #ucfbtch-parameters
   END
   ``
2. Save the UCFBTCH source module in your custom source library.
3. Assemble and link it into your custom load library by executing the z/OS Assemble and Link-Edit JCL (see page 465).

Substitute the name of your UCFBTCH source member and insert the following binder statements:

```
INCLUDE CUSTLIB(idmsopti)
ENTRY ucfbtche
```

**idmsopti**

(Optional) Specifies the name of your IDMSOPTI module.

**ucfbtche**

Specifies the entry point name of your UCF Batch front-end load module.

**ucfbtchi**

Specifies the name of your UCF batch front-end load module.

**To create a z/VSE batch front-end load module**

1. Assemble and catalog the module using the sample JCL in z/VSE Assemble JCL (see page 476). Modify the JCL by substituting the following in place of the Assembler input statements:

```
PUNCH 'CATALOG ucfbtcho.OBJ REPLACE=YES'
ucfbtche #UCFBTCH #ucfbtch-parameters
END
```

2. Link the UCF batch phase using the sample JCL in z/VSE Link JCL (see page 476). Modify the JCL by substituting the following statements in place of the Linkage editor control statements:

```
PHASE uucfbtchi,*
INCLUDE ucfbtcho
```

**ucfbtcho**

Specifies the name of your UCF batch front-end object module.

**z/VM batch front-end load module assembly**

```
GLOBAL MACLIB idmslib
FILEDEF TEXT DISK ucfbtch TEXT A
ASSEMBLE #ucfbtch
```

```
FILEDEF SYSLST PRINTER
FILEDEF SYSLMOD DISK idmslib LOADLIB a2 (RECFM V LRECL 1024 BLKSIZE 1024
LKED linkctllinkage editor control statements (linkctl):INCLUDE ucfbtch
INCLUDE IDMSOPTI
INCLUDE IDMS
INCLUDE RHDCUCFC
INCLUDE IDMSUITIO
ENTRY IDMSENTR
NAME ucfbtch(R)
```

<table>
<thead>
<tr>
<th>idmslib</th>
<th>filename of the CA IDMS MACLIB library</th>
</tr>
</thead>
<tbody>
<tr>
<td>idmslib LOADLIB a2</td>
<td>file ID of the CA IDMS LOADLIB library</td>
</tr>
<tr>
<td>linkctl</td>
<td>filename of the file containing the linkage editor control statements</td>
</tr>
</tbody>
</table>
Batch Front-end Execution

When executed, the batch front-end reads input from SYSIPT. Each statement is interpreted as terminal input and is passed to the DC/UCF system. The physical terminal name is used as the terminal identifier. Output from the DC/UCF task is returned to the front-end, which writes the output to SYSLST. The terminal disconnect occurs when the BYE task is invoked.

The batch front-end operates in dedicated mode only. You must provide a job card specifying a user ID and password to execute tasks with a security class other than zero. Alternatively, you can specify the SIGNON task code in the input:

```
SIGNON ABC ABCPASS
DCMT DISPLAY ACTIVE TASKS
DCMT DISPLAY ACTIVE PROGRAMS
DCUF SHOW USERS ALL
BYE
```

Batch Front-end Execution JCL

This section contains the JCL to execute the UCF batch front-end for z/OS, z/VSE, and z/VM operating systems.

**z/OS batch front-end execution**

```
// EXEC PGM=ucfbtch
//STEPLIB DD DSN=idms.custom.loadlib,DISP=SHR
// DD DSN=idms.cagjload,DISP=SHR
//SYSCTL DD DSN=idms.sysctl,DISP=SHR
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
//SYSIDMS parameters, as required
//SYSIPT DD *
// input statements
/*
```

**Note:** See Library and dataset References in z/OS JCL for variable information used in the JCL above.

**z/VSE batch front-end execution**

```
/* UPSI b
//idms.library',2099/365,DA
// EXTENT ,nnnnn,,ssss,1500
// LIBDEF *,SEARCH=(idmslib.sublib)
// EXTENT SYSnnn,nnnnn,,ssss,54
// ASSIGN SYSnnn,DISK, VOL=nnnnn,SHR
// DLBL SYSIDMS, '#SYSIPT', 0,SD
// EXEC ucfbtchinput statements/*
```

**Note:** You can define a SYSCTL file in the JCL to override IDMSOPTI specifications for the back-end system as follows:
Setting Options for Batch UCF Program Execution

The SET OPTIONS input statement allows users to establish input options during batch UCF program execution. The words SET OPTIONS must start in column 1 and must be followed by a blank. Each of the following options is specified in a separate SET OPTIONS statement:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>appropriate one- through eight-character UPSI bit switch, as specified in the IDMSOPTI module</td>
</tr>
<tr>
<td>idmslib</td>
<td>filename of the file containing CA IDMS modules</td>
</tr>
<tr>
<td>idmslib.</td>
<td>name of the sublibrary within the library containing CA IDMS modules</td>
</tr>
<tr>
<td>sublib</td>
<td></td>
</tr>
<tr>
<td>idms.library</td>
<td>file-id associated with the file containing CA IDMS modules</td>
</tr>
<tr>
<td>idms.sysctl</td>
<td>file-id of the SYSCTL file</td>
</tr>
<tr>
<td>nnnnn</td>
<td>volume serial number</td>
</tr>
<tr>
<td>ssss</td>
<td>starting track (CKD) or block (FBA) of disk extent</td>
</tr>
<tr>
<td>sysctl</td>
<td>filename of the SYSCTL file</td>
</tr>
<tr>
<td>sys008</td>
<td>logical unit assignment of the SYSCTL file</td>
</tr>
<tr>
<td>ucfbtch</td>
<td>phase name of the batch front-end module</td>
</tr>
<tr>
<td>z/VM batch front-end execution</td>
<td></td>
</tr>
</tbody>
</table>

Note: You can define a SYSCTL file in the JCL to override IDMSOPTI specifications for the back-end system as follows:

```plaintext
FILEDEF sysctl DISK sysctl idms a
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbb</td>
<td>blocksize</td>
</tr>
<tr>
<td>idmslib</td>
<td>name of the CA IDMS LOADLIB library</td>
</tr>
<tr>
<td>lll</td>
<td>logical record length</td>
</tr>
<tr>
<td>sysctl</td>
<td>ddname of the SYSCTL file</td>
</tr>
<tr>
<td>sysctl idms a</td>
<td>file ID of the SYSCTL file</td>
</tr>
<tr>
<td>sysidms parms a</td>
<td>file ID of the SYSIDMS parameter file</td>
</tr>
<tr>
<td>ucfbtch</td>
<td>name of the batch front-end load module</td>
</tr>
<tr>
<td>ucfbtch input a</td>
<td>file ID of the file containing the UCF batch input statements</td>
</tr>
</tbody>
</table>

Note: You can define a SYSCTL file in the JCL to override IDMSOPTI specifications for the back-end system as follows:

```plaintext
FILEDEF sysctl DISK sysctl idms a
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbb</td>
<td>blocksize</td>
</tr>
<tr>
<td>idmslib</td>
<td>name of the CA IDMS LOADLIB library</td>
</tr>
<tr>
<td>lll</td>
<td>logical record length</td>
</tr>
<tr>
<td>sysctl</td>
<td>ddname of the SYSCTL file</td>
</tr>
<tr>
<td>sysctl idms a</td>
<td>file ID of the SYSCTL file</td>
</tr>
<tr>
<td>sysidms parms a</td>
<td>file ID of the SYSIDMS parameter file</td>
</tr>
<tr>
<td>ucfbtch</td>
<td>name of the batch front-end load module</td>
</tr>
<tr>
<td>ucfbtch input a</td>
<td>file ID of the file containing the UCF batch input statements</td>
</tr>
</tbody>
</table>
INPUT start-column THRU end-column specifies the start and end columns of the input column range. The batch front-end program treats data within the specified columns as input and ignores any other data on the line. The default range is 1 through 80. Typically, this parameter is used to limit the input column range to columns 1 through 72.

ECHO/NOECHO specifies whether subsequent input statements are written to SYSLST:

- ECHO (default) specifies to write input statements to SYSLST.
- NOECHO specifies not to write input statements to SYSLST.

⚠️ **Note**: SET OPTIONS statements are always echoed, regardless of the input options that have been established.

**Example SET OPTIONS statements**

In the following example, SET OPTIONS statements limit input columns to 1 through 72 and request the program not to echo the user's password:

```
//UCFBTCH EXEC PGM=UCFBTCH
//STEPLIB DD DSN=idms.custom.loadlib,DISP=SHR
//SYSCTL DD DSN=idms.cagjload,DISP=SHR
//SYSLST DD DSN=idms.sysctl,DISP=SHR
//SYSUDUMP DD SYSOUT=A
//SYSIPT DD *
SET OPTIONS INPUT 1 THRU 72
SIGNON HARPO
SET OPTIONS NOECHO
SWORDFISH
SET OPTIONS ECHO
DCMT DISPLAY ACTIVE STORAGE 5
DCMT DISPLAY LINES
BYE
```

**CICS Front-end**

The UCF CICS front-end enables DC/UCF tasks to be executed from a terminal connected to CICS.

A UCF CICS front-end program contains an options table and a small stub module that calls the main processing routines.

To create the CICS front-end module, assemble a #UCFCICS macro and link edit the resulting object module as shown in the following section.

⚠️ **Note**: UCF requires the presence of the IDMSINTC interface module in the CICS region /partition. For instructions on assembling and link editing IDMSINTC, see CICS Considerations.
The following is the syntax for the #UCFCICS macro. The macro must be labeled; the label provides the module's entry point name.

Syntax

```
label #UCFCICS

COLOR = YES
    NO
    ONLY

DBCS = YES
    NO

.DEBUG = YES
    NO

.DISC = YES
    NO

.ERRODC = destination-name
    CSMT

.LASTOUT = TASKEND
    RESETKB

.OS = OS
    DOS

.PRINT = print-option
    OFF

.RDW = YES
    NO

.READBUF = YES
    NO

.RECVRTS = YES
    NO

.RESETKB = TASKEND
    ASIS

.SUFFIX = suffix

.UCTRAN = TCT
    NO

.USERCHK = YES
    NO
```
Parameters

- **COLOR=**YES|NO|ONLY
  Specifies whether the front-end is to check to see if the terminal is a 3279-type terminal with extended color support. If YES is specified, DC/UCF will treat the terminal as a 3279 terminal if CICS reports positive settings for either the COLOR or EXTDS attributes. If ONLY is specified, DC/UCF will treat the terminal as a 3279 terminal only if CICS reports a positive setting for the COLOR attribute. The default is NO.

- **DBCS=**YES|NO
  Specifies whether the terminal supports double-byte characters. If YES is specified, terminals that support COLOR or extended data streams are assumed to also have Double Byte Character Support. COLOR=YES must be specified for DBCS=YES to take effect.

- **DEBUG=**YES|NO
  Specifies whether EXEC CICS ENTER TRACEID commands will be issued at various processing points during the execution of each UCF transaction. DEBUG=YES is only used for diagnostic purposes. The default is NO.

- **DISC=**YES|NO
  Specifies what action the DC/UCF system is to take at the end of an intermittent mode pseudo-converse. DISC=YES specifies that all DC/UCF resources will be freed. DISC=NO specifies that the DC/UCF terminal should be put in a SUSPEND state. The default is NO. Note: If DISC=YES is specified, RHDCOPTF bit 20 must be set in the corresponding back-end DC system.

- **ERRDCT=**
  Identifies the CICS transient data destination to be used as the target for error messages produced by IDMSINTC and IDMSTRUE. The default destination name is CSMT. Use another destination if you want to route CA IDMS error messages to another CICS destination. The DCT entry should be defined with a logical record length of at least 130 characters.

- **LASTOUT=**TASKEND|RESETKB
  Specifies the action the UCFCICS front-end module takes when issuing WRITE commands. If LASTOUT=RESETKB, UCFCICS includes the LAST option on the EXEC CICS SEND command whenever it writes a datastream that includes the RESET keyboard indicator. If LASTOUT=TASKEND, UCFCICS does not specify the LAST option when writing datastreams. CICS will automatically transmit an "end-of-bracket" indicator at task end. RESETKB is the default.

- **OS=**OS|DOS
  Specifies the operating system under which the CICS front-end system will execute. OS applies to all supported z/OS versions of CICS. DOS applies to all supported z/VSE versions of CICS. The default is OS.

- **PRINT=**print-option
  Specifies what print option the Assembler is to use while processing the COPY commands which copy in the DSECTs used in the UCF front-end module. Valid options are ON|GEN|NOGEN. See the documentation for your Assembler product for details. The default is OFF.

- **RDW=**YES|NO
  Specifies whether UCFCICS should pass the RDW as part of the PDAT data when starting a front-end task in a distributed application as described in section 4.1. The default is NO.
- **READBUF=YES|NO**  
  Specifies whether the physical terminal can execute a READ BUFFER command; the default is YES. At bind time, the READBUF specification in the #UCFCICS macro overrides the READBUFFER /NOREADBUFFER specification established by the system generation PTERM statement for the physical terminal.

- **RECVRTS=YES|NO**  
  YES specifies that the CICS system is generated with RECOVERABLE AUX TEMP storage. This parameter may be needed for UCFPRINT to work properly on such systems. The default is NO.

- **RESETKB=TASKEND|ASIS**  
  Specifies whether the UCFCICS front-end module should suppress the RESET keyboard indicator when writing datastreams. If YES is specified, the RESET indicator will be removed from any datastream written to the terminal. This will also prevent the LAST option from being specified on all EXEC CICS SEND commands. At task termination, an extra EXEC CICS SEND command will be transmitted specifying zero length, RESET keyboard, and LAST. IF ASIS is specified, the RESET keyboard indicator will be left as passed by the application code running in the DC/UCF back-end system. The LAST option will be controlled by the LASTOUT parameter described above. The default is ASIS.

- **SUFFIX=suffix**  
  This is an obsolete parameter, but it is allowed for compatibility.

- **UCTRAN=TCT|NO**  
  Specifies whether to translate input data to uppercase before it is transferred to the DC/UCF system as follows:

  - **TCT** -- Translation of terminal input is determined by the presence or absence of the FEATURE=UCTRAN parameter in the TCT definition for the terminal.
  - **NO (default)** -- Terminal input is not translated to uppercase.

- **USERCHK=YES|NO**  
  Specifies whether UCFCICS is to check the CICS userid at the beginning of each task in a pseudo-converse. YES will prevent an operator who has newly signed on to a terminal from being connected to another user’s back-end UCF terminal session. This can occur if the first user’s session was aborted (for example, by “killing” the session from a VTAM session manager) and UCFCICZ is not implemented to clean up the associated UCFCICS storage in the CICS region where the UCFCICS transaction runs. YES causes UCFCICS to check the CICS USERID at the beginning of each task in a pseudo-converse. If the userid does not match the userid saved from the previous task in that pseudo-converse, a new session is started instead of resuming the old one. The old session will automatically be aborted. The new session will continue normally. USERCHK=YES is not enforced if no CICS signon is done. For example, consider the following scenario:

  - User A is on a CICS terminal, but is not signed on to CICS.
  - User A starts a UCF session and his CICS session is aborted while in a pseudo-converse.
  - User B comes into CICS and gets the same terminal. User B also is not signed on to CICS.
  - User B invokes UCF and picks up user A’s session.
Note that in the above scenario, the old session is aborted if either user A or user B (or both) is signed on to CICS.
The comparison of the new and saved userid is done after the call to the USRIDXIT, if that exit exists. The value set by the USRIDXIT is the one that is saved across a pseudo-converse, so the use of the exit in conjunction with USERCKHK=YES should not cause a problem even if the exit modifies the user id used to connect to the back-end DC/UCF system. Also note that the security function of the USERCHK parameter will not work if the USRIDXIT changes different CICS user-ids to a common back-end id.
The default is NO.

CICS Front-end JCL

To create a z/OS UCF CICS front-end program

1. Create a UCFCICS source module as follows:
   ```
   ucfcicse   #UCFCICS  ucfcics-parameters
   END
   ```
2. Save the UCFCICS source module in your custom source library.
3. Assemble and link it into your custom load library by executing the z/OS Assemble and Link-Edit JCL (see page 465).
   Substitute the name of your UCFCICS source member and insert the following binder statements:
   ```
   INCLUDE  CAGJLOAD(UCFCICX0)
   INCLUDE  CUSTLIB(ucffet)
   INCLUDE  CUSTLIB(idmscint)  (optional)
   INCLUDE  CUSTLIB(ucfprint)  (optional)
   INCLUDE  CUSTLIB(usridxit)  (optional)
   SETOPT  PARM(AMODE=31,RMODE=24)
   ENTRY  ucfcicse
   NAME  ucfcicse(R)
   ```
   - **ucffet**
     Specifies the name of your UCF front-end table.
   - **idmscint**
     Specifies the name of your IDMSCINT interface module.
   - **ucfprint**
     Specifies the name of your UCFPRINT exit routine. This is optional.
   - **usridxit**
     Specifies the name of your USRIDXIT exit routine. This is optional.
   - **ucfcicse**
     Specifies the entry point name of your UCFCICS options module:
     **Note:** Use UCFCXEP1 or the name specified as the label on the assembler instruction that invokes the #UCFCICS macro. The assembler label is optional. You cannot use UCFCXEP1 as the assembler label.
   - **ucfcics**
     Specifies the name of your UCF CICS front-end load module.
4. Define program \textit{ucfcics} in your CICS CSD.
For a library template, see the definition of PROGRAM(UCFICS) in member CICSCSD of the CAGJSRC library. The program must be defined as RESIDENT.

\textit{z/VSE CICS front-end load module assembly}

\textbf{To create a z/VSE UCF CICS front-end program:}

1. Assemble and catalog the module using the sample JCL in \textit{z/VSE Assemble JCL} (see page 476).
   Modify the JCL by substituting the following in place of the \textit{Assembler input statements}:
   \begin{verbatim}
PUNCH 'CATALOG ucfcics.OBJ REPLACE=YES'
   #UCFCICS #ucfcics-parameters
   END
   \end{verbatim}

2. Link the \textit{ucfcics} program using the sample JCL in \textit{z/VSE Link JCL} (see page 476).
   Modify the JCL by substituting the following statements in place of the \textit{Linkage editor control statements}:
   \begin{verbatim}
   PHASE ucfcics, *
   INCLUDE DFHEAI
   INCLUDE ucfcics
   INCLUDE ucfet
   INCLUDE idmscint
   INCLUDE ucfprint \hspace{1em} (optional)
   INCLUDE usridxit \hspace{1em} (optional)
   INCLUDE DFHEAI0
   MODE AMODE(31),RMODE(24)
   ENTRY ucfcicse
   \end{verbatim}

3. Define program \textit{ucfcics} in your CICS CSD.
For a library template, see the definition of PROGRAM(UCFICS) in member CICSCSD of the CAGJSRC library. Define the program as RESIDENT.

- \textit{ucfet}
  Specifies the name of your UCF front-end table.

- \textit{idmscint}
  Specifies the name of your IDMSCINT interface module.

- \textit{ucfprint}
  (Optional) Specifies the name of your UCFPRINT exit routine.

- \textit{usridxit}
  (Optional) Specifies the name of your USRIDXIT exit routine.

- \textit{ucfcicse}
  Specifies the entry point name of your UCFCICS options module.

\textbf{Note}: Use UCFCXEP1 or the name specified as the label on the assembler instruction that invokes the \#UCFCICS macro. The assembler label is optional. You cannot use UCFCXEP1 as the assembler label.

- \textit{ucfcics}
  Specifies the name of your UCF CICS front-end module.
CICS Front-end Execution

To execute the CICS front-end, you must add entries to the CICS CSD. See the definitions for PROGRAM UCFCICS and TRANSACTION(DBDC) in the source library created during CA IDMS installation.

Dedicated mode considerations

For the dedicated task, task-code must match the task code specified in the NTID parameter of the #UCFUFT macro used to create the CICS front-end table. The name ucfcics is the name of the CICS front-end module; the same module name should be used for all task codes, whether dedicated or intermittent.

Intermittent mode considerations

Each intermittent DC/UCF task defined in the CICS system must also be defined in the DC/UCF system with a TASK statement; the task code specified in the DC/UCF Sysgen must match the transaction name specified in the CICS CSD.

Entering the task code

Once defined to CICS and to the DC/UCF system, the CICS front end can be executed by entering the dedicated task code or an intermittent task code.

Sample UCF definitions for CICS

Sample UCF system definition: The following examples show statements that define UCF with printer support on a CICS system. The CICS system has 12 terminals and 3 printers. The sample system uses sample programs UCFPRINT and UCFDESPL.

For more information on these programs see Special CICS Considerations (https://docops.ca.com/display/IDMS19/Printer+Support#PrinterSupport-SpecialCICSConsiderations).

The sample system is defined as follows:

- The UCF front-end table that defines devices to UCF.
- System generation LINE and PTERM statements that define devices to DC/UCF.
- Related CICS definitions.
- Related CICS definitions.

Note: For print devices, the same print-device names must be specified in the #UCFUTD macros that define the print devices (PRTA, PRTB, and PRTC in this example), in the PTERM statements that define the printers to DC/UCF, and in CICS printer definitions.

Sample UCF front-end table definition:
CICSFET  #UCFUFT SYSTEM=CICS,  X
     NTID=DBDC,  *DEDICATED TASK ID  X
     PTID=UCFP  *PRINT TASK ID

* DEFINE INTERACTIVE TERMINALS TO UCF
  #UCFUTD TERM=T001,TYPE=T3277,MODEL=2,PRTCLS=10
  #UCFUTD TERM=T002,TYPE=T3277,MODEL=2,PRTCLS=10
  #UCFUTD TERM=T003,TYPE=T3277,MODEL=2,PRTCLS=20
  #UCFUTD TERM=T004,TYPE=T3277,MODEL=2,PRTCLS=30
  #UCFUTD TERM=T005,TYPE=T3277,MODEL=2,PRTCLS=20
  #UCFUTD TERM=T006,TYPE=T3277,MODEL=2,PRTCLS=10
  #UCFUTD TERM=T007,TYPE=T3277,MODEL=2,PRTCLS=30
  #UCFUTD TERM=T008,TYPE=T3277,MODEL=2,PRTCLS=30
  #UCFUTD TERM=T009,TYPE=T3277,MODEL=2,PRTCLS=30
  #UCFUTD TERM=T010,TYPE=T3277,MODEL=2,PRTCLS=10
  #UCFUTD TERM=T011,TYPE=T3277,MODEL=2,PRTCLS=20
  #UCFUTD TERM=T012,TYPE=T3277,MODEL=2,PRTCLS=20

* DEFINE PRINTER TERMINALS TO UCF
  #UCFUTD TERM=PRTA,TYPE=3280,FORMFD=NO
  #UCFUTD TERM=PRTB,TYPE=3280,FORMFD=YES
  #UCFUTD TERM=PRTC,TYPE=3280,FORMFD=YES

Sample DC/UCF system generation input

LINE UCF ENABLED TYPE IS UCFLINE MODULE IS RHDCFSTB.

PTERM UCF01 ENABLED TYPE IS UCFTERM IN LINE UCF.
  LTERM UCF01.
PTERM UCF02 ENABLED TYPE IS UCFTERM IN LINE UCF.
  LTERM UCF02.
PTERM UCF03 ENABLED TYPE IS UCFTERM IN LINE UCF.
  LTERM UCF03.
PTERM UCF04 ENABLED TYPE IS UCFTERM IN LINE UCF.
  LTERM UCF04.
PTERM UCF05 ENABLED TYPE IS UCFTERM IN LINE UCF.
  LTERM UCF05.
PTERM UCF06 ENABLED TYPE IS UCFTERM IN LINE UCF.
  LTERM UCF06.
PTERM UCF07 ENABLED TYPE IS UCFTERM IN LINE UCF.
  LTERM UCF07.
PTERM UCF08 ENABLED TYPE IS UCFTERM IN LINE UCF.
  LTERM UCF08.
PTERM UCF09 ENABLED TYPE IS UCFTERM IN LINE UCF.
  LTERM UCF09.
PTERM UCF10 ENABLED TYPE IS UCFTERM IN LINE UCF.
  LTERM UCF10.
PTERM UCF11 ENABLED TYPE IS UCFTERM IN LINE UCF.
  LTERM UCF11.
PTERM UCF12 ENABLED TYPE IS UCFTERM IN LINE UCF.
  LTERM UCF12.

* UCF PRINTERS

PTERM UCFPRTA ENABLED TYPE IS UCFTERM IN LINE UCF NAME=PRTA
  LTERM UCFPRTA PRINTER CLASS=(10,11,12,13,14).

PTERM UCFPRTB ENABLED TYPE IS UCFTERM IN LINE UCF NAME=PRTB
  LTERM UCFPRTB PRINTER CLASS=(20,21,22,23,24).

PTERM UCFPRTC ENABLED TYPE IS UCFTERM IN LINE UCF NAME=PRTC
  LTERM UCFPRTC PRINTER CLASS=(30,31,32,33,34).

Sample CICS generation input

DEFINE   PROGRAM(UCFCICS) LANGUAGE ASSEMBLER CEDF(NO)
           EXECKEY(CICS) RESIDENT
DEFINE   PROGRAM(UCFDESPL) LANGUAGE ASSEMBLER
DEFINE TRANSACTION(DBDC) PROGRAM(UCFCICS)
DEFINE TRANSACTION(UCFD) PROGRAM(UCFDESPL)
DEFINE TDQUEUE(PRTA) TYPE(INTRA) 
   ATIFACILITY(TERMINAL) FACILITYID(PRTA) 
   TRANSID(UCFD) TRIGGERLEVEL(1)
DEFINE TDQUEUE(PRTB) TYPE(INTRA) 
   ATIFACILITY(TERMINAL) FACILITYID(PRTB) 
   TRANSID(UCFD) TRIGGERLEVEL(1)
DEFINE TDQUEUE(PRTC) TYPE(INTRA) 
   ATIFACILITY(TERMINAL) FACILITYID(PRTC) 
   TRANSID(UCFD) TRIGGERLEVEL(1)

⚠️ **Note:** Transactions that invoke the UCFCICS program will determine screen size characteristics by issuing an EXEC CICS ASSIGN command. You might want to associate these transactions with a PROFILE that specifies SCRNSIZE(ALTERNATE) to allow IDMS mapping applications to recognize the true device characteristics of the CICS terminals.

⚠️ For more information on SCRNSIZE see the relevant IBM documentation.

### CICS Abort Session Program

The #UCFCICZ macro can be assembled to create an abort program to request UCF to abort the session for any terminal that disconnects or goes out of service. You can call the abort program by any combination of the following methods:

- The CICS terminal error program DFHTEP
- The node error program DFHZNEP
- The bridge facility global exit XFAINTU

#### #UCFCICZ Macro

By using #UCFCICZ, you can assure the timely release of back-end resources when a front-end abort occurs. You can also prevent the following scenario from occurring:

A user signs onto CICS through a bridge facility or onto a VTAM terminal through a multisession manager. During a terminal-read request from UCF, the user loses the connection or terminates the CICS session from the multisession manager. A second user simultaneously connects and is assigned to the same terminal identifier. The second user invokes the UCF front-end program and is placed in the middle of the session started by the first user.
The situation described previously can also be avoided if either user signs onto CICS and the USERCHK=YES parameter is specified on the #UCFCICS macro. For more information on the USERCHK parameter see the section CICS Front-end (see page 92).

Note: A UCF CICS abort session program contains an options table and a small stub module that calls the main processing routines. Create the program by invoking the #UCFCICZ macro and linking it as shown below.

Syntax

```plaintext
#UCFCICZ
BRIDGE= YES NO

ERRDCT= CSMT destination-name

INTCID= ( intc-start-task )

MACLVL= YES NO

NTID= dedicated-task-code

OPSYS= MV5 VSE

PASSVAL= TERMD TCTADDR

SQL= YES NO
```

Parameters

- **BRIDGE**
  Specifies whether the module generated by this #UCFCICZ macro will be called from a program invoked by the bridge facility exit point XFAINTU. If NO is specified, the UCF abort session program assumes that the aborted session is associated with a permanent terminal. If UCFCICS had modified the UCTRANST value associated with the terminal, the UCF abort session program attempts to restore the original UCTRANST value. Therefore, this parameter has no effect if the associated UCFCICS macro specifies UCTRAN=TCT. The default is NO.

- **ERRDCT**
  Identifies the CICS transient data destination to be used as the target for error messages produced by IDMSINTC and IDMSTRUE. The default destination-name is CSMT. Use another destination if you want to route CA IDMS error messages to another CICS destination. The DCT entry should be defined with a logical record length of at least 130 characters.

- **INTCID**
  Specifies the CICS transaction identifiers used to start each of the IDMSINTC interfaces. Use sublist notation when using more than one interface. When a lost terminal event occurs, each of these interfaces is notified so that it can sign the user off the respective IDMS system.
  If multiple #UCFCICZ macros are assembled with different NTID parameters, you need to specify each IDMSINTC transaction identifier only once.
Note: Do not specify an INTCID transaction identifier for an IDMSINTC transaction which runs in a different region from the Terminal Owning Region (TOR).

- **MACLVL**
  - This parameter is obsolete. Macro level programs are no longer supported in CICS.

- **NTID**
  - Specifies the task code of the #UCFUFT macro used to create the CICS front-end table. If more than one UCFCICS front-end table exists (to communicate with more than one CA IDMS DC/UCF back-end), you must assemble and link edit separate CICZ modules. Each assembly should specify one NTID. Each CICZ module should be linked with a unique name.

- **PASSVAL=TCTADDR/TERMD**
  - Specifies the format and value of the COMMAREA parameter that is passed to the UCF abort session program. PASSVAL=TCTADDR indicates that the COMMAREA contains a fullword address pointing to the Terminal Control table. PASSVAL=TERMD indicates that the COMMAREA contains the 4-byte identifier of the terminal or bridge facility associated with the aborted session. The default is TERMD.

- **SQL**
  - Specifies whether SQL suspended sessions will be cleared by the abort session program. The default is NO.

**How to Use the UCF CICS Abort Session Program**

One or two UCF CICS abort sessions are needed for each UCFCICS program created with a #UCFUFT macro that specifies the corresponding NTID. One program is needed for persistent terminals. A separate one may be needed for sessions associated with a bridge facility. A single program can be used if both the following conditions are true:

- All callers pass the same format COMMAREA to the UCF CICS abort session program as defined by the PASSVAL parameter. PASSVAL=TERMD is recommended.

  Note: The recommended value is PASSVAL=TERMD, which is the default in Release 17.0 and later.

- The associated #UCFCICS macro specifies UCTRAN=TCT.

For each UCF CICS abort session program you create, perform the following steps:

1. Assemble the #UCFCICZ macro with the appropriate parameters and link the resulting program with a unique name.

2. Add an entry to the CICS CSD for each session abort program as follows:

   ```
   DEFINE PROGRAM(ucfcicz) GROUP(groupnam) LANGUAGE(ASSEMBLER) CEDF(NO) EXECKEY(CICS) RESIDENT
   ```

3. Modify DFHTEP, DFHZNEP, and XFAINTU to call the appropriate versions of the program.
Assemble and link edit #UCFCICZ

To assemble and link #UCFCICZ under z/OS:

1. Create a UCFCICZ source module as follows:

   #UCFCICZ ucfcicz-parameters
   END

2. Save the UCFCICZ source module in your custom source library.

3. Assemble and link it into your custom load library by executing the z/OS Assemble and Link-edit JCL (see page 465). Substitute the name of your UCFCICZ source member and insert the following binder statements:

   ORDER DFHEAI
   INCLUDE CICLOAD(DFHEAI)
   INCLUDE CAGJLOAD(UCFCIZX0)
   INCLUDE CUSTLIB(idmscint)
   INCLUDE CICLOAD(DFHEAI0)
   ENTRY CICZEP
   SETOPT PARM(AMODE=31,REUS(NONE),RMODE=24)
   NAME ucfcicz(R)

   - **idmscint**
     Specifies the name of your IDMSCINT interface module.

   - **ucfcicz**
     Specifies the name of your UCF CICS abort session load module.

To assemble and link #UCFCICZ under z/VSE:

1. Assemble and catalog the module using the sample JCL in z/VSE Assemble JCL (see page 476). Modify the JCL by substituting the following in place of the Assembler input statements:

   PUNCH 'CATALOG ucfcicz.OBJ REPLACE=YES'
   #UCFCICZ #ucfcicz-parameters
   END

2. Link the UCFCICZ program using the sample JCL in z/VSE Link JCL. Modify the JCL by substituting the following statements in place of the Linkage editor control statements:

   PHASE ucfcicz,"
   INCLUDE DFHEAI
   INCLUDE ucfcicz
   INCLUDE ucfcizx6
   INCLUDE idmscint
   INCLUDE DFHEAI0
   MODE AMODE(31),RMODE(24)
   ENTRY CICZEP

   - **idmscint**
     Specifies the name of your IDMSCINT interface module.

   - **ucfcicz**
     Specifies the name of your UCF CICS abort session module.

Modify DFHTEP/DFHZNEP/XFAINTU to link to UCFCICZ
Modify the error programs or bridge facility tidy up program or both to link to the appropriate UCF session abort programs.

Note: For more information on DFHTEP, DFHZNEP, and XFAINTU, refer to the CICS system documentation.

The following examples illustrate one approach to the modification of error and tidy up programs.

For DFHTEP and DFHZNEP, insert the instructions immediately before the DFHTEP/DFHZNEP exit. The logic states that if the error action codes indicate that the application task (if any) is to abend, a link is made to two UCF CICS session abort programs.

**DFHTEP instructions when PASSVAL=TERMID**

The following statements add instructions to DFHTEP when the #UCFCICZ macro specifies PASSVAL=TERMID:

```
TM TCTLEECB+1,X'04' ABEND TASK?
BZ N0CICZ NO
LA 10,TCTLEPTE POINTER TO TCTTE
L 10,0(,10) TCTTETI
EXEC CICS LINK PROGRAM('UCFCICZ1')
    COMMAREA( 0(10) )
    LENGTH( 4 ).
EXEC CICS LINK PROGRAM('UCFCICZ2')
    COMMAREA( 0(10) )
    LENGTH( 4 ).
N0CICZ DS 0H
```

**DFHTEP instructions when PASSVAL=TCTADDR**

The following statements add instructions to DFHTEP when the #UCFCICZ macro specifies PASSVAL=TCTADDR. This method is provided for compatibility with earlier versions. PASSVAL=TERMID is recommended.

```
TM TCTLEECB+1,X'04' ABEND TASK?
BZ N0CICZ NO
LA 10,TCTLEPTE POINTER TO TCTTE
EXEC CICS LINK PROGRAM('UCFCICZ1')
    COMMAREA( 0(10) )
    LENGTH( 4 ).
EXEC CICS LINK PROGRAM('UCFCICZ2')
    COMMAREA( 0(10) )
    LENGTH( 4 ).
N0CICZ DS 0H
```

**DFHZNEP instructions when PASSVAL=TERMID**

The following statements add instructions to DFHZNEP when the #UCFCICZ macro specifies PASSVAL=TERMID:

```
TM TWAROPT2,TWA0AT ABEND TASK?
BZ N0CICZ NO
L 7,TWATCTA
EXEC CICS LINK PROGRAM('UCFCICZ1')
    COMMAREA(0(7))
    LENGTH( 4 ).
```
EXEC CICS LINK PROGRAM('UCFCICZ1')
    COMMAREA(TWATCTA)
    LENGTH(4).
NOCIZC DS 0H

DFHZNEP instructions when PASSVAL=TCTADDR

The following statements add instructions to DFHZNEP when the #UCFCICZ macro specifies
PASSVAL=TCTADDR. This method is provided for compatibility with earlier versions.
PASSVAL=TERMID is recommended.

EXEC CICS LINK PROGRAM('UCFCICZ1')
    COMMAREA(TWATCTA)
    LENGTH(4).
EXEC CICS LINK PROGRAM('UCFCICZ2')
    COMMAREA(TWATCTA)
    LENGTH(4).

XFAINTU instructions to invoke the session abort program

The following statements illustrate how to modify XFAINTU. The code should be executed only if
UEPFAREQ contains the value UEPFATU on entry to XFAINTU.

EXEC CICS LINK PROGRAM('UCFCICZ1')
    COMMAREA(UEPFANAM)
    LENGTH(4).
EXEC CICS LINK PROGRAM('UCFCICZ2')
    COMMAREA(UEPFANAM)
    LENGTH(4).

DC Front-end

The UCF DC front-end enables a terminal on one DC system to execute tasks on a second DC system.

UCF DC front-end programs contain an options table and a small set of executable code that calls the
main processing routines that reside in separate modules.

#UCFOPTS Macro

To create the DC front-end module, assemble a #UCFOPTS macro and link edit the resulting object
module as shown in the following section.

Syntax

label #UCFOPTS

CVNUM = cv-number, SVCNUM = svc-number
NODE = nodename, ACCTYPE= CCI, VTAM, TCP/IP, TCPIP
DISC= YES, NO
FEONLY= YES, NO
Parameter

Assembly JCL

This section contains the JCL to create the DC front-end load module for z/OS, z/VSE, and z/VM operating systems.

To create a z/OS UCF DC front-end program:

1. Create a UCFOPTS source module as follows:
   
   ```
   #UCFOPTS ucfopts-parameters
   END
   ```

2. Save the UCFOPTS source module in your custom source library.

3. Assemble and link it into your custom load library by executing the z/OS Assemble and Link-edit JCL (see page 465).
   Substitute the name of your UCFOPTS source member and insert the following binder statements:
   
   ```
   INCLUDE CAGJLOAD(RHDCDBDC)
   INCLUDE CUSTLIB(ucffet)
   SETOPT PARM(AMODE=31)
   ENTRY DBDCEP1
   NAME ucfdbc(R)
   ```

   - **ucffet**
     Specifies the name of your UCF front-end table.

   - **ucfdbc**
     Specifies the name of your UCF DC front-end load module.

To create a z/VSE UCF DC front-end program:

1. Assemble and catalog the module using the sample JCL in z/VSE Assemble JCL. (see page 476)
   Modify the JCL by substituting the following in place of the Assembler input statements:
   
   ```
   PUNCH 'CATALOG ucfoptso.OBJ REPLACE=YES'
   #UCFOPTS ucfopts-parameters
   END
   ```

2. Link the IDMSINTL interface program using the sample JCL in z/VSE Link JCL (see page 476).
   Modify the JCL by substituting the following statements in place of the Linkage editor control statements:
   
   ```
   PHASE ucfdbc,*
   INCLUDE ucfoptso INCLUDE CUSTLIB(ucffet)
   SETOPT PARM(AMODE=31,RMODE=ANY)
   ENTRY DBDCEP1
   ```

   - **ucfoptso**
     Specifies the name of your UCF DC options object module

z/VM DC front-end load module assembly

```
GLOBAL MACLIB idmslib
FILEDEF TEXT DISK ucfdbc TEXT A
```
ASSEMBLE #ucfopts
FILEDEF SYSLST PRINTER
FILEDEF SYSLMOD DISK idmslib LOADLIB a2 (RECFM V LRECL 1024 BLKSIZE 1024
LKED linkctl Linkage editor control statements (linkctl): INCLUDE ucfopts
INCLUDE RHDCUTCF
INCLUDE RHDDBDBD
INCLUDE ucfet
ENTRY DBDCEP1
NAME ucfdbdc(R)

<table>
<thead>
<tr>
<th>idmslib</th>
<th>filename of the CA IDMS MACLIB library</th>
</tr>
</thead>
<tbody>
<tr>
<td>idmslib LOADLIB a2</td>
<td>file ID of the CA IDMS LOADLIB library</td>
</tr>
<tr>
<td>linkctl</td>
<td>filename of the file containing the linkage editor control statements</td>
</tr>
<tr>
<td>ucfdbdc</td>
<td>name of the DC front-end module</td>
</tr>
<tr>
<td>ucfet</td>
<td>name of the DC front-end table</td>
</tr>
<tr>
<td>#ucfopts</td>
<td>filename of the file containing the #UCFOPTS macro statement</td>
</tr>
<tr>
<td>ucfopts</td>
<td>name of the UCF options macro</td>
</tr>
</tbody>
</table>

**DC Front-end Execution**

To execute the DC front-end module, include the following system generation statements in the definition of the DC front-end:

- A PROGRAM statement to define the DC front-end:
  
  ```
  ADD PROGRAM ucfdbdc LANGUAGE IS ASSEMBLER NOPROTECT NONREENTRANT.
  ```
  
  *ucfdbdc* is the name of the DC front-end module.

- A TASK statement for the dedicated task code and one for each intermittent task code:
  
  ```
  ADD TASK task-code INVOKES PROGRAM ucfdbdc INPUT.
  ```

**Dedicated mode considerations**

For the dedicated task, *task-code* must match the task code specified in the NTID parameter of the #UCFUFT macro used to create the DC front-end table. *ucfdbdc* is the name of the DC front-end module; use the same module name for all task codes, whether dedicated or intermittent.

**Intermittent mode considerations**

Each intermittent task defined in the DC front-end system must also be defined in the DC/UCF back-end with a TASK statement. The specified task code must match the task code specified in the DC front-end system definition.

**TSO Front-end**

The UCF TSO front-end enables DC/UCF tasks to be executed from a terminal connected to TSO.
Note: UCF TSO front-end programs contain an options table and a small set of executable code that calls the main processing routines that reside in separate modules.

To create the TSO front-end program, assemble a #UCFTSO macro and link edit the resulting object module as shown in the following section.

Note: The TSO front-end supports dedicated mode only and requires no front-end table definition.

IDMSOPTI

The IDMSOPTI module identifies the back-end system with which the front-end will communicate. The module is created by assembling an IDMSOPTI macro. If you omit IDMSOPTI from the link edit of the front-end system, the execution JCL for the front-end must include a SYSCTL file that identifies the back-end system.

Note: For more information on IDMSOPTI, see Setting Up Interpartition Communication and the SVC (see page 58).

Note: The TSO front-end load module is link edited into the CA IDMS load library during installation as module RHDCUCF.T. You should reassemble and relink edit the module only if the default options need to be changed.

The following is the syntax for the #UCFTSO macro. The macro must be labeled.

UCFTSO Syntax

```
label #UCFTSO

BREAKIN = YES NO
.RSHWKEY = 24
.SYSTEM = TSO
.TONE = YES NO
.VTAM = YES NO
.BUFSIZ = 0
.RESETKB = ASIS TASKEND
.OPSYS = MVS
```
UCFTSO Parameter

- **SYSTEM**
  Specifies the one- through eight-character identifier of the TSO front-end. The front-end system-id must match the identifier specified in the FESID parameter of the #FESTENT macro used to define the TSO front-end system in the UCF system table (described under #FESTENT Macro) (see page 158). TSO, the default, matches the FESID value for the TSO front-end supplied with the UCF system.

- **BREAKIN=NO|YES**
  Specifies whether the front-end will use the BREAKIN option when issuing TPUTS to non-3270 terminals; the default is NO.

- **RSHWKEY**
  Specifies the value of the function key that will be used by TSO to queue a dummy input when an UCF TSO session is interrupted by a message from the operator or by a notify. The dummy input (PF1 through PF24 or PA2) signals the front-end program to try to reshow the screen displayed before the interruption. The value 64 corresponds with PA2; 1 corresponds with PF1, and so on.

- **TONE=NO|YES**
  Specifies whether the front-end program will run under TONE rather than TSO; the default is NO. If YES is specified, the program will issue STAX macros rather than STTMPMD macros.

- **VTAM=NO|YES**
  Specifies whether TSO will run under ACF/VTAM; the default is YES. If YES is specified, the TSO front-end can issue GTTERM macros to determine if a 3279-type color terminal with extended data screen support is being used. The TSO front-end will use the TPG macro to execute 3270 read-buffer operations that support the UCF print-key facility.

- **BUFSIZ**
  When VTAM=YES is specified, GTTERM macro is used to obtain the terminal screen size. Buffer size is calculated to be 1.5 times the terminal screen size, plus a small amount of padding. This calculated amount is compared to any BUFSIZ specification (the default is 0). The higher value of the two is used. The calculated size is typically sufficient for processing. IBM 3270-type terminals using extended (color) datastreams, or MAPs having many fields may need additional buffer space. BUFSIZ may be used to increase the buffer space allocation.

- **RESETKB**
  Specifies whether the terminal keyboard will be reset after each terminal write. Default is 'ASIS,' where the keyboard will be reset after each terminal write. An occasional 'X-F' (input inhibit) may still appear, however. If 'TASKEND' is specified, the keyboard will be reset only at end-of-task. This causes one extra terminal write per task, but will eliminate the occasional input inhibit. This may be of particular value in an IDBCOMM environment.

- **OPSYS**
  Default is MVS. The STTMPMD macro does not contain the same parameters as the IBM version.
TCAM=NO|YES
Specifies whether TSO will run under TCAM. Default is NO. If TCAM=YES is specified, VTAM=NO is required. TCAM is no longer a supported monitor.

Assembly JCL
This section contains the JCL to create the TSO front-end load module for z/OS.

To create a TSO front-end load module assembly

1. Create a UCFTSO source module as follows:
   
   ```
   #UCFTSO ucftso-parameters
   END
   ```

2. Save the UCFTSO source module in your custom source library.

3. Assemble and link it into your custom load library by executing the z/OS Assemble and Link-Edit JCL (see page 465).
   Substitute the name of your UCFTSO source member and insert the following binder statements:
   
   ```
   INCLUDE CUSTLIB(idmsopti)
   ENTRY UCFTSOX
   SETOPT PARM(REUS=NONE,AMODE=31,RMODE=24)
   NAME ucftso(R)
   ```

   - `idmsopti`
     Specifies the name of your IDMSOPTI module. This is optional.

   - `ucftso`
     Specifies the name of your UCF TSO front-end load module.

Front-end execution
To execute the TSO front-end, use the TSO CALL verb, as shown below:

```
CALL 'your-custom-loadlib(ucftso)'
```

⚠️ **Note:** CALL can also be invoked through a TSO CLIST.

The UCF CLIST should contain an ALLOC statement for FILE(SYSIDMS), which points to the SYSIDMS file containing the CVRETRY=OFF parameter. This parameter causes an error to be returned to the program if the back-end system is not active. An example of a UCFTSO CLIST can be found in the sample JCL library offloaded at install time as member UCFTSO.

UCF Back-End

Contents
The UCF back-end exists in the DC/UCF runtime region/partition.

UCF uses a system table to control access to the back-end system by front-end programs and terminals. The system table contains the following information:

- The maximum number of front-end systems that can access the back-end concurrently
- The initial status of each front-end system (that is, online or offline)

A system table named RHDCFSTB is installed with UCF. This table permits access to the UCF back-end by any UCF front-end system that has been defined with a system identifier of: BATCH, CICS, z/VM, DC, or TSO. If you ensure that only front-end system identifiers from the above list are used, a new UCF system table need not be assembled. A system identifier need not reflect the actual TP monitor; for example, a CICS front-end could be defined with a front-end system identifier of CICS.

The UCF system table is created by assembling two macros: #FESTDEF and #FESTENT. The resulting object module must be linked as a separate load module. Each macro is discussed separately below, followed by a sample system table definition and the JCL used to link edit the system table.

The UCF system table is loaded at startup. The "DCMT Vary UCF Front-end System Table New Copy" command can be used to load an altered table without cycling the DC/UCF region or partition.

**FESTDEF Macro**

The #FESTDEF macro is used to create the header for the UCF system table. #FESTDEF specifies the maximum number of front-end systems permitted to access the UCF back-end concurrently. One #FESTDEF macro is assembled for each UCF back-end. #FESTDEF must be the first macro in the source file.

The following is the syntax for the #FESTDEF macro. The macro must be labeled; the label provides the module's entry point name.

**FESTDEF Syntax**

```plaintext
label #FESTDEF
CNT = system-count
```

**FESTDEF Parameter**
• **CNT=**
  Specifies the maximum number of front-end systems that can be defined in the UCF system table. *system-count* must be an integer in the range 1 through 32767.

**FESTENT Macro**

The **#FESTENT** macro is used to create an entry in the UCF system table for each front-end system. **#FESTENT** identifies the front-end systems that are permitted to access the UCF back-end and the type of processing permitted from that front-end.

One **#FESTENT** macro must be coded for:

- Each front-end system that is to access the back-end using UCF
- Each front-end system that supports external request unit processing

The number of **#FESTENT** macros cannot exceed the value specified in the CNT parameter of the **#FESTDEF** macro.

The following is the syntax for the **#FESTENT** macro. The macro must be unlabeled.

**FESTENT Syntax**

```
```#FESTENT
FESID = front-end-system-id
.BLKSIZ = packet-size
.ISTAT = ONLINE OFFLINE
.MAX = max-concurrent-session-count
```

**FESTENT parameter**

- **FESID=**
  Specifies the one- through eight-character name of the front-end system. For UCF processing, *front-end-system-id* must match the identifier specified in the **#UCFUFT** macro or, for the batch, z/VM, and TSO front-ends, the system name specified in the **#UCFBTCH**, **#UCFCMS**, or **#UCFTSO** macro.
  For external request unit processing, the following naming conventions apply:

  - The first four characters of *front-end-system-id* are one of the following values:
    - BATC (the default), specifying batch communication
    - *system-name*, as specified in the **TPNAME** parameter of the **IDMSINTC** macro, specifying the CICS system from which the request comes
- DCXX, specifying all DC systems from which request units may be initiated

- The last four characters of *front-end-system-id* are BULK.

- **BLKSIZ**

  Specifies the size, in bytes, of the packet to use for a batch external request. *packet-size* is an integer in the range 0 through 32000; the default is 4096. Smaller packets use less of the z/OS common system area (CSA) while larger packets use less CPU on both the front-end and the back-end for buffering.

- **ISTAT=ONLINE|OFFLINE**

  Specifies the initial status of the back-end system when the DC/UCF system starts up as follows:

  - **ONLINE** (default) -- The back-end system will be online at system startup. Connection requests from the system’s front-end terminals will be accepted by the DC/UCF system.

  - **OFFLINE** -- The back-end system will be offline at system startup. Connection requests from the system’s front-end terminals will not be accepted until the system is varied online with a DCMT VARY UCF SYSTEM command.

- **MAX=**

  Specifies the maximum number of concurrent sessions allowed for *front-end-system-id*, in the range -1 through +32767. The default value, -1, indicates that there is no limit. When the maximum number of sessions is reached, further connections are not allowed.

### Sample System Table Definition

The following statements define a UCF system table with BATCH, TSO, and CICS front-end systems and two systems that support external request units:

```
#MOPT  CSECT=RHDCFSTB,ENV=USER
COPY  #UCFDS
RHDCFSTB CSECT
FESTABLE  #FESTDEF  CNT=5
  #FESTENT  FESID=BATCH
  #FESTENT  FESID=TSO
  #FESTENT  FESID=CICS
  #FESTENT  FESID=BATCBULK, BLKSIZ=12000
  #FESTENT  FESID=CICPBULK, BLKSIZ=2000
END  FESTABLE
```

### System Table JCL

This section contains the JCL to assemble and link edit the UCF system table for z/OS and z/VSE operating systems.

**z/OS UCF system table assembly and link edit**

**To create a z/OS system table:**

1. Create a UCF system table source module as follows:
COPY #UCFDS DSECT for UCF control blocks
rhdcfstb CSECT
fstbep #FESTDEF CNT=system-count
#FESTENT macros
END fstbep

2. Save the source module in your custom source library.

3. Assemble and link it into your custom load library by executing the z/OS Assemble and Link-Edit JCL (see page 465).
Substitute the name of your source member and insert the following binder statement:

ENTRY fstbep
NAME rhdcfstb(R)

- **fstbep**
  Entry point name of the system table.

- **rhdcfstb**
  Name of the system table.

- **system-count**
  Number of #FESTENT macros.

**z/VSE UCF system table assembly and link edit**

**To create a z/VSE system table:**

1. Assemble and catalog the module using the sample JCL in z/VSE Assemble JCL (see page 476).
Modify the JCL by substituting the following in place of the Assembler input statements:

```asy
copy rhdcfstb csect
fstbep #festival cnt=system-count
#festival macros
end fstbep
```

2. Link the system table using the sample JCL in z/VSE Link JCL.
Modify the JCL by substituting the following statements in place of the Linkage editor control statements:

```zvse
phase rhdcfstb, *
include rhdcfstb setopt parm(amode=31, rmode=any)
```

- **fstbep**
  Entry point name of the system table.

- **rhdcfstb**
  Name of the system table.

- **system-count**
  Number of #FESTENT macros.
UCF System Generation Statements

To define a UCF system, include the following system generation statements in the DC/UCF system definition:

- One LINE statement that specifies TYPE IS UCFLINE.
- One PTERM statement for each UCF front-end physical terminal. The number of PTERM statements with TYPE IS UCFTERM equals the maximum number of UCF front-end terminals that can access the UCF back-end at one time.
- One LTERM statement for each physical terminal.
- One TASK statement for each intermittent task. The task code specified must match the task code specified in the host TP-monitor system definition. The dedicated task does not require a TASK statement.
- One PROGRAM statement for each program associated with each task intermittent task.
- Two TASK statements to define the BYE and SUSPEND system tasks.

CA IDMS CICS Considerations

DC/UCF supports the following TP-monitors: CICS, CMS (z/VM), CA IDMS, DC, TSO, and batch. When using any of these TP monitors, you can execute programs that use CA IDMS central version database services.

Each CA IDMS program that executes under one of these TP monitors must be link edited with the appropriate program interface module. To execute CA IDMS central version programs under TP monitors other than CICS, use the standard CA IDMS batch interface or a custom TP-monitor interface. This section describes the interfaces you need for CICS.

The CA IDMS CICS interface consists of three components:

- A stub module generated by the IDMSCINT or IDMSCINL macro that is linked with application programs requiring CA IDMS services.
- A stub module that is linked with a site-specific CICSOPT options module and exit routines.
- A site-independent interface module called IDMSCCMN that contains containing only CA-supplied code.

Both the standard IDMSINTC and the IDMSINTL interfaces conform to this architecture.

For more information, see the following topics:

- Standard CICS Interface (see page 162)
- IDMSINTL and IDMSCINL CICS Interface Macros (see page 183)
The standard CA IDMS interface requires CICS at a version level that is supported by IBM. Additionally, CA IDMS requires a minimum CICS level as follows:

- CICS Transaction Server for z/OS V3.1
- CICS Transaction Server for z/VSE V1.1.1

Check ca.com/support for compatibility with later releases of CICS.

A standard site-specific CICS interface is generically referred to as IDMSINTC throughout this document. A basic site-specific version of the standard CICS interface program is installed and configured during installation. That basic version is linked as z/OS load module or z/VSE phase IDMSINTC. Other versions of the interface module can be created and configured with site-selected names. Normally, a separate z/OS load module or z/VSE phase is required for each CA IDMS central version that is to be accessed. If the proper options are chosen, a single version of the site-specific interface can be used to access a given CA IDMS central version from multiple CICS regions.

Each site-specific CICS interface load module or phase is composed of several precompiled objects as well as an options table that is compiled on site during the installation process as follows:

- The CICSOPT macro is compiled onsite and provides a table of your selected options. The macro for the basic version of the interface is compiled during installation.
IDMSCSTB is a Stub module that contains minimal code to access a common CICS interface module, IDMSCCMN. The Stub module provides a linkage between the CICSOPT options table and the common module.

IBM supplied CICS interface modules DFHEAI and DFHEAI0.

An optional OPTIXIT.

An optional OPTIQXIT.

An optional USRIDXIT. For more information see the section USRIDXIT (see page 229).

For more information on the OPTIXIT and OPTIQXIT parameters, see the CICSOPT Parameters (see page 168).

If maintenance is applied to IDMSCCMN, it is not necessary to recompile or relink any site-specific interface modules. A new site-specific interface would only be needed to implement new or different options.

Each interface module must be defined in the CICS CSD. The program name is automatically detected and used at runtime to establish a TRUE (Task Related User Exit).

**IDMSINTC interface requirements**

The following requirements must be observed when using the IDMSINTC interface:

- Ensure that every CICS system has a consistent and unique identifier. You can do this by changing TPNAME parameters or by specifying a CICS_NAME SYSIDMS parameter.

- If you use the auto-commit feature to commit your CA IDMS database transactions through a CICS syncpoint operation, you must take additional steps to implement two-phase commit support between CICS and CA IDMS For more information, see Two-Phase Commit Support with CICS.

> **Note:** This requirement also applies to CA IDMS VSAM Transparency users.

- Install the CA IDMS entity definitions into the CICS CSD. These definitions are contained in the CAGJSRC library member CICSCSD. Consult the appropriate IBM documentation to ensure that these definitions take precedence over any previously installed definitions for the corresponding entities.

- You can no longer use macro-level CICS programs to access CA IDMS.

- Ensure that the DFHRPL concatenation in the CICS startup JCL contains the IDMS custom.loadlib and CAGJLOAD libraries.
IDMSINTC Startup

When the IDMSINTC program is started, through the PLT or by invoking its associated TRANSID, it performs the following functions:

- Stores the address of the IDMSINTC entry point address in the CWA
- Reads any SYSCTL files, specified or implied, by the CICSOPT SYSCTL and MAXCVNO parameters
- Establishes the CA IDMS front-end runtime environment
- If using CICS Transaction Server with two-phase commit enabled for IDMSINTC, and there are outstanding units of work (UOW) at CICS startup, resynchronization processing is initiated to ensure database integrity.

**Note:** It is recommended that the same TPNAME value be specified or defaulted for all IDMSINTC interfaces used in a single CICS region. If different TPNAME values are used in two IDMSINTC interfaces in the same region, the CICS startup JCL must contain a SYSIDMS parameter that specifies a CICS_NAME value.

Automatically starting IDMSINTC

It is useful to have the IDMSINTC interface module started automatically at CICS startup. To do this, perform the following steps:

1. Code the following entry for IDMSINTC in the program list table. The IDMSINTC entry should follow the one for DFHDELIM so that IDMSINTC is executed during phase 3 of CICS initialization.

   ```
   DFHPLT  TYPE=ENTRY
   PROGRAM=IDMSINTC
   ```

2. Enter the name of the program list table (PLTPI) in the system initialization table.

If you do not have IDMSINTC started as a function of the CICS startup routine, be sure to execute the IDMSINTC interface module before attempting communication with DC/UCF at runtime. To execute the IDMSINTC module, enter a TRANSID that corresponds to the IDMSINTC module in the PCT tables.

To start up IDMSINTC automatically after control is given to CICS, perform the following steps:

1. Code the following operands on the CICSOPT macro:

   ```
   PLT=NO,
   TRANSID=task-code
   ```
2. Perform steps 1 and 2 above and define task-code in the program control table (PCT) to invoke IDMSINTC.

⚠️ **Note:** For more information on the PCT table, see UCF Operations.

### Runtime processing

IDMSINTC must be resident in CICS and must be executed before the execution of any CA IDMS database application for that instance of the CICS region.

### Application processing

When an application program accesses a back-end DC/UCF system, control is passed to the IDMSCINT module linked with the program. Using the CWADISP value, IDMSCINT determines the address of the appropriate entry point in the IDMSINTC interface.

IDMSINTC performs the following functions:

- Allocates dynamic storage required by the CA IDMS interface to service the request.
- Enables the recovery exit program for the current CICS task. The exit program, IDMSTRUE, performs the necessary cleanup when the CICS task terminates.

⚠️ **Note:** For more information on IDMSTRUE, see **IDMSTRUE exit**.

- Passes control to CA IDMS, which sends the request to the DC/UCF region. The CA IDMS interface implicitly issues a CICS WAIT command, which places the current task in a wait state until the request is serviced.
- Passes the requested database record and/or error status to the user program.

⚠️ **Note:** For more information on runtime processing, see What Happens when a CA IDMS Instruction is Executed.

### IDMSTRUE exit

IDMSTRUE is a task termination exit, which is defined to CICS as a Task Related User Exit (TRUE).

IDMSTRUE features include the following:

- Standard CICS facilities (that is, it doesn't modify CICS control programs)
- Minimal overhead; IDMSTRUE gets control only for tasks performing CA IDMS database calls.
At task termination, IDMSTRUE detects the following conditions:

- Transactions still active at abnormal task termination. When IDMSTRUE detects this condition, database updates associated with the transaction are backed out immediately. This conserves the resources that would be held until the IDMS EXTERNAL WAIT INTERVAL expires or the DC/UCF system is recycled.
  To monitor transactions active at abnormal task termination, use the DFHPEP examiner or the SCP user-exit examiner.

- Transactions still active at normal task termination.
  This condition usually indicates a program logic error. That is, the program issued a BIND but never issued a FINISH. When IDMSTRUE detects this condition, database updates associated with the BIND are backed out immediately. This conserves the resources that would be held until the IDMS EXTERNAL WAIT INTERVAL expires or the DC/UCF system is recycled.

⚠️ **Note:** The action described previously can be modified through the use of the AUTOCMT, AUTONLY, ONBACK, and ONCOMT parameters. For more information on these options, see **CICSOPT syntax** parameters and the IDMSINTC macro.

⚠️ **Note:** Transparency applications do not explicitly issue the BIND RUN-UNIT and FINISH. Rather, a Transparency transaction's first request results in an implicit BIND RUN-UNIT. IDMSTRUE issues an implicit FINISH when the task terminates.

**CICSOPT Syntax**

```plaintext
CICSOPT CWADISP=cwa-intc-address-disp

OPSYS=operating-system

.AUTOCMT= (ON OFF)

.AUTONLY= (ON OFF)

.CVNUM= cv-number

.DBNAME=(db-name)

.DBUGDCT= destination-name

.DEBUG= (YES NO QTS)

.DICTNAM= dictionary-name
```
The following parameters are for use in a CICSOPT module tailored for SQL:

- **DICTNAM**
- **DICTNOD**

### CICSOPT Parameters

- **module-name**
  Identifies the CSECT name of the generated module. The default is CICSOPT.

- **CWADISP**
  Identifies the displacement within the CICS CWA of a fullword containing the address of an entry point within the CICSOPT module.

  - **cwa-intc-address-disp**
    must specify an integer representing an offset within the CWA. The lowest allowable value is zero. The highest allowable value is 4 less than the size of the CWA. This displacement within the CWA must not be used for any other purpose by any CICS application. The value specified must begin on a fullword boundary and be the same value that is specified in the CWADISP parameter of the IDMSCINT macro.

- **OPSYS**
  Identifies the operating system under which the DC/UCF (or CICS) system will run.

  - **operating-system**
    The valid values are the following:
    - OS390
    - MVS
    - VSE
    - DVS
    - DOS
    - DOSVS

- **AUTOCMT**
  Specifies whether database sessions opened by a program using this interface module are eligible for participation in a CICS UOW (Unit of Work).
- **ON**
  Specifies that database sessions are eligible to participate in a CICS unit of work (UOW). If the database session is active at the time a CICS syncpoint operation is performed, the session’s updates are committed as part of the CICS UOW.

- **OFF**
  Specifies that database sessions are not eligible to participate in a CICS unit of work (UOW). If TXNSHR=ON is specified, the default for AUTOCMT is ON; otherwise it is OFF. An assembly error results if TXNSHR=ON and AUTOCMT=OFF are specified.
  ALWAYS Specifies that the AUTOCMT behavior specified in the CICSOPT parameter overrides whatever was specified in the IDMSCINT module with which the application is linked.
  DEFAULT Specifies that the AUTOCMT behavior specified in the CICSOPT parameter applies only if the corresponding IDMSCINT parameter specifies DEFAULT. This is the default.

- **AUTONLY**
  Specifies if database sessions opened by a program using this interface module are forced to participate in a CICS UOW.

  - **ON**
    Specifies that database sessions are forced to participate in a CICS Unit Of Work (UOW). Even if the database session is terminated prior to the CICS syncpoint operation, the session’s updates are committed as part of the CICS UOW. DML commands that would normally cause the session's updates to be committed (such as FINISH or COMMIT WORK) have no impact on the session's transaction, although they do impact the session. Conversely, if the session's transaction is forced to back out (either because of a DML ROLLBACK request or because of events such as a deadlock), a CICS SYNCPOINT BACKOUT is issued forcing the entire CICS UOW to be backed out.

  - **OFF**
    Specifies that database sessions are not forced to participate in a CICS Unit Of Work (UOW). If TXNSHR=ON is specified, the default for AUTONLY is ON; otherwise it is OFF. An assembly error results if TXNSHR=ON and AUTONLY=OFF are specified.
    ALWAYS Specifies that the AUTONLY behavior specified in the CICSOPT parameter overrides the specifications in the IDMSCINT module with which the application is linked.
    DEFAULT Specifies that the AUTONLY behavior specified in the CICSOPT parameter applies only if the corresponding IDMSCINT parameter specifies DEFAULT. This is the default.

- **CVNUM**
  Identifies the number of the DC/UCF system to be accessed from CICS.

  - **cv-number**
    For cv-number, specify the number used for the CVNUM parameter in the system definition of the DC/UCF system.

- **DBNAME**
  Identifies the database (or data dictionary) name to be contained in the CICSOPT module. This parameter also identifies the conditions under which programs signing on to the DC/UCF system access the named database. This parameter has no effect on SQL database sessions. For SQL sessions, see the DICTNAM parameter.

  - **db-name**
    For db-name, specify the name of the database that programs are to access when running under the DC/UCF system. If the database name is not specified, DC/UCF obtains the
appropriate database name from the application program or from the SYSCTL file (z/OS only).
ALWAYS Indicates that \textit{db-name} is to override any database named by the program. Programs
signing on to DC/UCF always execute against the named database regardless of database
name specifications made by the program.
DEFAULT Indicates that programs signing on to DC/UCF are to execute against the named
database only if the program does not name a database.

\textbf{Note:} Under z/OS and z/VSE, SYSCTL database name specifications can override CICSOPT
and program specifications.

- **DBUGDCT**
  Identifies the CICS transient data or temporary storage destination to use as the target for error
  messages produced if DEBUG=YES or DEBUG=QTS is specified.

  - \textit{destination-name}
    The default \textit{destination-name} is DEBUG. Use another destination if you want to route
diagnostic messages to another CICS destination. If DEBUG=YES is specified, the DCT entry
should be defined with a variable length record of at least 136 characters. We recommend
that you use the values provided in source library member CICSCSD.

- **DEBUG**
  Specifies whether IDMSINTC produces extra debugging information on internal processing.

  - \textit{YES}
    Specifies that IDMSINTC produces extra information. This information is written using WRITEQ
TD to the destination specified on the DBUGDCT parameter.

  - \textit{NO}
    Specifies that IDMSINTC does not produce debugging information. This is the default. You
should always use DEBUG=NO unless otherwise requested by Technical Support to resolve a
system problem.

  - \textit{QTS}
    Specifies that IDMSINTC produces extra information. This information is written using WRITEQ
TS to the queue specified on the DBUGDCT parameter.

- **DICTNAM** \textit{dictionary-name}
  Sets the dictionary to which an SQL session will be connected unless it is overridden by the
application program. This parameter has no effect on non-SQL database sessions.

- **DICTNOD** \textit{dictionary-node}
  Sets the node to which an SQL session will be connected. This parameter has no effect on non-
SQL database sessions.

- **DL1MAC**
  Always NO. Provided solely for upward compatibility.
- **DSECT**
  Specifies whether to generate a DSECT only version of the CICSOPT macro.
  The parameter defaults to YES. However, to generate a working CICS interface module, you must specify DSECT=NO in the assembly of CICSOPT.

- **ERRDCT**
  Identifies the CICS transient data destination to use as the target for error messages produced by IDMSINTC and IDMSTRUE.

  - **destination-name**
    The default destination-name is CSMT. Use another destination if you want to route CA IDMS error messages to another CICS destination. The DCT entry should be defined with a logical record length of at least 130 characters.

- **ESCDLI**
  Specifies whether DLI/T transparency is in use with this interface.

- **EXTXIT**
  Specifies whether the IDMSINTC interface loads and calls the general use exits IDMSCEON and IDMSCEOX. The IDMSCEON exit is called before the requested IDMS function call is serviced. The IDMSCEOX exit is called immediately after the requested IDMS function call has been processed. The default is NO.

- **HLPI**
  Specifies whether HLPI support is required for DL1.

- **MACLVL**
  This parameter is obsolete. Macro level programs are no longer supported in CICS.

- **MAXCVNO additional-sysctl-cnt**
  Specifies the number of additional SYSCTL DD cards that can be accessed through this interface. DDNAMEs for the additional SYSCTL files are derived by replacing the first blank or the last character (if all 8-bytes are in use) of the SYSCTL operand value by the numbers 1 through MAXCVNO.

- **MAXCON**
  Specifies the maximum number of different back-end central versions that a CICS task can access simultaneously through this CICS interface module. This limit applies only to database sessions for which AUTOCMT is enabled. If an application uses different interface modules, each one has its own limit.

  - **maximum-connections**
    Must be a numeric value between 1 and 1000. If maximum-connections is not specified, the default maximum number of connections is 2.

- **MAXIDMS**
  Specifies the maximum number of different back-end central versions that a CICS interface module can access throughout the life of a CICS system. This limit applies only to database sessions for which AUTOCMT is enabled. If an application uses different interface modules, each one has its own limit.
- **maximum-IDMS-systems**
  Must be a numeric value between 1 and 1000. The default maximum number of back-end systems is the larger of 2 and 2 \* the value of the MAXCVNO parameter.

- **NODENAM**
  Identifies a system defined to the DC/UCF communications network to be contained in the CICSOPT module and the conditions under which programs signing on to the DC/UCF system will be directed to the named node for execution. This parameter has no effect on SQL database sessions. For SQL sessions, see the DICTNOD parameter.

  - **nodename**
    For nodename, specify the one- to eight-character name of a remote system. If nodename is not specified, the DC/UCF obtains the appropriate node name from the application program or from the SYSCTL file (z/OS only).
    - ALWAYS Indicates that nodename is to override any node named by the program. Requests from programs signing on to DC/UCF are always directed to the named node regardless of node name specifications made by the program.
    - DEFAULT Indicates that requests from programs signing on to DC/UCF are to be directed to the named node only if the program does not name a node.

  **Note:** Under z/OS and z/VSE, SYSCTL node name specifications can override CICSOPT and program specifications.

- **ONBACK**
  Specifies the action that should be taken for database sessions opened by a program using this interface module when they participate in a CICS backout operation.

  - **ROLLBACK**
    Specifies that database sessions should be terminated. This is the default.

  - **ROLLBACK-CONTINUE**
    Specifies that database sessions should continue but currencies freed.

  - **ALWAYS**
    Specifies that the ONBACK behavior specified in the CICSOPT parameter overrides whatever was specified in the IDMSCINT module with which the application is linked.

  - **DEFAULT**
    Specifies that the ONBACK behavior specified in the CICSOPT parameter applies only if the corresponding IDMSCINT parameter specifies DEFAULT. This is the default.

- **ONCOMT**
  Specifies the action that should be taken for database sessions opened by a program using this interface module when they participate in a CICS syncpoint operation.

  - **COMMIT-ALL**
    Specifies that database sessions should continue but currencies freed.

  - **COMMIT-CONTINUE**
    Specifies that database sessions should continue and currencies retained.
- **FINISH**
  Specifies that database sessions should be terminated. This is the default.

- **ALWAYS**
  Specifies that the ONCOMT behavior specified in the CICSOPT parameter overrides the specification in the IDMSCINT module with which the application is linked.

- **DEFAULT**
  Specifies that the ONCOMT behavior specified in the CICSOPT parameter applies only if the corresponding IDMSCINT parameter specifies DEFAULT. This is the default.

- **OPTIXIT**
  Indicates whether CICS transactions can modify the IDMSOPTI structure dynamically so that only the task thread is affected by the changes. YES indicates that the IDMSOPTI structure can be modified dynamically.
  IDMSINTC copies the static IDMSOPTI structure into dynamic storage and passes it to the user routine, which may alter it based on site-specific rules.

  **Note:** For more information on the OPTIXIT user exit, see OPTIXIT.

- **PLT**
  Indicates how IDMSINTC starts up.

  - **YES**
    Specifies that IDMSINTC can start up as a PLT-invoked program.

  - **NO**
    Specifies IDMSINTC always starts up as a user task once CICS start up is complete.

- **RSYNTXN**
  Specifies the name of the CICS resynchronization transaction defined for this interface.

  - **rsyn-transaction-name**
    Must be the name of a transaction defined to CICS and associated with a resynchronization program. If not specified, the default transaction name is RSYN.

- **SVC**
  Identifies the number of the CA IDMS SVC.

  - **svc-number**
    For **svc-number**, specify a value as follows:
    - If no SVC is being used, or if using SYSCTL, specify NO.
    - If an SVC is being used by the DC/UCF system, specify the SVC number.

    The SVC parameter is required if no SYSCTL file is specified.
• **SYSCTL ddname**
  Identifies the *ddname* of the file containing DC/UCF system control information. If the SVC (described previously) is *not* specified, the SYSCTL parameter is required. Likewise, if SYSCTL is desired, the SVC parameter must be NO (SVC=NO).

• **TIMEOUT**
  Indicates how long to hold a block of storage after termination of a CICS task that utilizes CA IDMS services. This storage block is acquired with a CICS GETMAIN and is initialized when a terminal makes a request for IDMS services. The valid values are as follows:

  - 0
    Specifies that once a block of storage is acquired for a particular CICS terminal that storage is not freed until one of the following occurs:
    - The user signs off the terminal.
    - A task aborts that is using that storage block.

The following situations apply to the TIMEOUT parameter:

  - **TIMEOUT=IMMEDIATE** is recommended *unless* the IDMSINTC interface is used with SQL transactions that utilize SUSPEND/RESUME processing. These type of transactions require a TIMEOUT value other than IMMEDIATE.

  - **TIMEOUT=IMMEDIATE** must be specified for any IDMSINTC module that runs in a CICS Application Owning Region (AOR) and which is invoked from terminals in a separate Terminal Owning Region (TOR).

  - Since several thousand bytes are held by each terminal, a non-IMMEDIATE value may result in a large amount of storage being held for terminals with no active tasks.

  - A TIMEOUT value of 1 through 10 must not be coded if any application program using IDMSINTC services also uses READQ TS ... with the SET option.

  - If the INTCID parameter and/or the NTID parameter of the #UCFCICZ macro are not coded correctly or if the UCFCICZ exit is not driven for some other reason, the IDMS storage block may not be freed when a CICS user is logged off a terminal. The next user who signs on that terminal may acquire that storage block. This can result in S019 or other S0nn abends or other application problems. If such problems are encountered, use TIMEOUT=IMMEDIATE.

  - **IMMEDIATE**
    Specifies that the storage is freed at task termination of the task that acquired the storage. If IMMEDIATE is coded, a new storage block is acquired and initialized for each CICS task that requests CA IDMS services. This is the default.

  - **number-of-minutes**
    Specifies the number of minutes storage is held. *number-of-minutes* can range from 1 to 10. If another CA IDMS request is made within that time, the same block of storage is utilized. If the storage is not used within that time, it is freed. The next request for CA IDMS services results in the acquisition and initialization of a new storage block.
**TPNAME system-name**
Specifies the name by which DC/UCF will identify all tasks running under this CICS system. For `system-name`, specify a four-character name.

If TPNAME is omitted or the `system-name` is specified as spaces, the `system-name` defaults to the four-character local CICS system ID.

The TPNAME value must be unique across all CICS systems that access any single central version.

If more than one IDMSINTC interface is used within a CICS system, they should all specify the same value for TPNAME. Otherwise, a CICS_NAME SYSIDMS parameter must be used to provide a unique identifier for the CICS system.

This name forms the first part of the local transaction ID for database requests and forms the first four characters of the front-end system ID for external request units. "BULK" is appended to the `system-name` to create the front-end system ID. The front-end system ID can be used:

- To access front-end system table to determine the packet size for communications and the maximum number of simultaneous requests.
- As an alternate task code for controlling external request unit processing.

**Note:** The front-end system ID is used by some Performance Monitor reports to determine the type of reporting to be done. Front-end systems with an ID which begins with "CI" are assumed to be CICS systems. Therefore, certain CICS statistics may not be reported if the front-end system ID is set to or defaults to a value which does not begin with CI.

**TRANSID**
Identifies the transaction coded in the program control table (PCT) as invoking IDMSINTC.

- `task-code`
  `task-code` must be the name of a task defined in the PCT table.
  **Note:** For more information on the PCT table, see UCF Operations.

**TRUE**
Specifies a prefix to be used in forming Task Related User Exit (TRUE) entry names.

- `true-prefix`
  Must be a one to five character value that is unique across all interface modules in use within a CICS system. If `true-prefix` is less than five characters, it is padded on the right with $'s. If not specified, the default prefix is constructed as the last five characters of the IDMSINTC module name, padded on the right with $'s if necessary.

**TRUEAPI**
Specifies whether to enable the IDMS TRUE exit with the OPENAPI attribute.

- **CICS**
  Specifies to enable the exit with the THREADSAFE attribute.

- **OPEN**
  Specifies to enable the exit with the THREADSAFE and OPENAPI attributes.

**TXNSHR**
Specifies whether database sessions opened by a program using this interface module should share the same transaction as other sessions started by the same CICS task.
ON
Specifies that database sessions should share transactions.

OFF
Specifies that database sessions should not share transactions. This is the default.

ALWAYS
Specifies that the TXNSHR behavior specified in the CICSOPT parameter overrides whatever was specified in the IDMSTCINT module with which the application is linked.

DEFAULT
Specifies that the TXNSHR behavior specified in the CICSOPT parameter applies only if the corresponding IDMSTCINT parameter specifies DEFAULT. This is the default.

Note: For more information on transaction sharing, see the Database Administering section.

USERCNT
Specifies the maximum number of concurrent CICS tasks using the CA IDMS interface. Any task that attempts to access CA IDMS through the interface while at the maximum will be aborted with a K214 transaction abend.

max-concurrent-CICS-tasks
Valid values are 1 to 100000. The default value is 100.

XA
Designates whether the operating system is capable of processing XA addresses. If you specify YES, IDMSTCINT allocates the primary user-oriented storage in the 31-bit storage area. This storage is retained across all successful task terminations for terminal-associated tasks, and this storage is reused on the next DC/UCF request. The storage is freed for any failing or non-terminal task.

Note: The CICSOPT macro allows other parameters which are not documented in the previous section. Some of these parameters are obsolete and are supported solely to maintain upwardly compatible syntax. Others are relevant only to clients who have the Transparency Option for TOTAL, DL/I, or VSAM. Unless otherwise documented, parameters other than those shown do not have to be coded when invoking the CICSOPT parameter.

Parameter Considerations

Note: The CICSOPT macro allows other parameters which are not documented in the previous section. Some of these parameters are obsolete and are supported solely to maintain upwardly compatible syntax. Others are relevant only to clients who have the Transparency Option for TOTAL, DL/I, or VSAM. Unless otherwise documented, parameters other than those shown do not have to be coded when invoking the CICSOPT parameter.

Create an IDMSINTC Interface Program

To create an IDMSINTC interface program under z/OS:

1. Create a CICSOPT source module as follows:

   GBLC &MODNAME
   &MODNAME SETC CICSOPT'
   CICSOPT cicsopt-parameters
   END
1. Save the CICSOPT source module in your custom source library.

2. Assemble and link the module into your custom load library by executing the z/OS Assemble and Link-Edit JCL.

3. Substitute the name of your CICSOPT source member and insert the following binder statements:

   ```
   ORDER DFHEAI
   INCLUDE CUSTLIB((Optional)
   INCLUDE CUSTLIB((Optional)
   INCLUDE CUSTLIB((Optional)
   INCLUDE CAGLOAD(IDMSCSTB)
   ENTRY STARTUP
   SETOPT PARM(AMODE=31,RMODE=24,REUS=NONE)
   NAME idmsintc(R)
   ```

- **optixit**
  Specifies the name of your OPTIXIT exit routine.

- **optiqxit**
  Specifies the name of your OPTIQXIT exit routine.

- **usridxit**
  Specifies the name of your USRIDXIT exit routine.

- **idmsintc**
  Specifies the name of your IDMSINTC interface load module.

To create an IDMSINTC interface program under z/VSE:

1. Assemble and catalog a CICSOPT options table using the sample JCL in z/VSE Assemble JCL (see page 476).
   Modify the JCL by substituting the following in place of the **Assembler input statements**:

   ```
   PUNCH 'CATALOG cicsopts.OBJ REPLACE=YES'
   CICSOPTS TITLE 'CA IDMS CICS OPTIONS MODULE'
   GBLC &MODNAME
   &MODNAME. SETC 'cicsopts'
   CICSOPT cicsopt-parameters
   END
   ```

2. Link the IDMSINTC interface program using the sample JCL in z/VSE Link JCL.
   Modify the JCL by substituting the following statements in place of the **Linkage editor control statements**:

   ```
   PHASE idmsintc,*
   INCLUDE DFHEAI
   INCLUDE optixit (Optional)
   INCLUDE optiqxit (Optional)
   INCLUDE usridxit (Optional)
   INCLUDE usridxit (Optional)
   INCLUDE cicsopts INCLUDE IDMSLST6
   INCLUDE DFHEAI0
   MODE AMODE(31),RMODE(24)
   ENTRY STARTUP
   ```

- **cicsopts**
  Specifies the name you choose for your IDMSINTC options table.
CA IDMS - 19.0

Note: You can have more than one options table with different names.

- **idmsintc**
  Specifies the name you choose for your IDMSINTC stub program.

Note: You can have more than one stub program with different names.

- **optixit**
  Specifies the name of your OPTIXIT exit routine.

- **optiqxit**
  Specifies the name of your OPTIQXIT exit routine.

- **usridxit**
  Specifies the name of your USRIDXIT exit routine.

- **idmsintc**
  Specifies the name of your IDMSINTC interface load module.

The IDMSCINT Application Stub Program

Each application program that accesses CA IDMS must be linked with an IDMSCINT stub program. The IDMSCINT stub program establishes communications with a particular IDMSINTC interface program. The CWADISP specified in the IDMSCINT stub program must match the CWADISP specified in the CICSOPT macro that is used when creating the IDMSINTC interface program. The following sections describe how to create an IDMSCINT application stub program.

IDMSCINT Syntax

To prepare an IDMSCINT macro, use the following syntax.

```
IDMSCINT(CWADISP=cwa-intc-address-displacement)

,AUTOCMT=ON,OFF,DEFAULT

,AUTONLY=ON,OFF,DEFAULT

,DML=YES,NO

,EP1=first-entry-point-label

,EP2=second-entry-point-label

,EXEC=YES,NO

,ONBACK=ROLLBACK
```
IDMSCINT Parameters

This section details the IDMSCINT parameters.

- **module-name**
  Identifies the CSECT name of the generated module.

- **CWADISP=cwa-intc-address-displacement**
  Identifies the displacement within the CICS CWA of a fullword that holds the address of the IDMSINTC module. For `cwa-intc-address-displacement`, specify the same value given to the CWADISP operand of the IDMSINTC macro.

- **AUTOCMT**
  Specifies whether database sessions opened by a program linked with this IDMSCINT module are eligible for participation in a CICS UOW (Unit of Work).
  - **ON**
    Specifies that database sessions are eligible to participate in a CICS Unit Of Work (UOW). If the database session is active at the time a CICS syncpoint operation is performed, the session’s updates are committed as part of the CICS UOW.
  - **OFF**
    Specifies that database sessions are not eligible to participate in a CICS Unit Of Work (UOW).
  - **DEFAULT**
    Specifies that whether database sessions are eligible for participation in a CICS Unit Of Work (UOW) is determined by the AUTOCMT parameter of the interface’s CICSOPT macro.

    If TXNSHR=ON is specified, the default for AUTOCMT is ON; otherwise it is OFF. An assembly error results if TXNSHR=ON and AUTOCMT=OFF are specified.

- **AUTONLY**
  Specifies whether database sessions opened by a program linked with this IDMSCINT module are forced to participate in a CICS UOW.
  - **ON**
    Specifies that database sessions are forced to participate in a CICS Unit Of Work (UOW). Even if a database session is terminated prior to the CICS syncpoint operation, the session’s updates are committed as part of the CICS UOW. DML commands that would normally cause
the session's updates to be committed (such as FINISH or COMMIT WORK) have no impact on
the session's transaction, although they do impact the session. Conversely, if the session's
transaction is forced to back out (either because of a DML ROLLBACK request or because of
events such as a deadlock), a CICS SYNCPOINT BACKOUT is issued forcing the entire CICS UOW
to be backed out.

- OFF
  Specifies that database sessions are not forced to participate in a CICS Unit Of Work (UOW).

- DEFAULT
  Specifies that whether database sessions are forced to participate in a CICS Unit Of Work
  (UOW) is determined by the AUTONLY parameter of the interface's CICSOPT macro.
  If TXNSHR=ON is specified, the default for AUTONLY is ON; otherwise it is OFF. An assembly
  error results if TXNSHR=ON and AUTONLY=OFF are specified.

- DML=YES|NO
  Determines whether the program linked with IDMSCINT can (YES) or cannot (NO) issue non-SQL
  DML statements. YES is the default.

- EP1=first-entry-point-label
  Identifies the name of the first entry point in the generated module. For first-entry-point-label,
  specify the name of the first entry point. The default value is either the value given to module-
  name above or IDMSCINT if module-name is not specified.

- EP2=second-entry-point-label
  Identifies the name of the second entry point in the generated module. For second-entry-point-
  label, specify the name of the second entry point.

Second Entry Point Considerations

- A second entry point is present only if the EXEC parameter described below specifies NO. In
  this case, if EP2 is not specified, the default value is IDMSINC2.

- If the EXEC parameter is coded as YES, no second entry point is generated, and EP2 should not
  be specified.

- EXEC=YES|NO
  Indicates whether the IDMSCINT module being generated is to be used with the CICS command-
  level interface.

  - YES
    Specify YES to indicate that the command-level interface is in use. YES is the default.
    If EXEC=YES, name the module IDMSCINT to allow it to be autolinked with applications.

  - NO
    Specify NO to indicate that the command-level interface is not in use. EXEC=NO must be
    specified if installing the CA IDMS DMS interface.
    If EXEC=NO, name the module IDMSINC1 to allow the use of autolink.

- ONBACK
  Specifies the action that should be taken for database sessions opened by a program linked with
  this IDMSCINT module when they participate in a CICS backout operation.
- **ROLLBACK**
  Specifies that database sessions should be terminated. This is the default.

- **ROLLBACK-CONTINUE**
  Specifies that database sessions should continue but currencies freed.

- **DEFAULT**
  Specifies that the backout action for sessions is determined by the ONBACK parameter of the interface’s CICSOPT macro.

- **ONCOMT**
  Specifies the action that should be taken for database sessions opened by a program linked with this IDMSCINT module when they participate in a CICS syncpoint operation.

  - **COMMIT-ALL**
    Specifies that database sessions should continue but currencies freed.

  - **COMMIT-CONTINUE**
    Specifies that database sessions should continue and currencies retained.

  - **FINISH**
    Specifies that database sessions should be terminated. This is the default.

  - **DEFAULT**
    Specifies that the commit action for sessions be determined by the ONCOMT parameter of the interface’s CICSOPT macro.

- **SQL=YES|NO**
  Determines whether the program linked with IDMSCINT can (YES) or cannot (NO) issue SQL DML statements. NO is the default.

- **SSC12=YES|NO**
  Determines whether the application program contains a PROTOCOL statement that specifies SUBSCHEMA-NAMES LENGTH IS 18. If this is the case, specify YES. Specify NO if the application program contains a PROTOCOL statement that specifies SUBSCHEMA-NAMES LENGTH IS 16 or if the PROTOCOL statement does not contain a SUBSCHEMA-NAMES LENGTH clause.
  Default: NO

- **TXNSHR**
  Specifies whether database sessions opened by a program linked with this IDMSCINT module should share the same transaction as other sessions started by the same CICS task.

  - **ON**
    Specifies that database sessions should share transactions.

  - **OFF**
    Specifies that database sessions should not share transactions. This is the default.

  - **DEFAULT**
    Specifies whether database sessions share transactions is determined by the TXNSHR parameter of the interface’s CICSOPT macro.
Assembling and Link Editing IDMSCINT

This section contains the JCL to assemble and link edit the IDMSCINT module for z/OS and z/VSE operating systems.

**z/OS IDMSCINT assembly and link edit**

**To assemble and link end IDMSCINT**

1. Create a source module as follows:

   ```
   IDMSCINT idmscint-parameters
   END
   ```

2. Save the source module in your custom source library.

3. Assemble and link it into your custom load library by executing the z/OS Assemble and Link-Edit JCL (see page 465). Substitute the name of your source member and insert the following binder statement:

   ```
   NAME idmscint(R)
   ```

4. Add the following statement to the link of each application program that accesses IDMS using this `idmscint` application stub program. The CWADISP specified in the IDMSCINT parameters must match the CWADISP specified in the CICSOPT parameters used in creating an IDMSINTC interface program:

   ```
   INCLUDE idmscint
   ```

   - **idmscint**
     
     Specifies the name of your IDMSCINT interface module.

**z/VSE IDMSCINT assembly and link edit**

**To create an IDMSINTC interface program under z/VSE:**

1. Assemble and catalog an IDMSCINT program using the sample JCL in z/VSE Assemble JCL (see page 476). Modify the JCL by substituting the following in place of the **Assembler input statements**:

   ```
   PUNCH 'CATALOG idmscint.OBJ REPLACE=YES'
   IDMSCINT idmscint-parameters
   END
   ```
2. Add the following statement to the link of each application program that accesses IDMS using this idmscint application stub program. The CWADISP specified in the IDMSCINT parameters must match the CWADISP specified in the CICSOPT parameters used in creating an IDMSINTC interface program:

```
INCLUDE idmscint
```

- **idmscint**
  - Specifies the name you choose for your IDMSCINT application stub program.

**Note:** You can have more than one application stub program with different names.

---

**IDMSINTL and IDMSCINL CICS Interface Macros**

**Contents**

- IDMSINTL (see page 184)
  - IDMSINTL Syntax (see page 186)
  - IDMSINTL Parameters (see page 186)
  - Create an IDMSINTL Interface Program (see page 189)
- IDMSCINL (see page 190)
  - IDMSCINL Syntax (see page 190)
  - IDMSCINL Parameters (see page 190)
  - Example Assembling and link editing IDMSCINL (see page 191)

The functionality of the CICS IDMSINTL interface has been stabilized at the r16 level except that an optional OPTIXIT exit point was added. For the applications that use this interface, we recommend using the IDMSINTC interface instead. You do not have to change or relink the application programs that were using IDMSINTL to use IDMSINTC. You can create an IDMSINTC interface module by compiling a CICSOPT with the same invocation parameters as the IDMSINTL interface being replaced.

The IDMSINTL/IDMSCINL interface differs from the standard CICS interface (IDMSINTC/IDMSCINT) in the following ways:

- Supports only native DML run-units using 16-character-format subschema controls. SQL, UCF, and Transparency interfaces remain available only through the standard interface.

- An OPTIXIT is provided, but it has more limited capabilities than the OPTIXIT provided with for the IDMSINTC interface.

- Each run-unit utilizes an ERE (External Request Element). The standard interface uses one ERE per CICS transaction regardless of the number of run-units bound by that transaction.

- Task-level commands (COMMIT TASK, ROLLBACK TASK, FINISH TASK) are not supported.

- CA IDMS load libraries are not required at runtime.
Because this interface offers a minimal subset of the features available through the standard interface, its resource requirements are therefore substantially reduced. Users with applications that demand no more than what this interface offers can use it in place of the standard interface to reduce resource consumption in the CICS system. Applications that need the full services of the standard interface can coexist freely in the same or different CICS systems.

**IDMSINTL/IDMSCINL considerations**

The following considerations apply to using IDMSINTL/IDMSCINL:

- Multiple copies of IDMSINTL and IDMSCINL can coexist within a single CICS region, but each must be associated with a unique CWADISP.

- Existing applications linked with a pre-12.0 version of IDMSCINT be executed with a Release 12.01 or higher version of IDMSINTC or IDMSINTL without relinking.

- Programs that share a single Subschema Control (SSC) must be linked with the same IDMSCINT or IDMSCINL stub module, that is, you cannot mix invocations of IDMSINTC with IDMSINTL using the same SSC.

- Programs linked with a pre-12.0 version of IDMSCINT or any other version of IDMSCINL can be executed using an IDMSINTC or IDMSINTL interface.

**IDMSINTL**

IDMSINTL is intended for applications that execute navigational DML (nDML) only. Applications that require the additional functionality such as SQL or mixed DML and SQL, must use IDMSINTC.

IDMSINTL contains an embedded TRUE exit that will abort (rollback) any un-FINISHed run-units at the end of the transaction, but does not provide AUTOCMT (CICS SYNCPOINT interception) as does the standard interface.

**Functions IDMSINTL performs at CICS startup**

The functions performed by IDMSINTL vary based on whether a CA IDMS SVC is specified in IDMSINTL as follows:

- If a CA IDMS SVC is specified, IDMSINTL performs the following functions when CICS is started up:
  1. Stores the address of the IDMSINTL entry point address table in the CWA
  2. Returns control to CICS

- If a CA IDMS SVC is not specified, IDMSINTL performs the following functions when CICS is started up:
  1. Opens and reads the SYSCTL file to obtain the necessary central version information (for example, the CVNUMBER value of the DC/UCF system and the CA IDMS SVC number)
  2. Stores the address of the IDMSINTL entry point address table in the CWA
  3. Returns control to CICS
Automatically starting IDMSINTL

It is useful to have the IDMSINTL interface module started automatically at CICS startup time. To do this, perform the following steps:

1. Code the following entry for IDMSINTL in the program list table:
   
   ```
   DFHPLT TYPE=ENTRY
   PROGRAM=IDMSINTL
   ```

2. Enter the name of the program list table (PLTPI) in the system initialization table.

If you don't have IDMSINTL started as a function of the CICS startup routine, be sure to execute the IDMSINTL interface module before attempting communication with DC/UCF at runtime. To execute the IDMSINTL module, enter a TRANSID that corresponds to the IDMSINTL module in the PCT tables.

To start up IDMSINTL automatically after control is given to CICS, perform the following steps.

1. Code the following IDMSINTL macro:
   
   ```
   PLT=NO,
   TRANSID=task-code
   ```

2. Perform steps 1 and 2 above and define `task-code` in the program control table (PCT) to invoke IDMSINTL.

   **Note:** For more information on the PCT table, see UCF Operations.

IDMSINTL processing at runtime

IDMSINTL must be resident in CICS and must be executed before the execution of any CA IDMS database application for that run of CICS. Control passes from IDMSCINL to the main entry point to notify DC/UCF to perform a service requested by the CICS user program.

IDMSINTL performs the following functions:

- Allocates dynamic storage required to service the run-unit.
- Enables the recovery exit entry point for the current CICS task. The exit performs the necessary cleanup when the CICS task terminates.
- Sends the request through the CA IDMS SVC to the DC/UCF region. IDMSINTL implicitly issues a CICS WAIT command, which places the current task in a wait state until the request is serviced.
- Passes the requested database record and/or error status to the user program.

   **Note:** For more information on runtime processing, see What Happens when a CA IDMS Instruction is Executed.
IDMSINTL Syntax

```
module-name
IDMSINTL CWADISP=cwa-intc-address-displacement
.OPSYS=operating-system
.CVNUM=cv-number
.SVC=svc-number
.SYSCTL=ddname
.TPNAME=system-name
.NODENAM=(nodename, ALWAYS, DEFAULT)
.DBNAME=(db-name, ALWAYS, DEFAULT)
.XA=YES, NO
.ERRDCT=destination-name
.PLT=YES, NO
.TRANSID=task-code
.MACLVL=YES, NO
```

IDMSINTL Parameters

- **module-name**
  Identifies the CSECT name of the generated module.

- **CWADISP=cwa-intc-address-displacement**
  Identifies the displacement within the CICS CWA of a fullword to hold the address of the IDMSINTL module. For `cwa-intc-address-displacement`, specify a number of bytes (maximum value is 3584) or the name of a field within the CSA copy book.
  **Considerations:** The specified field must be on a fullword boundary within the CWA and must be the same value given to the CWADISP parameter of the IDMSCINL macro.

- **OPSYS=operating-system**
  Identifies the operating system under which the DC/UCF system will run.
  Valid values for `operating-system`:
  - OS390
  - z/VSE
- DVS
- DS
- DOS
- DOSVS

- **CVNUM=**`cv-number`
  Identifies the number of the DC/UCF system to be accessed from CICS. For `cv-number`, specify the number used for the CVNUM parameter in the sysgen.

- **SVC=**`svc-number`
  Identifies the number of the CA IDMS SVC. For `svc-number`, specify a value as follows:
  - If no SVC is being used, or if using SYSCTL, specify NO.
  - If an SVC is being used by the DC/UCF system, specify the SVC number.

  The SVC parameter is required if no SYSCTL file is specified.

- **SYSCTL=**`ddname`
  Identifies the `ddname` of the file containing DC/UCF system control information. If no SVC (described above) is specified, the SYSCTL parameter is required. Likewise, if SYSCTL is desired, the SVC parameter must be NO (SVC=NO).

- **TPNAME=**`system-name`
  Specifies the name by which DC/UCF will identify all tasks running under this CICS system. If this parameter is omitted, the four-character local system id of this CICS system will be used, thereby permitting this CICS interface module to be used by multiple CICS systems. You may optionally specify a four-character name.
  This name forms the first part of the local transaction ID for database requests. It also forms the first 4 characters of the front end system ID for external request units. "BULK" is appended to `system-name` to form the front-end system ID. The front-end system ID is used in determining the packet size for communications and may also be used as an alternate task code for controlling external request unit processing.

- **NODENAM=**`nodename`
  Identifies a system defined to the DC/UCF communications network to be contained in the IDMSINTL module and the conditions under which programs signing on to the DC/UCF system will be directed to the named node for execution.
  For `nodename`, specify the 1- to 8-character name of a remote system. If the node name is not specified, the DC/UCF obtains the appropriate node name from the application program or from the SYSCTL file (z/OS only).

- **ALWAYS**
  Indicates that `nodename` is to override any node named by the program. Requests from programs signing on to DC/UCF are always directed to the named node regardless of node name specifications made by the program.

- **DEFAULT**
  Indicates that requests from programs signing on to DC/UCF are to be directed to the named node only if the program does not name a node.
Note: Under z/OS and z/VSE, SYSCTL database name specifications can override IDMSINTL and program specifications.

- **DBNAME=** `db-name`
  Identifies the database (or data dictionary) name to be contained in the IDMSINTL module. This parameter also identifies the conditions under which programs signing on to the DC/UCF system access the named database.
  For `db-name`, specify the name of the database that programs are to access when running under the DC/UCF system. If the database name is not specified, DC/UCF obtains the appropriate database name from the application program or from the SYSCTL file (z/OS only).

- **ALWAYS**
  Indicates that `db-name` is to override any database named by the program. Programs signing on to DC/UCF always execute against the named database regardless of database name specifications made by the program.

- **DEFAULT**
  Indicates that programs signing on to DC/UCF are to execute against the named database only if the program does not name a database.

Note: Under z/OS and z/VSE, SYSCTL database name specifications can override IDMSINTL and program specifications.

- **XA=** `NO|YES`
  Designates whether the operating system is XA (YES) or not (NO).

- **ERRDCT=** `destination-name`
  Identifies the CICS transient data destination to be used as the target for error messages produced by IDMSINTL. The default `destination-name` is CSMT. Use another destination if you want to rout CA IDMS error messages to another CICS destination. The DCT entry should be defined with a logical record length of at least 130 characters.

- **PLT=** `YES|NO`
  Indicates how IDMSINTL starts up. YES indicates that IDMSINTL can start up as a PLT-invoked program. NO indicates IDMSINTL always starts up as a user task once CICS start up is complete.

- **TRANSID=** `task-code`
  Identifies the transaction coded in the program control table (PCT) as invoking IDMSINTL. `task-code` must be the name of a task defined in the PCT table.

Note: For more information on the PCT table, see UCF Operations.

- **MACLVL=** `YES|NO`
  This parameter is obsolete. Macro level programs are no longer supported in CICS.
Create an IDMSINTL Interface Program

This section contains the JCL to assemble and link edit the IDMSINTL module for z/OS and z/VSE operating systems.

z/OS IDMSINTL assembly and link edit

To assemble and link edit IDMSINTL

1. Create an IDMSINTL source module as follows:

   IDMSINTL idmsintl-parameters
   END

2. Save the IDMSINTL source module in your custom source library.

3. Assemble and link it into your custom load library by executing the z/OS Assemble and Link-Edit JCL (see page 465).

   Substitute the name of your IDMSINTL source member and insert the following binder statements:

   ORDER DFHEAI
   INCLUDE CUSTLIB(optixit) (Optional)
   INCLUDE CAGJLOAD(IDMSLSTB)
   ENTRY STARTUP
   SETOPT PARM(REUS=NONE,AMODE=31,RMODE=24)
   NAME idmsintl(R)

   - **optixit**
     Specifies the name of your OPTIXIT exit routine.

   - **idmsintl**
     Specifies the name of your IDMSINTL interface load module.

z/VSE IDMSINTL assembly and link edit

To create an IDMSINTL interface program under z/VSE:

1. Assemble and catalog an IDMSINTLoptions table using the sample JCL in z/VSE Assemble JCL (see page 476).

   Modify the JCL by substituting the following in place of the Assembler input statements:

   PUNCH 'CATALOG idmsintl.OBJ REPLACE=YES'
   IDMSINTL idmsintl-parameters
   END

2. Link the IDMSINTL interface program using the sample JCL in z/VSE Link JCL.

3. Modify the JCL by substituting the following statements in place of the Linkage editor control statements:

   PHASE idmsintl,*
   INCLUDE DFHEAI
INCLUDE optiexit (Optional)
INCLUDE IDMSLST6
INCLUDE DFHEAI0
MODE AMODE(31),RMODE(24)
ENTRY STARTUP

- **idmsint**
  Specifies the name you choose for your IDMSINTL options table.

  Note: You can have more than one options table with different names

- **optixit**
  Specifies the name of your OPTIXIT exit routine.

**IDMSCINL**

IDMSCINL is essentially a copy of the Release 10.2 IDMSCINT with the exception that the CICS CWA is located using ADDRESS CWA. This makes IDMSCINL compatible with all releases of CICS.

You generate IDMSCINL at DC/UCF installation time. IDMSCINL must be link edited with each CICS user program that accesses DC/UCF (including Assembler modules). IDMSCINL retrieves the entry point address of IDMSINTL from the CWA and passes control to IDMSINTL. The module generated by the IDMSCINL macro is fully reentrant.

Note: For more information on runtime events, see What Happens when a CA IDMS Instruction is Executed.

To prepare an IDMSCINL macro, use the following syntax.

**IDMSCINL Syntax**

```
IDMSCINL CWADISP=cwa-intc-address-displacement

,EP1=first-entry-point-label

,EP2=second-entry-point-label

,EXEC=YES
```

**IDMSCINL Parameters**

- **IDMSCINT**
  Identifies the CSECT name of the generated module. This tag is optional.
• **CWADISP**=*	extit{cwa-intc-address-displacement}*
  This identifies the displacement within the CICS CWA of a fullword that holds the address of the IDMSINL module. For *	extit{cwa-intc-address-displacement}*, specify the same value given to the CWADISP operand of the IDMSINL macro.

• **EP1**=*	extit{first-entry-point-label}*
  Identifies the name of the first entry point in the generated module. For *	extit{first-entry-point-label}*, specify the name of the first entry point. The default value is either the value given to *module-name* above or IDMSCINT if *module-name* is not specified.

• **EP2**=*	extit{second-entry-point-label}*
  This parameter is obsolete and should no longer be used.

• **EXEC**=*YES|NO*
  Indicates whether the IDMSCINT module being generated is to be used with the CICS command-level interface.

  • **YES**
    Specify YES to indicate that the command-level interface is in use. YES is the default.

  • **NO**
    Specify NO to indicate that the command-level interface is not in use.

  \begin{quote}
    \textbf{Note:} This option should not be specified because macro level programs are no longer supported in CICS.
  \end{quote}

**Example Assembling and link editing IDMSCINL**

This section contains the JCL to assemble and link edit the IDMSCINL module for z/OS and z/VSE operating systems.

**z/OS IDMSCINL assembly and link edit**

**To assemble and link edit IDMSCINL**

1. Create a source module as follows:

   ```
   IDMSCINL idmscinl-parameters
   END
   ```

2. Save the source module in your custom source library.

3. Assemble and link it into your custom load library by executing the z/OS Assemble and Link-Edit JCL (https://docops.ca.com/display/IDMS19/CICS+Considerations).
   Substitute the name of your source member and insert the following binder statement:

   ```
   NAME idmscinl(R)
   ```
4. Add the following statement to the link of each application program that accesses IDMS using this `idmscinl` application stub program. The CWADISP specified in the IDMSCINL parameters must match the CWADISP specified in the IDMSINTL parameters used in creating an IDMSINTL interface program:

```
INCLUDE idmscinl
```

- `idmscinl`
  Specifies the name of your IDMSCINL interface module.

**z/VSE IDMSCINL assembly and link edit**

1. Assemble and catalog an IDMSCINL program using the sample JCL in [z/VSE Assemble JCL](page 476).
   Modify the JCL by substituting the following in place of the `Assembler input statements`:

   ```
   PUNCH 'CATALOG idmscinl.OBJ REPLACE=YES'
   IDMSCINL idmscinl-parameters
   END
   ```

2. Add the following statement to the link of each application program that accesses IDMS using this `idmscinl` application stub program. The CWADISP specified in the IDMSCINL parameters must match the CWADISP specified in the IDMSINTL parameters used in creating an IDMSINTL interface program:

```
INCLUDE idmscinl
```

- `idmscinl`
  Specifies the name you choose for your IDMSCINL application stub program.

**Note:** You can have more than one application stub program with different names.

---

**DC/UCF Execution Mode in the CICS Environment**

The DC/UCF region runs independently of the CICS monitor, but CICS programs use CA IDMS central version database management services.

The following diagram shows the CICS monitor and the DC/UCF system:
To operate CICS with DC/UCF

1. Start up the DC/UCF system.

2. Start up CICS following standard CICS operating procedures.

Dispatch priorities

You should assign DC/UCF a higher dispatching priority than CICS. If CICS has a higher priority, you run the risk, in a busy system, of starting many more CICS tasks and of binding the DC/UCF transactions without giving DC/UCF an opportunity to finish what was started. By having DC/UCF at a higher dispatching priority, work is more likely to be completed in an efficient first in - first out (FIFO) method. In addition, there is less likelihood of exceeding the MAXERUS task parameter limit and less likelihood of exceeding the MAXTASKS parameter limit on CICS when all the CICS tasks are waiting on CA IDMS requests to finish.

Running Multiple CICS or DC/UCF Systems

Contents

- CICS System Name (see page 193)
- Accessing Multiple CA IDMS Systems from a CICS Region (see page 194)
- Accessing One CA IDMS System from Multiple CICS Regions (see page 195)

Many configuration options are available when CA IDMS is integrated with CICS as follows:

- A single front-end CICS region can access multiple CA IDMS
- One CA IDMS system can be accessed by multiple CICS regions.

CICS System Name

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

The name of the CICS system is established as follows:

- The value in the CICS_NAME parameter specified in the SYSIDMS file included in the CICS startup JCL.
- If the CICS_NAME parameter is not specified, the value of the TPNAME parameter associated with the first IDMSINTC interface started within a CICS system.
If the CICS name is allowed to default to the TPNAME of the first CICS interface, all other IDMSINTC interface modules started within the CICS system must have the same TPNAME value or they will fail with a K213 abend code.

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

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

\[
\text{CICS\_NAME}=\text{cics-name}
\]

- **cics-name**
  - Specifies a 1-4 character value that identifies the CICS system being started. It should be unique across all CICS systems that may access the same central version.

### Accessing Multiple CA IDMS Systems from a CICS Region

You can use the IDMSINTC interface program to access multiple CA IDMS back-end systems in the following ways:

- **Traditional Method**
  - This method uses a separate CICSOPT macro for each back-end. Each CICSOPT macro specifies a different CWADISP and a unique SYSCTL value or a unique SVC/CVNUM combination. Each object module is linked with the IDMSINTC load module (created during installation) as described in IDMSINTC to create a uniquely named load module. Each interface program is invoked separately, creating multiple active interfaces. Each application program or UCFCICS interface program is linked with an IDMSCINT module as described in IDMSCINT. The CWADISP specified in the linked IDMSCINT module determines which IDMSINTC interface is used, and therefore, the CA IDMS back-end is accessed.

  **Note:** It is recommended that the same TPNAME value be specified or defaulted for all IDMSINTC interfaces used in a single CICS region. If different TPNAME values are used in two IDMSINTC interfaces in the same region, then the CICS startup JCL must contain a SYSIDMS parameter that specifies a **CICS\_NAME** value.

- **Alternative Method**
  - The method uses a single IDMSINTC interface in conjunction with the OPTIXIT or OPTIQXIT to access up to ten CA IDMS back-end systems. All application programs are linked with an IDMSCINT module that specifies the same CWADISP as the single IDMSINTC interface. The client-coded OPTIXIT or OPTIQXIT can use a variety of strategies to select the appropriate back-end. These selection strategies include the following:
    - Round-robin -- Selection is random. This is appropriate for a distributed processing application where multiple back-ends access the same data.
    - Database name -- Selection is based upon the database name specified in the BIND RUN UNIT
Application or CICS transaction -- Selection is based on the application program name or CICS transaction.

**Note:** For more information on these exits, see OPTIXIT and OPTIQXIT.

### Accessing One CA IDMS System from Multiple CICS Regions

When multiple CICS regions access one CA IDMS back-end system, the TPNAME value must be unique for each front-end. Traditionally, this meant a different IDMSINTC load module was needed for each region. Now, the TPNAME parameter can be omitted or specified as null on the CICSOPT macro invocation. In this case, the TPNAME is automatically set to the CICS SYSIDENT value of the CICS region in which the interface program is executed. If the CICS systems programmer assures that the SYSIDENT value is unique for each CICS system, one IDMSINTC interface module can be used to access a given CA IDMS back-end from multiple CICS regions.

### Components of the CICS and DC/UCF Environment

When used with DC/UCF, the CICS operating environment contains the components listed below and shown in the following diagram.

When CICS is used with DC/UCF, the CICS operating environment contains the following:

- CICS application programs
- CICS program interface module (IDMSCINT). IDMSCINT provides the necessary communication between the CICS user program and the CICS TP-monitor interface module IDMSINTC (described below).
- CICS nucleus
- CICS common system area (CSA). The CSA includes a common work area (CWA) that contains a fullword pointing to a table of addresses within IDMSINTC; the addresses in the table point to three entry points within IDMSINTC (described below).
- CICS TP-monitor interface module (IDMSINTC). IDMSINTC allows CICS and DC/UCF to communicate at runtime.
- IDMSTRUE, the task termination exit
Components of the CICS and DC/UCF Environment

CICS Extended Addressing Considerations

The CA IDMS interface to CICS supports user programs running above the 16 megabyte line. In order to take advantage of XA enhancements, the customer must be using an XA-compliant compiler.

In addition, the following steps are necessary to implement XA support for CICS programs invoking CA IDMS services:

1. Assemble IDMSINTC and IDMSCINT using the H Assembler program ASMA90.
2. Linkedit IDMSINTC as AMODE(31), RMODE(24).
3. Linkedit IDMSCINT as AMODE(31), RMODE(ANY).
4. Linkedit user programs as AMODE(31), RMODE(ANY).

The compiler option for dynamic areas in the user program may specify DATA(31).

⚠️ **Note:** Program UCFCICS (UCF CICS front-end) must be AMODE(31), RMODE(24), and can only run below the line.
What Happens when a CA IDMS Instruction is Executed

The following diagram shows what happens when a CICS program executes a CA IDMS database instruction when using the command-level interface.

1) Supervisor

- Application program
- IDMSCINT
- EP1
- CICS nucleus
  - CSA
  - CWA
- EP1
- IDMSINTC
- IDMS interface
- DC/UCF

The application program calls entry-point-1 (EP1) in IDMSCINT.

2) Supervisor

- Application program
- IDMSCINT
- EP1
- CICS nucleus
  - CSA
  - CWA
- EP1
- IDMSINTC
- IDMS interface
- DC/UCF

IDMSCINT finds in the CWA the address of the IDMSINTC entry-point-address table, and uses that address to access EP1 within IDMSINTC.
When DC/UCF is used with CICS, the IDMS communications block provides a 16-character field that CICS uses as a work area. This field immediately follows the direct db-key (DIRDBKEY) field.

Note: For information on using the IDMS communications block, see the DML Reference section for Assembler.

When the standard CICS interface (IDMSINTC) enters a CICS wait, the type of wait is identified, which you can use to diagnose unexplained processing delays.

The IDMSINTC interface specifies a NAME parameter when it issues an EXEC CICS WAIT. The value of this parameter identifies the type of wait being done. The values for the NAME parameter are:

- **IDMSDL1T**
  Task is waiting for CA IDMS DLI Transparency processing.

- **IDMSSYNC**
  Task is waiting for two-phase commit synchronization to complete.

- **IDMS EXT**
  Task is waiting for completion of a UCF or DML request to a CA IDMS back end.

Note: You can display the type of wait being done using CICS facilities such as CEMT INQ TASK.
Using the Various CICS Interfaces

The rules for choosing the CICS interface for application execution are as follows:

- Any application linked with a pre-12.0 version of IDMSCINT may only issue navigational DML requests and can be executed with any of the CICS interfaces.

- Any application linked with IDMSCINL may only issue navigational DML requests and can be executed with any of the CICS interfaces.

- Any application linked with the 12.0 or higher version of IDMSCINT may issue any supported CA IDMS request and can be executed with IDMSINTC.

CICS Storage Protection with IDMSINTC and IDMSINTL

Contents

- CICS Setup Procedures (see page 199)
- DC/UCF Setup Procedures (see page 200)

Beginning with Version 3.3 of CICS, IBM introduced storage protection. To use the features for RENTPGM and STGPROT with programs that access CA IDMS using the IDMSINTC or IDMSINTL interfaces, certain procedures must be followed.

⚠️ Note: TRANSACTION ISOLATION is NOT supported for CICS transactions that access CA IDMS using the IDMSINTL interface. It is supported for transactions that use the IDMSINTC interface if the appropriate procedures are followed.

CICS Setup Procedures

1. Ensure that IDMSINTC, IDMSINTL, and UCFCICS are linked with NORENT parameter.

2. The transactions that invoke the IDMSINTC and UCFCICS interfaces need to be defined with TASKDATAKEY=CICS.

3. The IDMSINTC and UCFCICS interface programs need to be defined with EXECKEY=CICS. Application programs can be defined with EXECKEY=USER.

4. IDMSINTL needs to be defined with EXECKEY=CICS on z/OS systems and with EXECKEY=USER on z/VSE systems.

5. The task that invokes IDMSINTL needs to be defined with TASKDATAKEY=CICS on z/OS systems and with TASKDATAKEY=USER on z/VSE systems.
DC/UCF Setup Procedures

The following procedures are required for z/OS operating systems. They are not required on z/VSE operating systems because the primary key for each partition is unique.

1. Set up a PPT entry for the CV startup module (RHDCOMVS) to specify any KEY other than the CICS protect key (usually 8). The key for the CV startup module should normally be 4.
   For more information on specifying a CV startup key, see Storage Key Considerations for z/OS CSA subpools (see page 449).

2. Set the CV SYSGEN SYSTEM statement STORAGE KEY IS parameter to any KEY other than the KEY specified for item 1.

3. Link the CV STARTUP module in an APF authorized library. The CV STARTUP JCL should contain only this APF AUTH library in the STEPLIB. Other loadlibs should be concatenated under CDMSLIB. Note that the STARTUP module must reside in an authorized library, however, the load module may or may not be authorized (that is, linked with AC(1)).
   For more information, see Creating a Secured CA IDMS System on z/OS in the section "Getting Started" in the Installing section -- z/OS.

CICS Threadsafe Support

Contents

- Threadsafe Concepts (see page 200)
- CA IDMS Support for Threadsafe Applications (see page 201)
- IDMSINTC Interface Considerations (see page 202)
  - CA IDMS Applications Which Must Not Be Declared Threadsafe (see page 203)
  - UCF Front-end (UCFICICS) Considerations (see page 203)
  - Distributed Processing with UDASCIC Considerations (see page 204)
  - CICS Abort Session Program Considerations (see page 204)
  - IDMSRSYN Resynchronization Program Considerations (see page 204)

CA IDMS is enhanced with CICS threadsafe support that allows threadsafe application programs to use multiple open TCBs while accessing CA IDMS.

Threadsafe Concepts

CICS Transaction Server for z/OS (CTS) provides a method for multiple CTS transactions to run simultaneously on separate TCBs. Application programs that are eligible to run in this mode are described as threadsafe. For more information on the CTS threadsafe operation, see the appropriate IBM documentation. A brief overview of this IBM feature as it relates to the CA IDMS interface is described in the next section.
Historically, all CICS application programs ran on the same TCB, which allowed only one program task to execute at any given instant. While multiple tasks could be active, only one task could execute instructions on a CPU. Under CTS, IBM has introduced the concept of threadsafe application programs that can be run on open TCBs, thus allowing multiple programs to execute simultaneously on different CPUs.

A program that is declared with the CONCURRENCY(THREADSAFE) attribute is considered to be eligible to run on an open TCB, but this attribute alone is not enough to cause the program to do so. Various conditions exist that cause a threadsafe program to execute on an open TCB. Three of the most common cases are the following:

- Define an application program with the API(OPENAPI) attribute. This attribute is only available in CTS V3.1 and later.
- Invoke a Task Related User Exit (TRUE) that has been enabled with the API(OPENAPI) attribute.
- Access a DB2 database using DB2 Version 6 or later. This is a special instance of the previous case because DB2 executes as a TRUE exit.

When a task begins to run on an open TCB, it continues to run until one of the following occurs:

- A non-threadsafe command is executed
- An EXEC CICS RETURN is made to a non-threadsafe program
- A particular point is reached during CICS task termination processing

A threadsafe command is one which can be executed on an open TCB. A program defined as threadsafe can issue a non-threadsafe command. However, issuing a non-threadsafe command causes the task to be switched to run on the QR (single-threaded) TCB. This can cause performance degradation, particularly if a lot of TCB switching is done.

If a threadsafe program defined with API(CICSAPI) is switched to the QR TCB, it stays there unless another OPENAPI TRUE exit is invoked. If a threadsafe program defined with API(OPENAPI) is switched to the QR TCB, it switches back to the open TCB when control is returned to the application program after execution of the non-threadsafe command.

**CA IDMS Support for Threadsafe Applications**

The CA IDMS interface modules that run in a CTS region have been enhanced to be threadsafe. Threadsafe application programs, that is, programs defined with the CONCURRENCY(THREADSAFE) attribute can use this enhancement to obtain increased throughput.

An application program that is running on an open TCB can access CA IDMS without switching to the single-threaded QR TCB. A new option, TRUEAPI, is provided to allow the first CA IDMS access by a task to force a switch to an open TCB. If the interface has been called by a program defined as threadsafe, the program continues to run on the open TCB after return from the CA IDMS call. For more information on the TRUEAPI option, see IDMSINTC.
CTS has rules and guidelines on whether an application program can or should be defined with CONCURRENCY(THREADSAFE) or API(OPENAPI) attributes or both. Before defining your own programs as THREADSAFE or OPENAPI, be sure to consult the appropriate IBM documentation.

The use of CA IDMS with an otherwise threadsafe program does not cause integrity problems and can provide significant performance improvement. Depending on the nature of the application, however, it may not improve performance and could conceivably cause performance degradation. In addition, if a client-written application program is declared to be threadsafe and the program itself violates the rules for threadsafe programs, the results are unpredictable.

⚠️ **Note:** A few cases exist where the CA IDMS interface issues CICS commands that force a switch to the QR TCB. For more information, see IDMSINTC Interface Considerations.

### IDMSINTC Interface Considerations

You can enter the IDMSINTC interface program using one of the following methods:

- Through the PLT or by invocation of a transaction that starts the IDMSINTC interface. By invoking IDMSINTC in this way, it is not threadsafe, so the program cannot be defined as THREADSAFE. See the sample definition of PROGRAM(IDMSINTC) in member CICSCSD in the installed CA IDMS source library.

- Through a branch entry from the application program using the IDMSCINT stub program. This entry functions as an extension of the calling program with the same program attributes. This includes the THREADSAFE and OPENAPI attributes.

Except for a few cases, discussed in Non-threadsafe Instructions, the IDMSINTC interface does not issue any non-threadsafe commands. Therefore, if IDMSINTC is entered on an open TCB, it stays on the open TCB throughout its execution and return to the application program.

When a task makes its first CA IDMS call, IDMSINTC invokes the CA IDMS TRUE exit. This exit is always enabled with the THREADSAFE attribute. The TRUEAPI=OPEN parameter is provided on the CICSOPT macro that causes the exit to also be enabled with the OPENAPI attribute. For more information on the TRUEAPI parameter, see CICSOPT Syntax.

If TRUEAPI=OPEN is specified, the first CA IDMS call in each task causes a switch to an open TCB. If the application program is defined as threadsafe, the interface continues to execute on that open TCB through its return to the application program.

### Non-threadsafe Instructions

A few cases exist where invocation of the CA IDMS interface from an application program causes a non-threadsafe instruction to be issued. If the application program is defined as THREADSAFE, but not OPENAPI, the interface continues to execute on the QR TCB through return to the application program.
program. If the application program is defined as OPENAPI, CICS switches to the QR TCB during execution of the non-threadsafe instruction and back to the open TCB after completion of the instruction. The interface continues to run on the open TCB through return to the application program.

The following cases can cause a non-threadsafe instruction to be issued:

- If the CICSOPT macro specifies a value other than IMMEDIATE on the TIMEOUT parameter, an EXEC CICS START TRANSACTION is issued at task termination. This is not an important performance consideration because there will be no return to an application program, and CICS always switches to the QR TCB at some point during task termination.

- If a ROLLBACK command is issued when AUTOCMT=ON is in effect, an EXEC CICS SYNCPOINT ROLLBACK is issued.

- Certain error conditions can cause a message to be written using an EXEC CICS WRITEQ TD command. These conditions are rare in a production system.

- If DEBUG=ON is specified in the CICSOPT macro or IDMSDBG=ON is specified as a SYSIDMS runtime parameter, various information is written using the EXEC CICS WRITEQ TD command. These options are usually used only in special situations when Technical Support personnel need diagnostic information to resolve a problem. For more information on using an alternative parameter value of DEBUG=QTS, see CICSOPT syntax.

CA IDMS Applications Which Must Not Be Declared Threadsafe

Some CICS application programs that access CA IDMS are not threadsafe. This can be the case even if these programs do not violate any of the restrictions of CICS threadsafe processing. Unpredictable results can occur if such a program is defined as THREADSAFE to CICS.

The limitations are as follows:

- Application programs that access CA IDMS via an IDMSINTL interface are not threadsafe.

- An early version of the release 12.0 IDMSCINT macro did not have the SSC12 parameter. An IDMSCINT application stub program compiled with this version of the macro can be entered from an application program that uses either a R10.2-format Subschema Control (with 16-character names) or a R12.0-format Subschema Control (with 18-character names). Any such application program is not threadsafe.

Later versions of the IDMSCINT macro that do contain the SSC12 parameter do not cause a program to become non-threadsafe. If a program is compiled with SUBSCHEMA-NAMES LENGTH IS 18, it must be linked with an IDMSCINT module that was assembled specifying SSC12=YES. If an application program is otherwise threadsafe, but it has been linked with an early R12.0 IDMSCINT module, the program can be declared threadsafe if it is relinked replacing the old IDMSCINT module with one created with a new IDMSCINT macro.

UCF Front-end (UCFCICS) Considerations

The UCF Front-end program does not violate any threadsafe rules and can be declared as a THREADSAFE or OPENAPI program. It does, however, issue various non-threadsafe commands, such as terminal I/O commands. Therefore, for best performance, it should be defined with CONCURRENCY (QUASIRENTRANT).
Distributed Processing with UDASCIC Considerations

The distributed processing program created with the #UDASCIC macro rarely issues a non-threadsafe command and is a good candidate to declare as a THREADSAFE or OPENAPI program. However, this program does issue a non-threadsafe command when a 1473 Error-Status is received from the CA IDMS interface because of a MAXERUS condition on the CA IDMS central version. In this case, the program waits by continuing to issue EXEC CICS DELAY INTERVAL(1) commands until the condition is alleviated or 100 attempts have been made.

CICS Abort Session Program Considerations

The CICS Abort Session Program is created by compiling the #UCFCICZ program and can be declared as a THREADSAFE and OPENAPI program.

The #UCFCICZ macro generates some code that is not compliant with the recommended usage with CICS Transaction Server. This code can cause problems with applications that are associated with a bridge facility. To provide compatibility with previous methods of calling #UCFCICZ from CICS error programs, two new parameters are added to the #UCFCICZ macro: PASSVAL and BRIDGE. To prevent the #UCFCICZ macro from issuing non-threadsafe commands, we recommend that you compile it with the PASSVAL=TERMID parameter.

Note: For more information, see CICS Abort Session Program.

IDMSRSYN Resynchronization Program Considerations

IDMSRSYN is threadsafe, but because it issues non-threadsafe commands, it should not be defined as OPENAPI.

Implementing CA IDMS DMLO in Multiple CVs Under CICS

This section describes how to install CA IDMS DMLO on multiple CVs under CICS.

To install CA IDMS DMLO on multiple CVs

1. Create additional copies of the DMLO CICS front end load module by linking USDTPIFS with the version of IDMSCINT that interfaces with the desired CA IDMS CV. Name each DMLO front end with a unique suffix, for example, USDTPIFA.
2. All CICS definitions for DMLO are provided in install member CAGJSAMP (USDCICS). USDCICS should be input to the CICS System Definition utility, DFHCSDUP. If multiple USDTPIFx modules are created for access to multiple CVs from a single CICS, definitions for program USDTPIF5 and transaction DMLO can be used as samples.

3. Add a unique transaction code to the PCT for each DMLO CICS front end module. The first three characters of the transaction codes must be unique to CA IDMS DMLO. The fourth character of each transaction code must be unique to for each front-end module. Distribution source library member USDPCT can be modified for this purpose.

Note: CA IDMS DMLO also delivers modules USDTPIF2 and USDTPIF3. Module names USDTPIF2 and USDTPIF3 cannot be used.

IDMS User Exits

Predefined user exits allow sites to call user-written routines (user exit routines) at predefined times during DC/UCF system execution. User exits allow for a variety of site-specific processing, including security checks, automatic display of system news at signon, and additional statistics collection.

For example, you can write a user exit routine that's executed immediately before system statistics are written to the log. The user exit routine can read current statistics records, reformat and add to selected records, and send an additional message to the log recording information of specific interest at your site.

Types of user exits

This section describes the following types of user exits that are available:

- DB exits allow users to receive control during database operations.
- DC/UCF exits allow users to receive control during system operations.
- CICS exits allow users to receive control during the processing of requests for CA IDMS data from a CICS Transaction Server.
- Tools exits allow CA IDMS Tools to perform special processing.
- Numbered exits allow users to receive control during system operations and during execution of user-written programs.

Costs associated with user exits

Individual sites can use user exits to obtain information on the internal operations of CA IDMS, and, to some degree, customize CA IDMS. Each site must evaluate whether the benefits associated with user exits outweigh the costs; the costs include:

- Implementation and testing
Performance

Recompilation and retrofit in the event that CA IDMS managed control blocks change; this cost directly affects how quickly a site can upgrade to a new release.

Control blocks

To write user exit routines, you must be familiar with both the layout and runtime usage of control blocks. When a user exit is called, the user exit routine can access information from system control blocks. During execution, a user exit routine can return information to control blocks, for later use by other programs. The portion of a user exit routine that accesses control blocks and registers must be coded in Assembler.

More Information

- For more information on layouts and descriptions of system control blocks, see the Reference section and the CA IDMS DSECT Reference section.
- For more information on using Assembler in the DC/UCF environment, see the CA IDMS DML Reference section for Assembler.

Named User Exit Enablement

To facilitate applying system maintenance, user-written exits are linked separately from the code that invokes the exit.

All named user-written exits are linked with IDMSUXIT, which is a list of VCONS that address the exits to be invoked. Numbered user exits, exits invoked in the CICS environment, the SVC exit, and exits that are loaded dynamically are not linked with IDMSUXIT. To enable the use of other named exits, you need to link your exit module with IDMSUXIT and place it in your custom load library, and then include this in your STEPLIB concatenation for batch jobs and in CDMSLIB for Central Version startup.

You can determine the named exits that are enabled within a DC/UCF system using the DCPROFIL system task.

Entry Point Names

Contents

- Compiler Exits and Entry Points (see page 207)
- WTOEXIT and WTOREXIT Implementation (see page 208)

The following table identifies the exits that can be linked with IDMSUXIT and the name of the entry point that must be used in each case.

<table>
<thead>
<tr>
<th>Exit</th>
<th>Entry Point Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>User ID exit for batch jobs using the 10.2 services interface</td>
<td>BTCIDXIT</td>
</tr>
</tbody>
</table>
Exit | Entry Point Name
---|---
RELOAD error exit | DBLUEREEX
IDD batch compiler exit | IDDEXITB
IDD online compiler exit | IDDEXITO
ARCHIVE JOURNAL exit | IDMSAIJNX
IDMSCALLC exit | IDMSCLCX
IDMSDBIO duplex exit | IDMSDPDX
IDMSDBIO IO statistics exit | IDMSIOXT
IDMSDBIO I/O exit | IDMSIOX2
IDMSDBIO journal exit | IDMSJNXL2
OLQ DML exit | OLQDMLX
Schema batch compiler exit* | SCHEXITB
Schema online compiler exit* | SCHEXITO
Subschema batch compiler exit* | SUBEXITB
Subschema online compiler exit* | SUBEXITO
SYSGEN batch compiler exit* | SGNEXITB
SYSGEN online compiler exit* | SGNEXITO
Ticker exit | TCKREXIT
User ID exit for batch jobs | USRIDXIT
Wait exit | WAITEXIT
Write to operator exit ** | WTOEXIT
Write to operator reply exit ** | WTOREXIT

* See the section Compiler Exits and Entry Points (see page 207)
**See the section Enabling WTOEXIT and WTOREXIT (see page 208)

**Compiler Exits and Entry Points**

Each compiler calls separate exit entry points in each of the batch and online environments. If you have separate exit modules for each compiler and each environment, you must change their entry points as indicated in the table in Entry Point Names. There are two ways you can change an entry point:

- Change the entry point name in the source code and recompiling the program.
- Use appropriate binder directives to change the entry point name when including the exit module in the link of IDMSUXIT.

We recommend changing the program source code and recompiling.
If you use the same exit module for more than one compiler or in more than one environment, you can continue to do so by changing the exit module to include entry points for each of the compilers and environments in which the module is to be used. Alternatively, you can use binder directives to change the name of one or more external references to match the name of your module’s entry point in IDMSUXIT. We recommend changing your exit program.

**WTOEXIT and WTOREXIT Implementation**

There are three ways you can implement WTO and WTOR exits:

1. By linking your exit routines with IDMSUXIT.
2. By specifying the name of the WTO and WTOR exits using DC/UCF system startup parameters.
3. In z/VSE, by linking your exit routines as phases WTOEXIT and/or WTOREXIT.

⚠️ **Note:** If the second or third technique is used, CA IDMS dynamically loads the exit modules during startup and ignores any corresponding exit routine linked with IDMSUXIT.

For more information on using an OPS/MVS API rule as a replacement for the WTOEXIT, see the Appendix Sample OPS/MVS API rule (see page 516). This can lead to improved performance when using the CA IDMS zIIP feature.

**Note:** You must link your exit modules with IDMSUXIT if you want to invoke the WTO and WTOR exits in batch jobs.

**IDMSUXIT Module Creation**

To create an IDMSUXIT module, link each exit you want to activate with the IDMSUXIT object and name the resulting module IDMSUXIT. If nothing is included, no named exits are active, which is equivalent to the installed version of the module. See IDMSUXIT in the sample JCL in the installation file.

**DB Exits**

**Contents**
- IDMSAJNX (see page 209)
- IDMSCLCX (see page 210)
- IDMSDPLX (see page 211)
- IDMSIOXT (see page 214)
- IDMSIOX2 (see page 214)
DB user exits allow the site to call site-specific database processing routines. You must associate each user exit routine with a user exit entry point as described in the previous section. These entry points are predefined by the system.

To include a DB user exit routine in the system, do the following:

1. Code the routine based on the information presented in this section.
2. Link edit the entry point with IDMSUXIT.

At runtime, the system calls a user exit routine by means of the predefined entry point for the exit. After the exit routine is performed, control returns to the calling module.

If you do not link edit a given user exit entry point with IDMSUXIT, the user exit is bypassed at runtime, except for WTOEXIT and WTOREXIT, which can be loaded dynamically. In this case, database operations continue without interruption.

DB user exits are listed in the following table and described individually after the table.

**Registers 2 through 13:** If any of these DB exits use registers 2 through 13, ensure that the exit routine saves and restores the contents of these registers.

<table>
<thead>
<tr>
<th>Entry point</th>
<th>CA IDMS module</th>
<th>Usage of exit</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDMSAJ</td>
<td>IDMSUAJNX</td>
<td>To review journal records</td>
</tr>
<tr>
<td>IDMSC</td>
<td>IDMSDBMS</td>
<td>To compute a CALC key target page LCX</td>
</tr>
<tr>
<td>IDMSD</td>
<td>IDMSDBIO</td>
<td>To maintain duplicate journal and/or database files PLX</td>
</tr>
<tr>
<td>IDMSIO</td>
<td>IDMSDBIO</td>
<td>To read I/O statistics associated with a database file XT</td>
</tr>
<tr>
<td>IDMSIO</td>
<td>IDMSDBIO</td>
<td>To capture I/O statistics, maintain duplicate database or journal files, to replace normal I/O calls with calls issued by this exit X2</td>
</tr>
<tr>
<td>IDMSJ</td>
<td>IDMSDBIO</td>
<td>To write a second journal file NL2</td>
</tr>
</tbody>
</table>

**IDMSAJNX**

**Entry point**

- **IDMSAJNX**
  
  IDMSUAJN calls this user exit immediately *after* a journal is archived (*offloaded*) from disk to tape.

**Sample uses**

IDMSAJNX can be called to review journal records to do the following:
CA IDMS - 19.0

- Collect statistics on database activities
- Collect audit trail information on database usage

**Journal record control block**

When IDMSAJNX is called, register 1 points to a fullword that contains the address of the journal record control block. This control block is described by copybook #JTRDS.

**More Information**

- For more information on journaling and journal record types, see the *Database Administering section*.
- For more information on the layout of journal records, see the *DSECT Reference section*.

### IDMSCLCX

**Entry point**

- **IDMSCLCX**
  
  IDMSDBMS and IDMSUTIL call this user exit routine each time a program requests a CALC key target page. The IDMSCLCX exit routine receives database information and the CALC key for the record. From this information, the exit routine can calculate the target-page value.

**Sample uses**

IDMSCLCX can be called to perform a user-written CALC routine as follows:

- Access database and CALC information passed to IDMSCLCX *without* calculating a target page. To do this, the IDMSCLCX exit routine:
  1. Reads information passed by control blocks
  2. Prepares the system to execute the standard CALC routine by returning binary zeros to the target-page field in the CALC key record control block
  3. Returns control to IDMSDBMS

After control returns to IDMSDBMS, the standard CA IDMS CALC routine (IDMSCALC) is invoked if the target-page field contains binary zeros.

- Compute a target CALC page. To do this, the IDMSCLCX user exit routine:
  1. Computes a target page by using:
     - Database definitions from the CALC key record control block
     - The actual CALC key value from the CALC key control block
2. Returns the computed target-page number to the system. This is accomplished by placing the page number in the target-page field of the CALC key record control block.

3. Returns control to IDMSDBMS.

The standard CA IDMS CALC routine is not invoked for the record when the target-page field returns with values other than binary zeros.

Parameters

When IDMSCLCX is called, register 1 points to a parameter list that contains the following two fullword address constants (ADCONs):

- The first address constant contains the address of the CALC-key record control block. This control block contains the information listed below.

Field | Field description
--- | ---
Target page (full word) | Suggested page number for storage of the record occurrence. Initialized by the system to binary zeroes.
High page (fullword) | The number of the highest page on which the record occurrence can be stored, as specified in the schema WITHIN clause.
Low page (fullword) | The number of the lowest page on which the record occurrence can be stored, as specified in the schema WITHIN clause.
CALC-key length (halfword) | The length, in bytes, of the CALC-key value.
Record ID (halfword) | The ID assigned to the record type in the schema.
Area name (16-byte alphanumeric field) | The name of the area to which the record type is assigned.

- The second address constant contains the address of the CALC-key control block. This control block contains the CALC-key value, which is alphanumeric of varying length.

Considerations

When you define the IDMSCLCX routine to the operating system, remember that the routine must be reentrant if it will be stored in the following areas:

- z/OS -- The link pack area (LPA)
- z/VSE -- The shared virtual area (SVA)

IDMSDPLX

Entry point

- IDMSDPLX
  IDMSDBIO calls this routine:
  - At file open time
At file close time

- At disk write time (twice for each disk write)
- At the time of an I/O error (when the SYNAD exit is called)

At disk write time, control passes to the IDMSDPLX entry point immediately before IDMSDBIO issues the WRITE. Then, immediately after the WRITE, control passes again to IDMSDPLX before the #WAIT and CHECK macros are issued.

**Sample uses**

IDMSDPLX can be called to maintain duplicate database and/or journal files. The responsibilities of the IDMSDPLX user exit routine include:

- Acquiring variable storage needed by the routine
- Defining and allocating duplicate files
- Opening, closing, and writing to the duplicate files
- Managing I/O, recovery, and archival of the duplicate files

**Parameters**

When the IDMSDPLX entry point is called, registers 1 and 13 point to the following information:

Register 1 points to a parameter list that contains the following information:

- Parameter 1 -- The exit function:
  - 0 -- File open
  - 1 -- File close
  - 2 -- After disk write
  - 3 -- Before disk write
  - 4 -- When the SYNAD exit is called

- Parameter 2 -- The address of the current file control block (FCB) or journal control block (JCB). The address (storage anchor) of user storage is given in the following locations:
  - FCBUSER (fullword) in the FCB
  - JCBUSER (fullword) in the JCB

- Parameter 3 -- The address of the global storage anchor (DMCUSR in the DSECT) in the DMCL table (DMC).

- Parameter 4 -- The address of the current buffer memory area (BMA).
Parameter 5 -- The address of the fullword of reentrant storage. The storage is located in the work area of the IDMSDBIO variable information block (VIB).

Parameter 6 -- The address of the file status block (FST).

Register 13 points to the current entry in the task's TCE stack (as when a database procedure is called) for functions 0 through 3.

When function 4 "When the SYNAD exit is called", register 13 will point to 18 words in TCE. No stack size checking will be in effect if a #GETSTK or #CALL is executed for this function.

Registers R8 through R14 should be preserved by the duplex exit for all functions and control should be returned through R14.

Considerations

IDMSDBIO checks the availability of stack storage before calling IDMSDPLX (and before executing a WRITE). If sufficient storage does not exist, IDMSDBIO abends the task thread that is currently writing to the database or journal. The TCE stack size should be increased if this type of abend recurs.

Call IDMSDPLX by means of the following instruction:

```
BALR R14, R15
```

Control is passed to the IDMSDPLX exit in 31-bit mode. Ensure the exit changes the AMODE to 24-bit mode before issuing any I/O commands if EXCP-level I/O is not being used. Then ensure that the exit restores the previous AMODE at the completion of the I/O command.

The z/OS operating system requires that all files be opened or closed by the address space or partition's main task. If the IDMSDPLX exit will issue any OPEN or CLOSE macros for duplex files, it is the user's responsibility to insure that affinity is set to the CV's main task. This is especially important when multitasking is employed by the CV. To set affinity, do the following:

- If any DCBs or ACBs are opened by the exit, surround each OPEN with a pair of #AFFINITY macros as shown below:

  ```
  #AFFINITY SET, SCA=MAINTASK, RGSV=... 
  OPEN... 
  #AFFINITY RELEASE, RGSV...
  ```

  Multiple OPENs may be coded between the two #AFFINITY macros and code other than OPENs may be included, but no CV or DC macros may be coded in the instruction path between the two #AFFINITY macros.

- Similarly, any OS CLOSE macros must be enclosed between a pair of #AFFINITY macros as shown below:

  ```
  #AFFINITY SET, SCANUM=MAINTASK, RGSV=... 
  CLOSE... 
  #AFFINITY RELEASE, RGSV...
  ```

  Multiple CLOSEs may be coded within a single pair of #AFFINITY macros but no CV or DC macros may appear between them.
The 'RGSV=' parameter names any registers in the range 2 through 8, which should be preserved across the #AFFINITY macro. For example, if you want to save registers R2 through R8, the parameter would be coded like this:

, RGSV=(R2-R8)

Individual registers can be saved by coding the desired registers separated by a comma (,):

, RGSV=(R3, R5)

**IDMSIOXT**

**Entry point**

- **IDMSIOXT**
  IDMSDBIO calls this entry point before each disk read or write operation.

**Sample uses:**
The IDMSIOXT user exit can be called to capture I/O statistics associated with a database file.

**Parameters**

When IDMSIOXT is called:

- Register 1 contains the address of a fullword that points to one of the following:
  - File control block (FCB)
  - Journal control block (JCB)
- Register 4 contains:
  - 0 -- A read is being performed
  - 4 -- A write is being performed

**IDMSIOX2**

**Entry Point**

- **IDMSIOX2**
  IDMSDBIO calls this exit when any of the following occur:
  - Prior to a file open command
  - Prior to a file close command
  - Prior to an I/O call (Both Read and Write)
  - Prior to a #WAIT/CHECK on an I/O call
After a #WAIT/CHECK on an I/O Call (indicating whether the I/O was successful or in error)

When a read is satisfied from cache instead of doing an I/O

After a write to cache call on a WRITE I/O

This exit is NOT called for native VSAM files.

Sample uses

You can use this exit as follows:

- In place of or in addition to the exits: IDMSDPLX, IDMSJNL2, and IDMSIOXT.
- Maintain a duplicate database and/or journal file.
- Capture I/O statistics.
- Replace the normal I/O calls that CA IDMS issues with I/O calls issued by the exit.
- Force a file into input mode.

Calling the exit

This exit uses standard CA IDMS/DC system mode calling conventions. Use a #CALL statement to call this exit. You must compile the exit with the #MOPT ENV=SYS macro. The entry point must be defined using a #START macro and control returned using a #RTN macro.

You should code the #START macro with the MPMODE=CALLER option to reduce call overhead and to preserve the current MPMODE lock, if any, that may be held by the current task.

Using the exit with XA systems

On XA systems, the exit is called in Amode 31. If the exit issues CA IDMS/DC calls or when the control is returned, the same Amode must be in effect.

Using the exit in multitasking systems

In a multitasking system, it is the exit’s responsibility to establish affinity on the correct TCB before issuing OPEN and CLOSE macros. You can use the #AFFINITY macro for this.

The only resources locked for the current task (TCE) are the storage owned by the current task, and in the case of an open or close call, the FCB/JCB. It is the responsibility of the exit to control concurrent access to other resources.

To avoid putting CA IDMS/DC into an opsys wait use the #WAIT macro on an ECB before doing any opsys function that may wait on that ECB.

Shared or memory cache

If a file is defined to be in a shared cache (Parallel Sysplex environment) or memory cache, IDMSDBIO continues to read from and write to the cache even if the physical I/O is suppressed by the exit. Depending on how you use the exit, this may or may not be desirable. For instance I/O’s written to a
shadow file should not be written to a shared cache because that could corrupt the primary file through another CV, but writing the same I/O’s to a non-shared memory cache may be desirable. It is the responsibility of the exit to disable undesirable caching at open time. You can do this by setting the correct flags in #IOX2DS parameter list.

**Pages in cache buffer**

If a page being read is in a cache buffer, the physical I/O is bypassed. As a result, the calls to the Pre-Read and Pre-Read-Check exits are not made. However, the Post-Read exit is called with a flag set in the parameter list indicating this condition.

After a successful write, if the page is successfully written to the cache, the Post-Write exit is called a second time after the cache write, with the IOX2CAC flag set. If the cache write fails, the write I/O error exit is called with the IOX2CAC flag set, even if the I/O was successful and the Post-Write exit was called.

**Prefetch enabled**

When Prefetch is enabled for a file, multiple reads may be issued for a file before a CHECK is done. The work storage associated with an I/O on the Pre-Read exit remains constant for the pre-read and post-read calls for the same I/O, but other storage could change.

The Pre-Read exit precedes IDMSQSAM processing. If the Read is suppressed, it bypasses IDMSQSAM processing as well as the normal Read. If IDMSQSAM finds the record, the Pre-Check exit is still called, but the I/O on the primary file will have completed. If IDMSQSAM is enabled and the IOX2 exit is waiting on any I/O, it may negate the benefits of IDMSQSAM.

When I/O to a primary file does not require a check macro to be issued, for example, VSAM under DOS or QSAM, the Pre-Check exit is called anyway. In this case, a flag is set in the IOX2 parameter list indicating that the I/O is complete.

**Input mode processing**

A database file can be forced into input mode by using the Pre-Open exit to set the IOX2INPUT flag. The #IOX2DS copy book documents this flag. The file is opened in input mode and remains that way even if writes are attempted on the file. The Pre-Write exit can be used to intercept writes to the file; otherwise, any writes to the file will fail.

The Pre-Write exit can force the file to be reopened on the next write by setting the IOX2REOPEN flag. The file remains in the current mode for the current write, but the next write closes and reopens the file. If the file is not forced to input mode again, it is opened in update mode to satisfy the write.

**User anchor words**

The exit is provided with an address of a work field at the system level, another at the file level, and a third at the I/O level. The exit should not rely on any of these work fields residing in a particular control block. The exit should use the addresses provided in the parameter list.

We recommend that the work fields at the system level and the file level be used with the following rules:

- The word is used to anchor storage, not to store data.
• The storage should contain an 8-byte prefix:
  • CL4'xxxx' -- An eyecatcher unique to the storage.
  • A(0) -- A "next" address for future storage.
  • 0X -- User fields would follow.

• The first exit would anchor its storage in the word provided by the exit. If another exit also needed to anchor storage at the same level, it would follow the chain of user storage blocks and chain its storage to the last block in the chain.

• The four-byte eyecatcher should be unique to each block of storage, so an exit can identify its own storage.

• Once allocated a storage block should not be deleted, as this could break the chain.

The I/O level work field remains constant for the life of an I/O, but it is not guaranteed to last beyond the Post-I/O exit call. So you should not use this field to chain storage as you would with the more permanent work fields. The concern is that as storage comes, goes, and is reused, it is difficult to maintain a reliable chain.

You should not issue #GETSTG statements for each I/O call because this can affect performance.

Register usage

The following are the standard IDMS/DC conventions:

• R15/R14 contain entry point and return addresses. These are automatically handled by the #START and #RTN macros.

• R15 -- On Exit must contain a return code value.

• R13 -- Current stack pointer.

• R12 -- Base register for exit after #START.

• R11 -- A register automatically saved across DC calls.

• R10 -- CSA - Do not modify.

• R9 -- TCE - Do not modify.

• R8 -- R2 -- Available. They were saved prior to calling IDMSIOX2; no guarantee as to content.

• R1 -- Parameters - On entry.

• R0 -- No guarantee as to content.

Parameters

When the IDMSIOX2 exit is called, R1 points to a parameter list described by the #IOX2DS copy book. The parameter list contains the following information:
A function code defining when the exit was called:

0 -- Pre File Open
1 -- Pre File Close
2 -- Pre Read
3 -- Pre Read Check
4 -- Post Read
5 -- Read I/O Error
6 -- Pre Write
7 -- Pre Write Check
8 -- Post Write
9 -- Write I/O Error.

Flags that are used in some situations to coordinate control between IDMSDBIO and IDMSIOX2.

The address of:

- A fullword associated with the system, for exit use.
- A fullword associated with the file, for exit use.
- A fullword associated with a specific I/O, for exit use.
- The current FCB/JCB.
- The IOP.
- A list of buffer addresses and RBNs to be read/written. The last pair is marked with the 'X'80000000' bit in the buffer address.

Return codes

On return from the exit, R15 should contain one of the following values:

- 0 -- No Errors, Proceed with normal processing. Supported on all functions.
- 4 -- No Errors, Suppress next I/O function. Supported on the "Pre" functions; for example, pre-open, pre-read, pre-check, etc.
- 8 -- I/O Error. Supported on I/O functions only. IDMSDBIO behaves as if an I/O error had occurred on this file, returning a 30xx code to its caller.
- 12 -- Retry an I/O after an error. Supported on the I/O Error function only.

I/O error function

The IDMSIOX2 exit is called with the I/O Error function when an I/O error occurs on the primary file. No information on the error is passed.

This function is provided so the IDMSIOX2 exit can Cleanup or Wait on a pending I/O. You can also use this function to issue a request to retry the I/O. For example, suppose this exit were being used for duplexing. When the function requests to try the I/O in error again, the exit could suppress the I/O to the primary file when it retries the I/O and satisfy it from the duplex file.

Suppressing I/O
With the IDMSIOX2 exit, it is possible to suppress the I/O normally generated by IDMSDBIO. However it is the exit’s responsibility to handle the I/O itself.

For example the Pre-Read exit is called passing the address of a buffer and the RBN of the page that needs to be read into that file. The exit could issue the read to a duplex file and suppress the read to the primary file.

When the Pre-Read-Check exit is called, the exit would #WAIT on the ECB associated with its read. When complete, it could fill in the DBIO buffer and then suppress the DBIO CHECK. IDMSDBIO would then behave as if it had read the block itself. The Post-Read exit would not be needed in this case and could simply return. Or it could verify that the contents were in sync with its own version of the page.

### IDMSJNL2

**Entry point**

- **IDMSJNL2**
  
  IDMSDBIO calls this routine as follows:
  
  - Once before the journal buffer is written to the standard journal file
  - Once after the journal buffer is written to the standard journal file, which lets the exit know if the I/O to the standard journal file was successful
  - When the standard journal file is closed

  Control passes to the IDMSJNL2 entry point before the journal buffer is written to the standard journal file. At sites where an IDMSJNL2 user exit routine is not used, the journal buffer is written only to the standard journal file.

**Sample uses**

IDMSJNL2 can be used to write a duplicate journal file to perform the following functions:

- Collect statistical information
- Copy journal records as they are being written
- Create a backup journal in case of a permanent I/O error on a disk journal file
  
  To make output from an IDMSJNL2 user exit routine available to the ROLLBACK and ROLLFORWARD utility statements, compile a DMCL module that defines an archive journal with a block size equal to the size of the blocks written by the exit routine.

**Note:** For more information on these utilities, see the *Administrating section.*

**Parameters**

When IDMSJNL2 is called, register 1 points to a parameter list that contains the following information:
Parameter 1 -- Points to an address constant that contains the address of the journal control block. This is the only parameter passed if the high order bit is on. The length of the journal control block is always fixed and equals the length of the journal record, as defined in the DMCL. The journal control block contains the information listed in the following table.

**Note:** The journal control block resides in XA storage.

<table>
<thead>
<tr>
<th>Field</th>
<th>Field description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block length (halfword)</td>
<td>The length, in bytes, of significant journal data (see below) contained in the journal control block. This length includes the block-length halfword and the system-reserved-area halfword. The block length is -1 (negative one) if the standard journal file is being closed.</td>
</tr>
<tr>
<td>(halfword)</td>
<td>System-reserved area.</td>
</tr>
</tbody>
</table>
| Journal data (one or more alphanumeric journal records) | The journal records being written to the second journal file:
The length of the journal data section is fixed. The length equals the buffer size, as described for the journal file in the DMCL, minus 4 bytes. The length of actual data in this section is given in the block length halfword (see above).
If the journal data written to the journal control block does not completely fill the journal data section, the rest of the section contains undefined data. |

Parameter 2 -- Indicates if the standard journal file I/O was successful (0) or unsuccessful (X'00000BC3', which is equivalent to error code 3011).

**Considerations**

Output produced by an IDMSJNL2 user exit routine cannot be used by the ARCHIVE JOURNAL utility or for automatic recovery purposes.

The IDMSJNL2 user exit impacts the reentrancy of IDMSDBIO. If IDMSJNL2 performs an I/O function, the central version does not service any active transactions until the I/O is completed.

**DC/UCF Exits**

DC/UCF user exits allow the site to call site-specific system processing routines. You must associate each user exit routine with a user exit entry point. These entry points are predefined by the system.

- IDMSSVCX (see page 221)
- TCKREXIT (see page 228)
- USRIDXIT (see page 229)
- WAITEXIT (see page 230)
- WTOEXIT (see page 231)
- WTOREXIT (see page 238)

To include a DC/UCF user exit routine other than IDMSSVCX in the system, do the following:
1. Code the routine based on the information presented in this section.

2. Link edit the entry point with IDMSUXIT.

To include the IDMS SVC exit routine in the system on z/OS and z/VM systems, link it with the SVC as described in IDMSSVCX (see page 221). On z/VSE systems, assemble and link the exit routine as a standalone program.

At runtime, the system calls a user exit routine by means of the predefined entry point for the exit. After the exit routine is performed, control returns to the calling module.

If you do not link edit a given user exit entry point with IDMSUXIT, the user exit is bypassed at runtime, except for WTOEXIT and WTOREXIT, which can be loaded dynamically. In this case, system operations continue without interruption.

DC/UCF user exits are listed in the following table and described individually after the table.

<table>
<thead>
<tr>
<th>Entry point</th>
<th>DC/UCF module</th>
<th>Usage of exit</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDMSSVCX</td>
<td>The CA IDMS SVC (or equivalent module)</td>
<td>To capture transaction accounting statistics</td>
</tr>
<tr>
<td>TCKREXIT</td>
<td>DC/UCF startup module</td>
<td>To monitor time-related events</td>
</tr>
<tr>
<td>USRIDXIT</td>
<td>IDMSSTRT, UCFCICS, or IDMSINTC</td>
<td>To modify the user who is to sign on to the DC/UCF system.</td>
</tr>
<tr>
<td>WAITEXIT</td>
<td>DC/UCF startup module</td>
<td>To monitor operating system waits</td>
</tr>
<tr>
<td>WTOEXIT</td>
<td>IDMSOS00 or the DC/UCF startup module</td>
<td>To review DC/UCF messages</td>
</tr>
<tr>
<td>WTOREXIT</td>
<td>IDMSOS00 or the DC/UCF startup module</td>
<td>To modify operator communications</td>
</tr>
</tbody>
</table>

**IDMSSVCX**

**Entry point**

- **IDMSSVCX**
  IDMSSVCX is executed whenever an external request attempts to sign on to any DC/UCF system using the CA IDMS SVC. IDMSSVCX is called when the external request attempts signon and is entered in the application program’s address space. For z/VM, IDMSSVCX is entered in the virtual machine that is running DC/UCF.

The exit permits you to write information to the following areas:

- **z/OS and z/VSE** -- To the extension of the external request element (ERE) in the CA IDMS SVC
- **z/VM** -- To the accounting interface for the z/VM system

The SVC exit can contain multiple routines that each update a new variable item in the SVX (described below). Each routine should find the beginning of the SVX and then add the value in SVXAMTUS to find the location at which a new item may be added. Before adding an item, be sure
that the total extension length will not exceed the value in ESESVCXLN. Even the first routine in the
exit must use the existing value for the SVXAMTUS since some space in the extension may have been
reserved for internal CA IDMS items.

The total length of all the items added by the SVC exit, including the 4-byte fixed portion of each
item, must be added together to calculate the value for the SVCXLEN parameter on the #SVCOPT
macro that is discussed in Setting Up Interpartition Communication and the SVC (see page 58).

Sample uses

IDMSSVCX can be used to do the following:

- Perform extended user processing of ERUS accounting data. For example, checking the time of
day to verify that the user/job is allowed to run at that time.

- Extract information from online programs. For example, CICS accounting information can be
extracted from a CICS program.

- Handle task statistics records. Information written to the ERE extension is passed to DC/UCF, and
the first 40 bytes of one item subsequently are written to the task statistics record in the system
log file. After the log file has been offloaded to an archive file, data extracted by the IDMSSVCX
exit can be used to identify, sort, report on (using DC/UCF statistics reports), and otherwise
manipulate task statistics records.

If an item in the ERE extension has an item id of 'UA', the first 40 bytes of that item are written to
the system log file. If no item has an id of 'UA', then the first 40 bytes of the first item are used.
Task statistics must be enabled at system generation time, as discussed in Statistics.

ERE extension

The ERE extension consists of a 4 byte header followed by a variable number of entries. The header
and variable entries are defined by the SVX DSECT available in the installed source member #SVXDS.
The DSECT is as follows:

*******************************************************************************
***                     SVX: SVC EXTENSION DSECT.                           ***
***                     *UNUSED*                                          ***
*******************************************************************************
SVX DSECT

THE SVC EXTENSION HAS ONE HEADER, AS DESCRIBED BELOW.

* NOTE: TOTAL BYTES IN SVC EXTENTION IS STORED IN ESESVCXLN.

* SO, NBR OF FREE BYTES = ESESVCXLN - SVXAMTUS.

*******************************************************************************
SVXAMTUS DS H AMOUNT OF BYTES USED IN SVC EXTEN.
SVXFIXHL EQU "-SVXAMTUS" LENGTH OF FIXED HEADER.

*******************************************************************************
* THERE ARE N ITEMS WITHIN THE EXTENSION, N > OR = 0.                      *
* ADRESS OF FIRST Item = A(ERE) + ESESOXOF + ESES0XLN + SVXFIXHL          *
* ADDRESS OF NEXT Item = CURRENT_ADDR + SVXFIXIL + SVXITMLN                *
* "C" IN 1ST BYTE OF SVXITMID MARKS CA IDMS RESERVED ITEMS.               *
* USER DEFINED ITEMS MUST NOT HAVE "C" IN 1ST BYTE OF THE ID.              *
* IF AN SVCX ITEM EXISTS WITH AN ID OF "UA", THE FIRST 40 BYTES OF        *
* THAT ITEM'S DATA GETS WRITTEN IN THE TASK STATISTICS RECORD.            *
* IF NO "UA" ITEM EXISTS, THE DATA FROM THE FIRST ITEMS GETS USED.
* BEFORE LAYING DOWN A NEW EXTENSION, MAKE SURE THAT YOU DON'T EXCEED THE LENGTH STORED IN ESESVLN. *

********************************************************************
SVXITEM DS 0H ONE ITEM.
SVXITMLN DS H LENGTH OF SVXITEMD.
SVXITMID DS CL2 ITEMS ID.
SVXUACCT EQU C'UA' ID LENGTH FOR USER ACCOUNTING DATA.
SVXFIXIL EQU -*.SVXITEM LENGTH OF FIXED PART OF ITEM.
SVXITEMD DS OC ITEM DATA. LENGTH IS VARIABLE.
EJECT

Parameters

The following registers pass values to IDMSSVCX:

- Register 1 (z/OS and z/VSE) provides addressability to the local request element (LRE).
  - Register 1 (z/VM) provides a parameter list of the following control blocks:
    - Logical request element (LRE) address, at offset +0
    - External request element (ERE) address, at offset +4
    - External service element (ESE) address, at offset +8
    - z/VM element (VME) address, at offset +12
  - Register 2 (z/OS and z/VSE) provides addressability to the external service element (ESE).
  - Register 5 (z/OS) provides addressibility to the SVX.
  - Register 15 contains either of the following pieces of information:
    - z/OS and z/VSE -- Provides addressability to the external request element (ERE)
    - z/VM -- Contains the entry-point address

Considerations

A IDMSSVCX routine can be coordinated with the following two numbered exits, which are detailed in System-invoked Numbered Exits.

- Exit 5 (task termination exit I) can be invoked to extract accounting data at run-unit signoff.
- Exit 14 (bind run unit and ready area exit) can be invoked to perform security checks when a BIND RUN UNIT or READY AREA statement is issued. For example, exit 14 can cause a transaction to abort based on the job accounting information collected through IDMSSVCX.

IDMSSVCX can use:

- Register 0
- Register 3
- Register 4 for non-z/OS systems
- Register 5
Register 10 for z/OS systems

Return codes

None.

IDMSSVCX Examples

A sample IDMSSVCX macro appears below for z/OS, z/VSE, and z/VM operating systems. The sample is also available in the installed CA IDMS source library. Tailor the sample for your operating system environment.

The exit routines in this sample macro extract information from the jobcard associated with each external request unit signing on to the DC/UCF system.

z/OS sample IDMSSVCX user exit

```
MACRO &NAME. IDMSSVCX &VM=NO.,&SVLEN=0.,&VSE=NO.
AIF ('&VM'. EQ 'YES' AND '&VSE'. EQ 'YES'). ERROR
AIF ('&VM'. EQ 'YES'). VMXIT ? VM EXIT
AIF ('&VSE'. NE 'YES'). NOTVSE
****************************************************************
* ON ENTRY R1 = A(LRE)                                        *
* R2 = A(ESE)                                                    *
* R13 = SAME AS WHEN SVC INVOKED                                 *
* NOTE : FOR CICS THIS IS THE CSA ADDRESS                        *
* R15 = A(ERE)                                                   *
* R14 = RETURN ADDRESS                                          *
****************************************************************
********************************************************************************
* SVX@DTLN EQU 16         LENGTH OF DATA WE WANT TO STORE.       *
USING ERE,R15
USING ESE,R2
USING LRE,R1
AIF ('&NAME'. EQ ' '). VSE2
&NAME. CSECT
AGO .VSE2A
.VSE2 ANOP
IDMSSVCX CSECT
.VSE2A ANOP
BALR R8,0
USING *,R8
PUSH USING
LA R5,ERE          LOAD ERE BASE.
AH R5,ESESOXOF    POINT R5 TO
AH R5,ESESOXLN    SVC EXTENSION.
USING SVX,R5      (SVX)
LH R3,SVXAMTUS    GET AMOUNT USED.
LA R4,0(R3,R5)    POINT R4 TO UNUSED AREA.
USING SVXITEM,R4  (SVXITEM)
AH R3,=AL2(SVX@DTLN+SVXFIXIL) ADD IN WHAT WE WANT.
CH R3,ESESXVXLN   IF USED+WANTED > SVX LENG
BH SVX@END       THEN: END WITHOUT ADO.
STH R3,SVXAMTUS   NEW AMOUNT USED.
MVC SVXITMLN,=AL2(SVX@DTLN) ITEM'S LENGTH.
MVC SVXITMID,=AL2(SVXUACCT) ITEM'S ID.
********************************************************************************
* GET THE ACCOUNT INFORMATION AND MOVE IT INTO THE DATA AREA     *
* OF THE SVXITEM THAT R4 POINTS TO.                             *
********************************************************************************
*```
SPACE
CLC =C'VME',LREID COMING FROM VM?
BNE DSVX01 NO
*
USING VME,R1
MVC SVXITEMD(8),VMEUSER MOVE VM USER NAME
MVI SVXITEMD+8,C' '
MVC SVXITEMD+9(SVX@DTLN-9),SVXITEMD+8 CLEAR REST
USING LRE,R1
BR R14 EXIT
*
DSVX01 DS 0H
L R3,X'14' LOAD CURRENT COMREG AREA
LTR R3,R3 VALID ADDRESS?
BZR R14 ..NO GOOD - MUST BE LOST
*
USING COMREG,R3 (COMREG)
L R3,JAPART GET JOB PARTITION TABLE
LTR R3,R3 ANY ADDRESS?
BZR R14 NO - EXIT
DROP R3
*
USING JPT,R3 (JPT)
CLC JPTUFLD,=CL16' ' ANY USER = DATA
BER R14 ..NO - EXIT
CLC JPTUFLD,=16X'00' IS MEMORY NULL
BER R14 ..YES - NO FIELDS TO MOVE
MVC SVXITEMD(SVX@DTLN),JPTUFLD MOVE ACCOUNT INFO TO EXT
*
SVX@END DS 0H
BR R14 RETURN
*
LTORG
*
#REGEQU
MAPCOMR
*
JPT DSECT mm/dd/yy
JPTUFLD EQU JPT+56 16 BYTES OF ACCOUNTING INFO
*
COPY #LREDS
COPY #EREDS
COPY #ESEDS
COPY #SVXDS
MEXIT
.NOTVSE ANOP
COPY #EREDS
COPY #ESEDS
COPY #SVXDS
#REGEQU

IDMSSVCX CSECT
***********************************************************************
* This code is executed for non z/VSE and z/VM systems. It supports the
* the following systems: z/OS (all flavors).
* *
* Register utilization is as follows:
* R1 - Points to the LRE.
* R2 - Points to the ESE.
* R3 - Used to get to the account info.
* R4 - Used to return to IDMSSVC.
* R5 - Points to the SVX.
* R10 - Points to the SVXITEM.
* R12 - Bases the exit code.
* R14 - Must not be modified.
* R15 - Points to the ERE.
*
***********************************************************************

SPACE
SVX01000 DS 0H
BALR R12,0
USING *,R12
USING ESE,R2
USING SVX,R5

R5 POINTS TO THE SVC EXTENSION

B
SVX01100
Skip over the eyecatcher.

DC
CLB’IDMSVVCX’
Eyecatcher for dump reading.

+0
SVX01100

LA
R10,0(R3,R5)
POINT R10 TO UNUSED AREA.

AH
R3,=AL2(SVX@DTLN+SVXFIXIL) ADD IN WHAT WE WANT.

CH
R3,ESESVDLN
IF USED + WANTED > SVX LENGTH

BH
SVX@END
THEN: END WITHOUT ADD.

STH
R3,SVXAMTUS
NEW AMOUNT USED.

MVC
SVXITMLN,=AL2(SVX@DTLN) ITEM’S LENGTH.

MVC
SVXITMID,=AL2(SVXUACCT) ITEM’S ID.

****************************
* Get the account information and move it into the data area *
* of the SVXITEM that R10 points to.                        *
****************************

L
R3,X’B4’,(R9) PICKUP A(JSCB)

L
R3,X’104’,(R3)
R3---JCT-X’10’

LA
R3,X’10’,(R3)
R3---JCT

L
R3,X’28’,(R3)
R3---ACT

SA
R3,8 MOVE ADDR OVER

MVC
SVXITEMD(SVX@DTLN),X’31’(R3) MOVE ACCOUNT INFO TO EXT

SPACE

SVX@END

+0
BR
R4 RETURN TO IDMSMSVC

SVX@DTLN

EQU 40 LENGTH OF DATA WE WANT TO STORE.

MEEXIT

.VMEXIT

ANOP

******************************************************************************
* SVC EXIT FOR VM STARTS HERE. WHEN INVOKEID BY IDMSUSVC REGISTER          *
* CONTENTS WILL BE AS FOLLOWS:                                           *
* R1  PLIST OF FOLLOWING FORMAT                                          *
* +0  A(LRE)                                                            *
* +4  A(ERE)                                                            *
* +8  A(ESE)                                                            *
* +C  A(VME)                                                            *
* R13 STANDARD OS SAVE AREA                                            *
* R14 RETURN ADDRESS                                                    *
* R15 ENTRY POINT ADDRESS                                               *
*                                                                         *
* WE WILL ESTABLISH R12 AS BASE REGISTER WITHIN OUR CODE.               *
* PUSH AND POP ASSEMBLER DIRECTIVES ARE USED TO SAVE AND                *
* RESTORE THE BASE.                                                     *
*                                                                         *
* RETURN IS MADE VIA R14. SET R15 TO A NON-ZERO VALUE IF                 *
* THE RUN UNIT IS TO BE ABENDED.                                        *
******************************************************************************

SPACE

PUSH USING
ENTRY SVCEXIT
USING SVCEXIT,R15

SVCEXIT
B VMG6SYSNDX.
DC H’65VCXLEN’.

VMG6SYSNDX.
+0H

DROP R15

LR R12,R15 ESTABLISH BASE

USING SVCEXIT,R12
L R3,4(R1) R3 ERE
L R4,8(R1) R4 ESE

USING ERE,R3
USING ESE,R4
LA R5,ERE
AH R5,ESEXSOXOF
CA IDMS - 19.0

AH R5,ESESIXLN
USING SVX,R5 (SVX)
LH R2,SVXAMTUS GET AMOUNT USED.
LA R5,0(R2,R5) POINT R5 TO UNUSED AREA.
DROP R5
USING SVXITEM,R5 (SVXITEM)
AH R2,=AL2(SVX@DTLN+SVXFIXIL) ADD IN WHAT WE WANT.
CH R2,ESESIXLN IF USED+WANTED > SVX LENG
BH SVX@END THEN: END WITHOUT ADD.
STH R2,SVXAMTUS NEW AMOUNT USED.
MVC SVXITMLN,=AL2(SVX@DTLN) ITEM'S LENGTH.
MVC SVXITMID,=AL2(SVXUACCT) ITEM'S ID.
L R2,0(R1) R2 LRE
USING LRE,R2

*********************************************************************
* R5 NOW POINTS AT THE ERE EXTENSION ITEM.                        *
* INSERT ADDITIONAL CODE TO PROCESS THE REQUEST AND SET           *
* THE RETURN CODE FOLLOWING THIS COMMENT BLOCK.                   *
*********************************************************************
SVX@END DS 0H
LA R15,0
BR R14
DROP R12,R2,R3,R4,R5
POP USING
.ERROR ANOP
MNOTE 8, 'VSE=.. AND &VM=.. CAN NOT BE BOTH SPECIFIED AS YES'
MEND
/*

Steps to Add IDMSSVCX to Your System

Perform the following steps to add IDMSSVCX to your system:

1. Code SVC exit parameters

2. Modify the sample IDMSSVCX macro

3. Make the IDMSSVCX module available

4. Link edit IDMSSVCX

Code SVC Exit Parameters

Code SVC exit parameters appropriate to your operating system to specify that the IDMSSVCX exit is used at the site:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>SVC Exit Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS</td>
<td>Specify the following parameter in the #SVCOPT macro</td>
</tr>
<tr>
<td></td>
<td>SVCXLEN; specify, in bytes, the length of the ERE extension</td>
</tr>
<tr>
<td>z/VSE</td>
<td>Specify the following parameter in the #DEFSVC macro when IDMSVCTB is generated:</td>
</tr>
<tr>
<td></td>
<td>SVCXNAME; specify the name of your SVC user exit module</td>
</tr>
<tr>
<td></td>
<td>SVCXLEN; specify, in bytes, the length of the ERE extension</td>
</tr>
<tr>
<td>z/VM</td>
<td>Specify the following parameters in the IDMSSVCX macro:</td>
</tr>
<tr>
<td></td>
<td>VM=YES</td>
</tr>
<tr>
<td></td>
<td>SVCXLEN; specify, in bytes, the length of the ERE extension</td>
</tr>
</tbody>
</table>
Modify the sample IDMSSVCX macro

Modify the sample IDMSSVCX macro to add any site-specific logic such as site-coded validation or accounting routines.

Make the IDMSSVCX module available

Make IDMSSVCX available, as follows:

- **z/OS** -- After modifying the IDMSSVCX macro, assemble and link it into your custom load library by executing the *z/OS Assemble and Link-Edit JCL* (see page 465). Include your IDMSSVCX load module when creating the IDMS SVC.

  - **Note:** For more information, see *#SVCOPT Parameters* (see page 79).

- **z/VM** -- Create an Assembler source module that invokes the IDMSSVCX macro. This module can contain other exits to simplify maintenance.

- **z/VSE** -- Assemble and link IDMSSVCX as a stand-alone phase.

  1. Assemble and catalog the module using the sample JCL in *z/VSE Assemble JCL* (see page 476). Modify the JCL by substituting the following in place of the *Assembler input statements*.

    - **Note:** You must first place your modified IDMSSVCX macro in your idms.sublib.

      ```
PUNCH 'CATALOG idmssvcx.OBJ REPLACE=YES'
IDMSSVCX VSE=YES
END
```

  2. Link the IDMSSVCX program using the sample JCL in *z/VSE Link JCL* (see page 476). Modify the JCL by substituting the following statements in place of the *Linkage editor control statements*:

      ```
PHASE IDMSSVCX
INCLUDE idmssvcxENTRY IDMSSVCX
```

      - `idmssvcx`
        Specifies the name of the object module created by the assembly.

**TCKREXIT**

**Entry point**
TCKREXIT

This exit is called whenever the ticker interval expires. The ticker interval is established at system generation time by the TICKER INTERVAL parameter of the SYSTEM statement.

Sample uses

TCKREXIT can be used to monitor time-related events.

Considerations

This exit routine must not destroy the contents of registers 8 through 15. Note that no save area is passed in register 13.

Call this exit by using the following instruction:

BALR R14,R15

Parameters

None.

Return codes

None.

USRIDXIT

Entry point

USRIDXIT

Linking USRIDXIT

USRIDXIT can be linked with the following programs:

- IDMSUXIT (see page 208) for access from batch or from a TSO front-end.
- UCFCICS for UCF access from a CICS Transaction front-end. For more information, see the sample link-edit JCL in CICS Front-end JCL (see page 142).
- IDMSINTC for DML or SQL access from a CICS Transaction Server front-end. For more information, see the sample link-edit JCL in IDMSINTC (see page 164).

Description

Called whenever an automatic signon to a DC/UCF system is to be performed from a non-DC/UCF front-end.

Parameters

When USRIDXIT is called, register 1 points to the SON. The SON is described by the #SONDS DSECT.
Considerations

Two sample source modules are supplied in the installed IDMS source library: USRIDXIT and USRXCICS. The USRIDXIT source is suitable for batch and TSO access. It can also be used for UCF, DML, or SQL access from CICS provided that there is no need to code EXEC CICS commands in the exit. This will be the case at most CICS sites.

If it is necessary to code EXEC CICS commands, base your exit on the sample in source member USRXCICS. In all cases, compile the exit as a stand-alone load or object module as documented within the sample source member. The exit must have an ENTRY or CSECT name of USRIDXIT. Then link the exit with the appropriate system module as documented under Link edit with.

It may be desirable to have a different exit for TSO access and for batch access. In that case, make sure that a different copy of IDMSSTRT is used in each environment and that an appropriate version of the exit (or no exit) is linked with each copy.

Return codes

None.

WAITEEXIT

Entry point

WAITEEXIT

This exit is called whenever DC/UCF is about to issue an operating system WAIT, indicating that no DC/UCF tasks are ready to execute.

Sample uses

WAITEEXIT can be used to monitor the activity of the DC/UCF system.

Considerations

The exit routine can use registers 0 through 5 only. Note that no save area is passed in register 13.

Call this exit by using the following instruction:

BALR R14, R15

Parameters

One parameter is passed: the operating system event control block (ECB) list.

Return codes

None.
WTOEXIT

Entry point

WTOEXIT

Linking WTOEXIT

WTOEXIT can be linked as follows:

- With IDMSUXIT
- As a stand-alone load module and specify its name on the runtime options in the startup JCL.
- On z/VSE: link WTOEXIT as a standalone phase in a library that is searched at CV startup. Name the phase WTOEXIT so that the phase is recognized by the CV.

WTOEXIT is called each time a message is routed to the console. Control is passed to the entry point before the message is written to the console. By using this user exit, it is possible to:

- Review, alter, redirect, and suppress the text of messages written to the operator's console.
- (Under z/OS) Review, alter, redirect, and suppress the WTO DESC and ROUTCDE values of all messages written to the operator's console.

Sample uses

WTOEXIT can be called to perform the following activities:

- Automatically offload the disk journal file when it becomes full. To do this, WTOEXIT:
  1. Evaluates messages, looking for messages indicating that the disk journal file is full
  2. Passes control to a subroutine that invokes ARCHIVE JOURNAL
  3. Receives control back from the subroutine
  4. Returns control to the system

Sample WTOEXIT routines that perform this processing are shown later in this discussion for z/OS and z/VSE operating systems.

- Perform site-specific functions in response to particular DC/UCF messages.
  Note that altered system messages are not, under any circumstances, written to the DC/UCF log file. System messages are written to the log as they are originally issued.

For more information on using an OPS/MVS API rule as a replacement for the WTOEXIT, see Sample OPS/MVS API rule (see page 516). This can lead to improved performance when using the CA IDMS zIIP feature.

Parameters
When the WTOEXIT entry point is called, register 1 contains the address of the WTO message control block. The layout of this control block depends on the operating system in use. For z/OS, z/VSE, and z/VM the following table shows the layout of this control block:

<table>
<thead>
<tr>
<th>Field</th>
<th>Field description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message length (halfword initialized to 130 by the startup module)</td>
<td>The length, in bytes, of the message to the console.</td>
</tr>
<tr>
<td>(halfword)</td>
<td>z/OS and z/VSE only System-reserved area.</td>
</tr>
<tr>
<td>Message text (132-byte alphanumeric field)</td>
<td>The text of the message to the console. Do not adjust the message-length field in the WTO control block, even if the text length changes when the text is altered. When the return code is set to zero, DC/UCF adjusts the field by ignoring trailing blanks.</td>
</tr>
<tr>
<td>WTO DESC (halfword)</td>
<td>z/OS and z/VSE only The value of the WTO descriptor code that would be in effect if the standard z/OS or z/VSE WTO macro were invoked.</td>
</tr>
<tr>
<td>WTO ROUTCDE (halfword)</td>
<td>z/OS and z/VSE only The value of the WTO route code that would be in effect if the standard z/OS or z/VSE WTO macro were invoked.</td>
</tr>
</tbody>
</table>

**Considerations**

To call WTOEXIT in local mode, link edit WTOEXIT with IDMSUXIT. Calling conventions for local mode are the same as for central version.

**Return codes**

Before returning to the calling module, WTOEXIT places a return code of 0 or 4 in register 15:

- 0 -- The message text (altered or unchanged) is to be written to the console.
- 4 -- The message text is not to be written to the console.

**WTOEXIT Examples**

Sample Assembler WTOEXIT user exit routines appear below for z/OS and z/VSE operating systems. The corresponding source can be found in the installed source library. For a sample z/VM WTOEXIT routine, see the Installing section -- z/VM.

In each of these routines, WTOEXIT tests each DC/UCF message to see if it requests the ARCHIVE JOURNAL utility statement:

- If the message *does* request ARCHIVE JOURNAL, the exit routine:
  1. Invokes the ARCHIVE JOURNAL utility statement
  2. Sets register 15 to 4, indicating that the DC/UCF message is not to be written to the console
If the message does not request ARCHIVE JOURNAL, the exit routine sets register 15 to 0, routing the message to the console.

z/OS sample WTOEXIT user exit

*PROCESS OVERRIDE(NORENT)
WTO TITLE 'CA IDMS WTO EXIT FOR PLOG, PTRC AND AJNL'
*
* NOTE: THIS ROUTINE IS NOT REENTRANT. HOWEVER, THE ONLY MESSAGES
* WE'RE DOING SOMETHING WITH ARE ALWAYS OUTPUT BY RHDCWTL, WHICH
* IS HOLDING AN MPMODE LOCK (DC AT THE TIME OF THIS WRITING).
* THEREFORE, NO PROBLEMS WILL EXIST.
* THIS ROUTINE MUST BE MADE REENTRANT IF ANY MESSAGES OTHER THAN
* THOSE GENERATED THROUGH #WTL ARE TO BE PROCESSED.
*
WTOEXIT CSECT WRITE TO OPERATOR EXIT
PRINT NOGEN
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
STM R14,R12,12(R13) SAVE REGISTERS
LR R3,R15 R3 IS PROGRAM BASE
USING WTOEXIT,R3
LR R4,R1 R4 IS MESSAGE CONTROL BLOCK
USING WTOMCB,R4 BASE
LA R15,RUNAJNL
CLC WTOMTEXT(13),=C'IDMS DC205003' IF JOURNAL MESSAGE
BE SAVEREGS
LA R15,TSTFULL
CLC WTOMTEXT(13),=C'IDMS DC050001' IF LOG MESSAGE
BE SAVEREGS
LA R15,RUNPLOG
CLC WTOMTEXT(13),=C'IDMS DC050004' LOG FULL?
BE SAVEREGS
LA R15,TRCTEST
CLC WTOMTEXT(13),=C'IDMS DC050024' IF TRC MESSAGE
BE SAVEREGS
LA R15,TRCRUN
CLC WTOMTEXT(13),=C'IDMS DC050027' TRC FULL?
BE SAVEREGS
B RETURNLM
*
SAVEREGS DS 0H
ST R13,SAVEAREA+4 AND SET BACKWARD
LA R12,SAVEAREA AND FORWARD CHAINS
ST R12,8(R13)
LR R13,R12
BR R15 OFF TO THE REQUESTED ROUTINE
*
RUNAJNL LA R0,0
L R15,=A(SETAMODE)
BALR R14,R15
OPEN (JESRDR,(OUTPUT))
OPEN (AJNLJOB,(INPUT))
NEXTCARD GET AJNLJOB,CARD
PUT JESRDR, CARD
B NEXTCARD

CLOSES CLOSE (JESRDR)
CLOSE (AJNLJOB)
LA R0, 1
L R15, =A(SETAMODE)
BALR R14, R15
B RETURN

* TSTFULL
LA R6, WTOMTEXT
LH R5, WTOMLEN
AR R6, R5
LA R5, WTOMTEXT+10

NEXCHAR
LA R5, 1(R5)
CR R5, R6
BH RETURN

CLI 0(R5), C' %'
BNE NEXCHAR
S R5, =F' 2'
PACK PNUM, 0(2, R5)
CP PNUM, =P' 25'
BNL RUNPLOG
B RETURN

RUNPLOG
LA R0, 0
L R15, =A(SETAMODE)
BALR R14, R15
OPEN (JESRDR, (OUTPUT))
OPEN (PLOGJOB, (INPUT))

NEXONE GET PLOGJOB, CARD
PUT JESRDR, CARD
B NEXONE

CLOSE2 CLOSE (JESRDR)
CLOSE (PLOGJOB)
LA R0, 1
L R15, =A(SETAMODE)
BALR R14, R15

* *

TRCTEST
LA R6, WTOMTEXT
LH R5, WTOMLEN
AR R6, R5
LA R5, WTOMTEXT+10

TRCNXCH
LA R5, 1(R5)
CR R5, R6
BH RETURN

CLI 0(R5), C' %'
BNE TRCNXCH
S R5, =F' 2'
PACK PNUM, 0(2, R5)
CP PNUM, =P' 25'
BNL TRCRUN
B RETURN

TRCRUN
LA R0, 0
L R15, =A(SETAMODE)
BALR R14, R15
OPEN (JESRDR, (OUTPUT))
OPEN (PTRCJOB, (INPUT))

TRCNEXT GET PTRCJOB, CARD
PUT JESRDR, CARD
B TRCNEXT

TRCCLOS CLOSE (JESRDR)
CLOSE (PTRCJOB)
LA R0, 1
L R15, =A(SETAMODE)
BALR R14, R15

* RETURN
L R13, SAVEAREA+4
RETURNLM R14, R12, 12(R13)
XR R15, R15
BR R14

*
SETAMODE DS 0H
LA R14,0,(,R14)
SLL R0,31
OR R14,R0
BSM R15,R14
*
SAVEAREA DC 18F'0'
PNUM DS PL2
CARD DS CL80
JESRDR DCB BLKSIZE=80,LRECL=80,RECFM=F,DDNAME=JESRDR,DSORG=PS, MACRF=PM
AJNLJOB DCB DDNAME=AJNLJOB,DSORG=PS,MACRF=GM,EODAD=CLOSES
PLOGJOB DCB DDNAME=PLOGJOB,DSORG=PS,MACRF=GM,EODAD=CLOSE2
PTRCJOB DCB DDNAME=PTRCJOB,DSORG=PS,MACRF=GM,EODAD=TRCCLOS
LTORG
WTOMCB DSECT WTO MESSAGE CONTROL BLOCK
WTOMLEN DS H MESSAGE LENGTH
WTOMFLGS DS H MESSAGE CONTROL SYSTEM FLAGS
WTOMTEXT DS CL132 MESSAGE TEXT
WTOMDESC DS H MESSAGE DESCRIPTOR CODE
WTOMROUT DS H MESSAGE ROUTE CODE
END

z/VSE sample WTOEXIT user exit

TITLE 'WTO EXIT FOR DOS'
WTOEXIT CSECT 00001000
* ************************************************************** 00002000
* R2 - XPCCB 00004000
* R3 - WTOEXIT BASE REGISTER 00006000
* R4 - WTO MESSAGE CTL BLOCK 00006000
* R5 & R6 - WORK REGS 00007000
* R7 - SPLWTO - POWER SPOOL 00008000
* R8 & R9 - WORK REGS 00009000
* R10- DATA TO BE SENT TO POWER VIA SPL 00010000
* R11- DATA TO BE RCVD FROM POWER VIA SPL 00011000
* ************************************************************** 00012000
* #REGEQU 00013000
R0 EQU 0 00014000
R1 EQU 1 00015000
R2 EQU 2 00016000
R3 EQU 3 00017000
R4 EQU 4 00018000
R5 EQU 5 00019000
R6 EQU 6 00020000
R7 EQU 7 00021000
R8 EQU 8 00022000
R9 EQU 9 00023000
R10 EQU 10 00024000
R11 EQU 11 00025000
R12 EQU 12 00026000
R13 EQU 13 00027000
R14 EQU 14 00028000
R15 EQU 15 00029000
WTOEXIT CSECT 00030000
ENTRY WTOEXIT 00031000
USING WTOEXIT,R15 ESTABLISH ADDRESSABILITY 00032000
STM R14,R12,12(R13) SAVE REGISTERS 00033000
ST R13,SAVEAREA+4 AND SET BACKWARD 00034000
LA R12,SAVEAREA AND FORWARD CHAINS 00035000
ST R12,8(R13) 00036000
LR R13,R12 00037000
LR R3,R15 R3 IS PROGRAM BASE 00038000
DROP R15 00039000
USING WTOEXIT,R3 00040000
LR R4,R1 R4 IS MESSAGE CONTROL BLOCK 00041000
USING WTOMCB,R4 BASE 00042000
MVI SPLIND,X'00' INITIALIZE SWITCH 00043000
CK205003 CLC WTOMTEXT(13),=C'IDMS DC205003' IF JOURNAL MESSAGE 00044000
BNE CK50001 NOT JNL -- CHECK LOG 00045000
LA R9,AJNLJOB 00046000
LA R5, AJNLEN
MVC JOBNA, JNAME
B CONNECT

CK50001 CLC WTOMTEXT(13), 'C,IDMS DC050001' IF LOG MESSAGE
BNE RETURN

TSTFULL LA R6, WTOMTEXT
LH R5, WTOMLEN
AR R6, R5
LA R5, WTOMTEXT+10

NEXCHAR LA R5, 1(R5)
CR R5, R6
BH RETURN
CLI 0(R5), 'C','%'
BNE NEXCHAR
S R5, =F'2'
PACK PNUM, 0(2, R5)
CP PNUM, =P'2'
BL RETURN

RUNLOG DS 0H
LA R9, PLOGJOB
LA R5, PLOGLEN
MVC JOBNA, PNAME

OI SPLIND, X'40' SET LOOK FOR PLOG IND

* *********************************************************
* ISSUE IDENTIFY & CONNECT TO POWER
* *********************************************************

* *********************************************************
* CHECK FOR PLOG JOB ALREADY IN RDR QUEUE
* *********************************************************

CONNECT DS 0H
LA R7, SPLWTO
USING SPLWTODS, R7 POINT TO PWRSPL
LA R2, WTXPCCB
USING IBXPCCB, R2 POINT TO XPCCB
LA R10, IBXUSR SR POINT TO DATA TO BE SENT
USING PXUSER, R10
LA R11, IBXUSR SR POINT TO DATA TO BE RCVD
USING PXUSER, R11
XPCC FUNC=IDENT, XPCCB=WTOXPCCB
CLM R15, 1, X'08' ERROR: R15 LT 8 SAYS OK
BO SPOOLLRR RC OF 8 SAYS NO GOOD ON ID
XPCC FUNC=CONNECT, XPCCB=WTOXPCCB
LTR R15, R15 CONNECT AVAILABLE?
CLM R15, 1, X'08' ERROR: R15 = 8 SAYS NO GOOD
DBL SPOOLLRR R15 = 4 SAYS WAIT FOR POWER TO POST
SR R6, R6

BUSY DS 0H
SETIME 5, TECB POST ECB
WAIT TECB WAIT 5 SECONDS
TM IBXECB=2, X'80' CONN MADE??
BO CONNOK YET- DON'T HAVE TO WAIT
LA R6, 1(R6)
CH R6, =H'5' WAIT 5 SECONDS 5 TIMES
CR R6, R5
LA R6, 1(R6)
RH R5, WTOMLEN
TSTFULL LA R6, WTOMTEXT
BNE RETURN
CK50001 CLC WTOMTEXT(13), 'C,IDMS DC050001' IF LOG MESSAGE
CLM R15, 1, X'08' ERROR: R15 LT 8 SAYS OK
BO SPOOLLRR RC OF 8 SAYS NO GOOD ON ID
XPCC FUNC=CONNECT, XPCCB=WTOXPCCB
LTR R15, R15 CONNECT AVAILABLE?
CLM R15, 1, X'08' ERROR: R15 = 8 SAYS NO GOOD
DBL SPOOLLRR R15 = 4 SAYS WAIT FOR POWER TO POST
SR R6, R6

CONNOK DS 0H
OI SPLIND, X'80' SET GOOD CONNECT SWITCH
TM SPLIND, X'40' LOOK FOR PLOG IN QUEUE?
BZ SUBMIT

* *********************************************************
* CHECK FOR PLOG JOB ALREADY IN RDR QUEUE
* *********************************************************

LOK4PLOG DS 0H
PWRSPL TYPE=UPD, FUNC=DISPLAY, JOBN=PNAME, QUEUE=RDR, REQ=CTL,
SPL=SPLWTO
MVI PXUBTYP, PXUBTSPL CHECK FRO PLOG JOB IN RDR
MVI PXUACTL, 0
STCM R7, 7, IBXADDR
CA IDMS - 19.0

LA R15,SPLGSLN 00118000
ST R15,IJBXBLN 00119000
XPCC XPCCB=WTOXPCCB,FUNC=SEND R 00120000
LTR R15,R15 ZERO RC? 00121000
BNZ SPOOLERR NO 00122000
LA R6,IJBXSEC B 00123000
WAIT (R6) WAIT FOR COMPLETION OF SEND R 00124000
CLI IJBXREAS,0 ANY ERRORS? 00125000
BNE SPOOLERR 00126000
CLI PXPRETCD,0 RC OF 0 SAYS PLOG JOB ALREADY IN RDR 00127000
BE DISC 00128000
CLI PXPRETCD,X'04' NO PLOG JOBS FOUND? 00129000
BNE SPOOLERR 00130000
BNE SPOOLERR 00131000
BNE SPOOLERR 00132000
* *********************************************************
* ISSUE OPEN TO POWER USING SPL *
* *********************************************************
SUBMIT PWR SPL TYPE=UPD, SPL=SPLOWT0, REQ=PUT, QUEUE=RDR 00138000
MVI PXUBTYP, PXUBT SPL 00139000
MVI PXUACT1,0 00140000
MVI PXUSIGNL,0 USE SPL FOR PUT OPEN REQUEST 00141000
STCM R7,7, IJBXADR 00142000
LA R15, SPLGSLN 00143000
ST R15, IJBXBLN 00144000
OI IJBXIND, IJBXM80 SET LIST TYPE 00145000
XPCC XPCCB=WTOXPCCB,FUNC=SEND R 00146000
LTR R15,R15 ZERO RC? 00147000
BNZ SPOOLERR NO 00148000
LA R6, IJBXSEC B 00149000
WAIT (R6) WAIT FOR COMPLETION OF SEND R 00150000
CLI IJBXREAS,0 ANY ERRORS? 00151000
BNE SPOOLERR 00152000
CLI PXPRETCD,0 RC OK? 00153000
BNE SPOOLERR 00154000
BNE SPOOLERR 00155000
* *********************************************************
* MOVE JOB TO POWER RDR QUEUE *
* *********************************************************
SET ADDR DS 0H 00159000
STCM R9,7, IJBXADR 00160000
OI IJBXIND, IJBXM80 SPECIFY PROPER BUFFER TYPE 00161000
ST R5, IJBXBLN STORE LENGTH 00162000
MVI PXUBTYP, PXUBT ND B MOVE BUFF TYPE DATA 00163000
MVI PXUACT1, PXUACT BD SET EOD INDICATOR 00164000
MVI SPLGJB, JOBNA MOVE JOBNAME 00165000
MVI PXUSIGNL,0 USE SPL FOR PUT OPEN REQUEST 00166000
XPCC XPCCB=WTOXPCCB,FUNC=SEND R 00167000
LTR R15,R15 00168000
BNZ SPOOLERR NO 00169000
LA R6,IJBXSEC B 00170000
WAIT (R6) WAIT FOR COMPLETION OF SEND R 00171000
CLI IJBXREAS,0 ANY ERRORS? 00172000
BNE SPOOLERR 00173000
CLI PXPRETCD,0 RC OK? 00174000
BNE SPOOLERR 00175000
CLI PXPFBCD,0 FEEDBACK OK? 00176000
BNE SPOOLERR 00177000
B DISC YES -- GET OUT 00178000
SPOOL ERR DS 0H ADD ANY ERROR HANDLING HERE 00179000
PDUMP SPLWTO,BUFEND 00180000
TM SPLIND,X'80' WAS CONNECT DONE? 00181000
BNO RETURN 00182000
* *******************************************************
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* DISC DS 0H 00188000
CA IDMS - 19.0

XPCC XPCCB=WTOXPCCB,FUNC=DISCONN GO DISCONNECT 00189000
LTR R15,R15 00189000
BNZ RETURN YES -- JUST RETURN 00191000
TERM DM DS 0H 00192000
XPCC XPCCB=WTOXPCCB,FUNC=TERMIN SAY GOODBYE 00193000
RETURN DS 0H RESTORE OLD SAVE AREA 00194000
LM R13,SAVEAREA+4 RESTORE OLD SAVE AREA 00195000
LM R14,R12,12(R13) AND REGISTERS 00196000
X R15,R15 R15 TO ZERO FOR RETURN CODE 00197000
BR R14 00198000
PNAME DC CL8'PLOGJOB' 00199000
JNAME DC CL8'AJNLJOB' 00200000
JOBNA DC CL8' ' 00201000
SAVEAREA DC 18F'0' 00202000
* ORRECBA DS A 00203000
* ORREPA DS A 00204000
PNUM DS PL2 00205000
TECB TECB 00206000
AJNLJOB DC A(80,0),CL80** ** JOB JNM=AJNLJOB,CLASS=A,PRI=9,DISP=D,U* 00207000
SER=CULLDBA' 00208000
DC A(80,0),CL80** ** LST CLASS=Z' 00209000
DC A(80,0),CL80'// JOB AJNLJOB S851CC07500' 00210000
DC A(80,0),CL80'// EXEC PROC=DBDCLIB' LIBDEFS 00211000
DC A(80,0),CL80'// EXEC PROC=DBDCAJNL' AJNL DLBL=EXEC 00212000
DC A(80,0),CL80'&.&,.' 00213000
DC A(80,0),CL80'** $E0J' 00214000
AJNLLEN EQU *-AJNLJOB 00215000
PLOGJOB DC A(80,0),CL80** ** JOB JNM=PLOGJOB,CLASS=A,PRI=9,DISP=D,U* 00216000
USER=CULLDBA' 00217000
DC A(80,0),CL80** ** LST CLASS=Z' 00218000
DC A(80,0),CL80'// JOB PLOGJOB S851CC07500' 00219000
DC A(80,0),CL80'// EXEC PROC=DBDCLIB' LIBDEFS 00220000
DC A(80,0),CL80'// EXEC PROC=DBDCPLOG' PLOG DLBL=EXEC 00221000
DC A(80,0),CL80'&.&,.' 00222000
DC A(80,0),CL80'** $E0J' 00223000
PLOGLEN EQU *-PLOGJOB 00224000
SPLIND DC XLI'00' '80' MEANS CONNECT DONE '40' LOOK4PLOG 00225000
SPLWTO PJNSTYLE TYPE=GEN,JOBN=JOBNA,PREFIX=WTO,QUEUE=RDR,REQ=PUT, USERID=CULLDBA,OPT=NOWAIT 00226000
WTOXPCCB XPCCB APPL=WOTEXIT,TOAPPL=SYSPWR,REAPERA=(WTOBUF,200) 00227000
WTOBUF DC CL200' ' 00228000
BUFEND EQU * 00229000
WTOMCB DSECT WTO MESSAGE CONTROL BLOCK 00230000
WTOMLEN DS H MESSAGE LENGTH 00231000
WTOMFLGS DS H MESSAGE CONTROL SYSTEM FLAGS 00232000
WTOMTEXT DS CL132 MESSAGE TEXT 00233000
WTOMDESC DS H MESSAGE DESCRIPTOR CODE 00234000
WTOMROUT DS H MESSAGE ROUTE CODE 00235000
MAPXPCCB 00236000
SPLWTODS PJNSTYLE TYPE=MAP 00237000
END WTOEXIT 00238000

WTOREXIT

Entry point

WTOREXIT

Link edit with:

- IDMSUXIT
- As a stand-alone load module and specify its name on the runtime options in the startup JCL.

23-Dec-2017 238/524
On z/VSE: link WTOREXIT as a standalone phase in a library that is searched at CV startup. Name the phase WTOREXIT so that the phase is recognized by the CV.

The DC/UCF startup module calls this user exit each time the system is ready to solicit an operator command.

To notify DC/UCF that an operator command is present, WTOREXIT:

1. Places the length and text of the command in the WTOR reply area. This is true for commands either issued by an operator or generated by the exit routine.

2. Posts a reply event control block (ECB) to notify the DC/UCF that an operator command is present:
   - z/OS and z/VM -- By performing an operating-system POST to set the reply ECB completion code to 240
   - z/VSE -- By performing an operating system POST

Posting of the reply ECB causes DC/UCF to process the operator command placed in the WTOR reply area. After processing the command, DC/UCF clears the reply ECB and the WTOR reply area. DC/UCF is again ready to solicit an operator command, which in turn prompts another call to the WTOREXIT entry point.

**Sample uses**

The WTOREXIT user exit routine can be used to:

- Review and optionally alter commands before they reach the DC/UCF system.
- Solicit operator commands from the console.
- Generate site-specific operator commands.
- Disallow operator commands. To accomplish this, see parameter descriptions below.

Having WTOREXIT generate commands is especially useful when direct operator-to-system communication is not allowed. Even when direct operator-to-system communication is allowed, the WTOREXIT routine can solicit operator commands from the console and/or generate its own operator commands and pass them to DC/UCF.

**z/VSE Systems**

The WTOREXIT can be used to intercept and reply to immediate WTOR requests. It can reply to a general operator prompt, but cannot suppress the outstanding operator prompt. It cannot intercept or suppress the OC EXIT.

**Considerations:** For details on how DC/UCF allows operators to enter console commands when WTOREXIT is not linked, see the System Tasks and Operator Reference section.

**Parameters**

When the WTOREXIT entry point is called:
• Register 0 contains the address of a reply event control block (ECB).

• Register 1 contains the address of the WTO reply area to which operator commands are written. The following table shows the layout of the WTO reply area:

<table>
<thead>
<tr>
<th>Field</th>
<th>Field description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reply length (halfword)</td>
<td>The length, in bytes, of the reply.</td>
</tr>
<tr>
<td>Reply text (alphanumeric, varying length)</td>
<td>The text of the reply message.</td>
</tr>
</tbody>
</table>

The WTOREXIT routine can save the addresses of the reply ECB and the WTO reply area for future use.

Return codes

WTOREXIT returns either 0 or 4 in register 15.

• 0 -- Suppress issuing of WTOR command:
  - z/OS and z/VM -- Disallows direct operator-to-system communication by suppressing the DC/UCF console outstanding-response message.
  - z/VSE -- Allows the exit to reply to a WTO request instead of prompting the operator console.

Typically, WTOREXIT sets register 15 to 0 when the exit routine solicits its own operator commands and passes them to the DC/UCF system.

• 4 -- Allows direct operator-to-system communication:
  - z/OS and z/VM -- By directing the DC/UCF outstanding-response message to the console
  - z/VSE -- By allowing an immediate WTOR command to be issued.

If the WTOREXIT user exit routine sets register 15 to 4, the routine cannot review or alter the text of future operator commands before the commands reach the DC/UCF system. This is because the exit is never called again.

WTOREXIT Examples

A sample WTOREXIT user exit for a z/OS system appears below. This exit routine:

1. Suppresses the DC/UCF outstanding-response message
2. Solicits operator commands from the console
3. Passes to DC/UCF the resultant operator commands

This WTOREXIT user exit could include additional code to alter the text of acquired operator commands before passing the commands to the ECB.

z/OS sample WTOREXIT user exit
**SAMPLE WTOREXIT**

This routine is intended to serve as a sample WTOREXIT routine only. It disables the standard CV WTOR routine and replaces it with a different WTOR.

In order to be asynchronous with the main CV task, this routine attaches an OS subtask that waits for operator responses.

Note the following:

1. Typically, the responses from the operator would be screened or otherwise processed by WTOREXIT before being passed to the CV. In this example, operator responses are passed to the CV without being reviewed or processed by the WTOREXIT user exit.

---

```assembly
WTORCSC CSECT
   DC   CL8'WTORSAMP'
 *
   #REGEQU
   EJECT

WTORCSC CSECT
   ENTRY WTOREXIT

WTOREXIT STM R14,R12,12(R13)
   LR R12,R15           Establish addressability
   USING WTOREXIT,R12
   STM R0,R1,SAVEPARM   Save input parameters
   LTR R0,R0            Is this an immediate WTOR?
   BZ WTOREXIT200       Yes; do not suppress

   The above test for R0=0 is only relevant to IDMS-DC systems, and
   applies to special WTORs that prompt for startup parameter
   overrides. In that case, DC should be allowed to issue the WTOR,
   so that register 15 will be set to 4.

   WTOREXIT100 L R1,WTORECBA Clear ECB and
   XC 0(4,R1),0(R1) then
   WTOREXIT101 LM R14,R12,12(R13) return
   SR R15,R15          Do not perform standard WTOR
   BR R14
   SPACE 3
   WTOREXIT200 LM R14,R12,12(R13) Return and allow the
               WTOR to be issued by CV/DC
   LA R15,4
   BR R14
   DROP R12
   EJECT

ATTNXIT STM R14,R12,12(R13) Save registers
   LR R12,R15           Establish addressability
   USING ATTNXIT,R12

ATTNXIT1 WTOR 'SPECIAL PRAOPT',WTORREP,WTORMSGL,WTORECB
   WAIT ECB=WTORECB Wait for reply
   XC WTORECB,WTORECB Clear reply ECB
   L R5,WTORECBA If ECB is still posted,
   TM 0(R5),X'40' then prior message not yet
   BO ATTNXNO processed
   L R4,WTORREPA Point to reply area
   LH R3,0(.R4) Get length
   LA R5,WTORMSGL Get input length
   CR R3,R5 Do not exceed buffer length
   BNL +*6
   LR R5,R3
   SH R5,=H'1' Subtract 1 for MVC
   BM +*8 but do not do it if length=0
   EX R5,MOVCREPLY Move reply string
   AH R5,=H'1' Fix length
   STH R5,0(.R4)
   L R1,WTORECBA Post user
```

---

23-Dec-2017
CICS Exits

Contents

- IDMSCEON (see page 243)
- IDMSCEOX (see page 244)
- OPTIQXIT (see page 245)
- OPTIXIT (IDMSINTC) (see page 247)
  - OPTIXIT Example (see page 249)
- OPTIXIT (IDMSINTL) (see page 250)
  - Example OPTIXIT for the IDMSINTL Interface (see page 251)

Tools Exits (see page 252)

- Hooked Exits (see page 252)

CICS user exits allow the site to call site-specific routines in a CICS environment. You must associate each user exit routine with a user exit entry point. These entry points are predefined by the system.

To include a CICS exit routine in the system, do the following:

1. Code the routine based on the information presented in this section.

2. Link edit an exit for OPTIQXIT or OPTIXIT with the CA IDMS interface stub module specified for the user exit routine in the following table.

3. Link edit a IDMSCEON or IDMSCEOX exit as a stand-alone load module.

At runtime, the system calls a user exit routine by means of the predefined entry point for the exit. After the exit routine is performed, control returns to the calling module.
If you do not link edit an OPTIQXIT or OPTIQXIT with its CA IDMS stub module, the user exit is bypassed at runtime. In this case, CICS operations continue without interruption.

CICS user exits are listed in the following table and described individually after the table.

In addition to the exits described below, there are special considerations for DC/UCF user exit USRIDXIT. For more information, see USRIDXIT (see page 229).

<table>
<thead>
<tr>
<th>Entry point</th>
<th>CA IDMS module</th>
<th>Usage of exit</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDMSC EON</td>
<td>IDMSINTC</td>
<td>To gain control upon entry into IDMSINTC interface before the requested function is passed along to the appropriate client.</td>
</tr>
<tr>
<td>IDMSC EOX</td>
<td>IDMSINTC</td>
<td>To gain control upon return from processing the requested function and before control is returned to the caller of IDMSINTC.</td>
</tr>
<tr>
<td>OPTIQX IDMSINTC IT</td>
<td>IDMSINTC</td>
<td>To alter dynamically so that an individual SQL session can be routed to a specific back-end CV</td>
</tr>
<tr>
<td>OPTIXI IDMSINTC T</td>
<td>IDMSINTC</td>
<td>To alter dynamically the location where the request will be processed.</td>
</tr>
<tr>
<td>OPTIXI IDMSINTL T</td>
<td>IDMSINTC</td>
<td>To alter dynamically the location where the request will be processed</td>
</tr>
</tbody>
</table>

IDMSCEON

**Entry point**

The entry point of the IDMSCEON module/phase you assemble and link will be used.

**Link edit with**

Does not need to be linked with any module. The module is loaded when the IDMSINTC interface is started if the EXTXIT=YES parameter is specified in the CICSOPT macro.

The IDMSCEON module allows you to gain control upon entry into IDMSINTC interface before the requested function is passed along to the appropriate client.

**Considerations**

This exit routine is for use under CICS only. The load module name/core image library phase name must be IDMSCEON.

The exit routine must be linked to AMODE 31.

The module must be defined to CICS using the same definition parameters used by IDMSINTC.

```
DEFINE PROGRAM(IDMSCEON)
GROUP(IDMSGRP) LANGUAGE(ASSEMBLER) CEDF(NO) RSL(PUBLIC) RES(YES) EXECKEY(CICS)
```
This exit is called via a "BALR R14,R15" instruction. All registers must be preserved within your exit code and restored before returning to IDMSINTC.

Register Usage

- R1 Callers Plist (Caller of IDMSINTC)
- R4 TWACBIX
- R5 Request Type Index
- R12 DFHEISTG
- R13 TOP of the TCE stack
- R14 Return Address
- R15 Exits Base Register

Parameters

None.

Return codes

None.

IDMSCEOX

Entry point

The entry point of the IDMSCEOX module/phase you assemble and link will be used.

Link edit with

Does not need to be linked with any module. The module is loaded when the IDMSINTC interface is started if the EXTXIT=YES parameter is specified ub tge CICSOPT macro.

The IDMSCEOX module allows you to gain control upon return from processing the requested function and before control is returned to the caller of IDMSINTC.

Considerations

This exit routine is for use under CICS only. The load module name/core image library phase name must be IDMSCEOX.

The exit routine must be linked to AMODE 31.

The module must be defined to CICS using the same definition parameters used by IDMSINTC.
DEFINE PROGRAM(IDMSEOX)
GROUP(IDMSGRP) LANGUAGE(ASSEMBLER) CEDF(NO) RSL(PUBLIC)
RES(YES) EXECKEY(CICS)

This exit is called via a "BALR R14,R3" instruction. All registers must be preserved within your exit code and restored before returning to IDMSINTC.

Register Usage

- R3 Exits Base Register
- R12 DFHEISTG
- R14 Return Address

Parameters

None.

Return codes

None.

**OPTIQXIT**

**Entry point**

OPTIQXIT

**Link edit with**

IDMSINTC Interface Program

This exit is invoked whenever an IDMS SQL session is initiated within the CICS environment. It allows the default OPTI structure to be dynamically modified so that an individual SQL session can be routed to a specific back-end CV.

IDMSINTC copies the static IDMSOPTI structure into dynamic storage and passes it to the user routine, which may alter it based on site-specific rules.

**Considerations**

This exit routine is for use under CICS only. Link OPTIQXIT into IDMSINTC with the external name OPTIQXIT.

**Note:** For more information on linking the IDMSINTC interface, see Create an IDMSINTC Interface Program (see page 176).

This exit is passed information on an IDMS request and where it is be routed by default. The exit may update the routing information in order to direct it to another location.
The first and third parameters passed to the exit contain information on the request: the address of the subschema control block and the caller's registers at the time the request was issued, respectively.

The second and fourth parameters contain routing information. The second parameter is the address of an OPTI structure generated either from parameters in the CICSOPT macro or from the SYSCTL file specified in the CICSOPT SYSCTL parameter. The fourth parameter may contain addresses of OPTI structures that were generated from additional SYSCTL files, if the MAXCVNO parameter was specified in the CICSOPT macro. The fourth parameter is always passed, even if no MAXCVNO was specified.

In order to direct the request unit to a different back-end CV, the exit must update the OPTI structure passed in the second parameter. It may use information passed in the fourth, fifth, and sixth parameters to determine which back-end CVs are available to service this request.

Registers at entry

- Register 1 points to a six-word parameter list described below
- Register 13 points to a savearea for use by the exit
- Register 14 points to the return address
- Register 15 points to the entry address

Parameters

- **Fullword 1**
  The address of the application program's subschema control block (SSC).

- **Fullword 2**
  The address of an OPTI structure describing where the request will be routed unless overridden by the exit. (The OPTI structure is described by DSECT #OPIDS.)

- **Fullword 3**
  The address of the savearea where the registers at entry to the interface were saved.

- **Fullword 4**
  The address of an array of fullwords whose contents are:

  - **Fullword 1**
    The MAXCVNO value specified in the CICSOPT macro. This value indicates the number of additional OPTI addresses present in this array.

  - **Fullwords 2-4**
    Three words that are available for use by the exit. The contents of these fields are preserved across invocations of the OPTIQXIT (and OPTIXIT if present). These three words can be used to retain information such as the last CV to which a request unit was routed. On the first call to the exit, the value of the first word will be the address of the default OPTI structure (see Fullword 5). The value of the remaining two words will be zero. These words are shared by both exits.
- **Fullword 5**
The address of the default OPTI structure. This is the OPTI structure generated from the CICSOPT macro or from the file whose DDNAME is specified by the SYSCTL parameter of the CICSOPT macro.

- **Remaining Fullwords**
  Occurs depending on MAXCVNO. Addresses of the OPTI structures generated for additional SYSCTL files. If the MAXCVNO value is 0, no additional addresses are present in the array.

- **Fullword 5**
The SQL command information block (DSECT #SQLCIB) that describes the statement being executed.

- **Fullword 6**
The contents of this field depend on the type of SQL statement being executed as indicated by the SQCIBCMD field of the command information block (Fullword 5):
  - If the SQL statement being executed by the application is a CONNECT (SQCIBCMD value 7), then Fullword 6 contains the address of the dictionary name to which the SQL session is connecting.
  - If the SQL statement being executed by the application is an EXECUTE IMMEDIATE (SQCIBCMD value 14), then Fullword 6 contains the address of the text string representing the statement to be executed. If the text represents a CONNECT statement, the dictionary name to which the session is connecting is also in the text string. (The length of the text string is in SQCIBCML.)
  - If the SQL statement being executed is neither a CONNECT nor an EXECUTE IMMEDIATE, Fullword 6 is zero.

**Return codes**
None.

**OPTIXIT (IDMSINTC)**

**Entry point**
OPTIXIT

**Link edit with**
IDMSINTC interface program.

This exit allows CICS transactions to modify the IDMSOPTI structure dynamically so that only the current task thread is affected by the changes.

IDMSINTC copies the static IDMSOPTI structure into dynamic storage and passes it to the user routine, which may alter it based on site-specific rules.

**Sample uses**
OPTIXIT can be used to alter the location where the request is processed dynamically.

Considerations

This exit routine is for use under CICS only. Link OPTIXIT into IDMSINTC with the external name OPTIXIT.

**Note:** For more information on linking the IDMSINTC interface, see Create an IDMSINTC Interface Program (see page 176).

Two-phase commit resynchronization requests require special attention. For more information, see OPTIXIT Considerations.

This exit is passed information on an IDMS request and where it is be routed by default. The exit may update the routing information in order to direct it to another location.

The first and third parameters passed to the exit contain information on the request: the address of the subschema control block and the caller's registers at the time the request was issued, respectively.

The second and fourth parameters contain routing information. The second parameter is the address of an OPTI structure generated either from parameters in the CICSOPT macro or from the SYSCTL file specified in the CICSOPT SYSCTL parameter. The fourth parameter may contain addresses of OPTI structures that were generated from additional SYSCTL files, if the MAXCVNO parameter was specified in the CICSOPT macro. The fourth parameter is always passed, even if no MAXCVNO was specified.

In order to direct the request unit to a different back-end CV, the exit must update the OPTI structure passed in the second parameter. It may use information passed in the fourth parameter to determine which back-end CVs are available.

**Registers at entry**

- Register 1 points to a four-word parameter list described below
- Register 13 points to a savearea for use by the exit
- Register 14 points to the return address
- Register 15 points to the entry address

**Parameters**

- **Fullword 1**
  The address of the application program’s subschema control block (SSC).

- **Fullword 2**
  The address of an OPTI structure describing where the request will be routed unless overridden by the exit. (The OPTI structure is described by DSECT #OPIDS.)

- **Fullword 3**
  The address of the savearea where the registers at entry to the interface were saved.
- **Fullword 4**
  The address of an array of fullwords whose contents are:
  - **Fullword 1**
    The MAXCVNO value specified in the CICSOPT macro. This value indicates the number of additional OPTI addresses present in this array.
  - **Fullwords 2-4**
    Three words that are available for use by the exit. The contents of these fields are preserved across invocations of the OPTIXIT (and the OPTIQXIT if present). These three words can be used to retain information such as the last CV to which a request unit was routed. On the first call to the exit, the value of the first word will be the address of the default OPTI structure (see Fullword 5). The value of the remaining two words will be zero. These words are shared by both exits.
  - **Fullword 5**
    The address of the default OPTI structure. This is the OPTI structure generated from the CICSOPT macro or from the file whose DDNAME is specified by the SYSCTL parameter of the CICSOPT macro.
  - **Remaining Fullwords**
    Occurs depending on MAXCVNO. Addresses of the OPTI structures generated for additional SYSCTL files. If the MAXCVNO value is 0, no additional addresses are present in the array.

**Return codes**

`one`.

**OPTIXIT Example**

The following is an example of the type of coding necessary to recognize and route resynchronization requests successfully:

```assembly
TITLE 'OPTIXIT - example of CICS OPTI exit needed for CICS RESYNC'
OPTIXIT CSECT
USING OPTIXIT,R15        ---> Base
B START                  Go process OPTI exit call
DROP R15
#MOPT CSECT=OPTIXIT,ENV=USER

START DS 0H
STM R14,R12,12(R13)     Save callers registers
LR R12,R15              Swap base to R12
USING OPTIXIT,R12       ---> Base
USING OPTXPLST,R1       --->; Parameter list
L R2,OPTXSSCA           Get address of Subschema Control
USING SSC,R2            --->; SSC
L R3,OPTXOPTA           Get address of OPTI structure
USING OPI,R3            --->; OPTI structure
CLC SSCPNAME,=C'INTCRSYN' Pseudo SSC for CICS RESYNC?
BE CICSRSYN             Yes, special process for CICS RESYNC
***********************************************************************
* perform normal OPTIXIT logic for real SSC
***********************************************************************
B RETURN                 Exit
CICSRSYN DS 0H
LA R5,OPINODE           Point at name of backend CV
LA R4,SYSLIST           Get table of known backend CVs
USING SYSTABLE,R4      --->; SYSTABLE
LOOP DS 0H
```
CLI SYSNAME,C'*'.  
Is this end of CV table ?

BE RETURN  
Yes, just exit

CLC SYSNAME,0(R5)  
Is this CV one of my CVs ?

BE MATCH  
Yes,&nbsp;we. have a match

LA R4,SYSTSIZE(R4)  
Bump to next CV in the table

B LOOP  
Keep looking for my CVs

MATCH DS 0H

MVC OPICVNUM,SYSCV#  
Update OPTI with CV number

MVC OPISVCNO,SYSSVC#  
Update OPTI with SVC number

B RETURN  
Exit

RETURN DS 0H

LM R14,R12,12(R13)  
Restore callers registers

BR R14  
Return to caller

DROP R2,R3,R4,R12  
Drop SSC, OPI, SYSTABLE, base

SYSLIST EQU *  
Backend CV table

DC C'SYSTEM71',AL1(71),AL1(173)  
CV 71 uses SVC number 173

DC C'SYSTEM72',AL1(72),AL1(176)  
CV 72 uses SVC number 176

DC C'SYSTEM73',AL1(73),AL1(176)  
CV 73 uses SVC number 176

DC C'SYSTEM74',AL1(74),AL1(173)  
CV 74 uses SVC number 173

DC C'*'  
End of backend CV table

SPACE 2  
Literal pool

LTORG ,  

OPTXPLST DSECT  
OPTI exit PLIST

OPTXSSCA DS A(0)  
A(SSC)

OPTXOPTA DS A(0)  
A(OPTI)

SPACE 2  

SYSTABLE DSECT  
Backend CV table dsect

SYSNAME DS CL8  
Backend CV node name

SYSCV# DS XL1  
Backend CV number

SYSSVC# DS XL1  
Backend CV SVC number

SYSTSIZE EQU *:SYSTABLE  
Size of one SYSTABLE entry

EJECT  

COPY #OPIDS  
OPTI dsect

COPY #SSCD5  
Subschema control dsect

END

OPTIXIT (IDMSINCTL)

Entry point

OPTIXIT

Link edit with

IDMSINCTL interface program

This exit allows CICS transactions to modify the IDMSOPTI structure dynamically so that only the current task thread is affected by the changes.

The IDMSINCTL interface copies the static IDMSOPTI structure into dynamic storage and passes it to the user routine, which may alter it based on site-specific rules.

Sample uses

OPTIXIT can be used to alter the location where the request is processed dynamically.

Considerations

To take advantage of this feature, you must write an assembler program and link it with your CICSOPT options module.
When creating an OPTIXIT module for IDMSINTL, consider the following:

- The module must have an entry point of OPTIXIT.
- The module is passed a different parameter list than the OPTIXIT for the IDMSINTC interface. You cannot use the same routine for both.

When called, the exit is passed three parameters. The first and third parameters contain information on the request: the address of the subschema control block and the caller's registers at the time the request was issued, respectively.

The second parameter passed to the exit is the address of an OPTI structure generated either from parameters in the CICSOPT macro or from the SYSCTL file specified in the CICSOPT SYSCTL parameter. To direct the request unit to a different back-end CV, the exit must update the OPTI structure passed in the second parameter.

Registers at entry to exit:

- Register 1 points to a three-word parameter list described next
- Register 13 points to a save area for use by the exit
- Register 14 points to the return location
- Register 15 points to the exit’s entry address

Parameters

- **Fullword 1**
  Defines the address of the application program's subschema control block (SSC).

- **Fullword 2**
  Defines the address of an OPTI structure describing where the request will be routed unless overridden by the exit.
  **Note:** The OPTI structure is described by DSECT #OPIDS.

- **Fullword 3**
  Defines the address of the savearea where the registers at entry to the interface were saved.

Return codes:

None

**Example OPTIXIT for the IDMSINTL Interface**

The following example of an OPTIXIT for the IDMSINTL interface forces all requests to be routed to the target system accessed through SVC 226 and CV number 173:

```
TITLE 'SAMPLE OPTIXIT FOR IDMSINTL'
*---------------------------------------------------------------
* SAMPLE OPTIXIT FOR USE WITH VERSION OF IDMSINTL THAT SUPPORTS
* AN OPTIXIT
*
* ON ENTRY:
* R1  --> ADDRESS OF PARM LIST DESCRIBED BY DSECT OPTXPLST
```
* R13 --> CALLER'S SAVE AREA
* R14 --> RETURN ADDRESS
* R15 --> THIS MODULE'S ENTRY POINT
* 
* TO INSTALL THIS EXIT, LINK THE ASSEMBLED MODULE WITH IDMSINTL BY
* ADDING AN INCLUDE STATEMENT TO YOUR NORMAL IDMSINTL LINK JOB.
* 
*-----------------------------------------------------------------

TESTEXIT CSECT
ENTRY OPTIXIT
USING OPTIXIT,12
USING OPTIXIT,12

OPTIXIT DS 0H
STM 14,12,12(13) SAVE CALLER'S REGISTERS
LR 12,15 SET LOCAL BASE
LA 15,SAVEAREA POINT TO MY SAVE AREA
ST 13,4(15) SA FORWARD CHAIN
ST 15,8(13) SA BACKWARD CHAIN
LR 13,15 SET SAVE AREA POINTER
USING OPTXPLST,1
L 2,OPTXOPTA
MVI OPICVNUM,226 SAMPLE HARD-CODED CV NUMBER
MVI OPTISVCNO,173 SAMPLE HARD-CODED SVC NUMBER
L 13,4(13) CALLER'S SAVE AREA
LM 14,12,12(13) RESTORE REGISTERS
XR 15,15 CLEAR RETURN CODE
BR 14 AND EXIT

SAVEAREA DC 18'0'
LTORG

OPTXPLST DSECT
OPTXSSCA DS A(0) A(16-CHARACTER SUBSCHEMA-CTRL
OPTXOPTA DS A(0) A(0) DSECT STORAGE)
OPTXSAVA DS A(0) A(INTL CALLER'S SAVEAREA)
COPY #0PIDS
END

Tools Exits

Tools exits are CA-supplied exits that are used by CA IDMS Tools products. All tools exit routines are loaded and managed independently from RHDCUXIT. The RHDCUXIT module does not need to be relinked if maintenance or release upgrades change one of the tools exit routines.

Hooked Exits

Hooked exits gain control before a CA IDMS nucleus module does its processing. Hooked exits are put into place by issuing a #HOOK macro from a numbered user exit or from a user mode program.

HOOK Macro

The #HOOK macro is used to designate a hooked exit that will gain control before an IDMS vector is given control. When zIIP is enabled, a #HOOK must be used to overlay IDMS vectors. A program that attempts to overlay these vectors without using the #HOOK macro while zIIP is enabled will receive a D003 abend.

- **IN**
  Creates a hook

- **VECTOR=**
  Name of vector to hook
WHEN=
Before

LOC=
Entry point of your processing routine

HOOKID=
Name of HOOKID

PLIST
Either =* for an inline PLIST or the address of some working storage to use as a plist.

Considerations
- A register cannot be specified for LOC if an inline PLIST is being used (PLIST=*=). If you are calling an address outside of the program issuing the #HOOK, you must use a register.
- The hooked exit must have its entry point created with #START MPMODE=CALLER.

Note: For more information on the #START macro, see Macros Required for DC/UCF Calling Conventions (see page 259).

- The program issuing the #HOOK must be in usermode and cannot be storage protected. If #HOOK is issued from a program with Storage Protect on, the program will abend with a D005.
- The program that is given control must be fully multitasking capable.
- User programs must save and return R0-R1.

#HOOK Return Codes
After completion of the #HOOK function, the value in register 15 indicates the outcome of the operation.

<table>
<thead>
<tr>
<th>Register Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>The vector has been hooked successfully.</td>
</tr>
<tr>
<td>X'16'</td>
<td>The vector is not a vector that can be hooked.</td>
</tr>
<tr>
<td>X'20'</td>
<td>The vector has already been hooked.</td>
</tr>
</tbody>
</table>

Vectors that cannot be hooked:

<table>
<thead>
<tr>
<th>Vector</th>
<th>Vector</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSACSELA</td>
<td>CSAXCLI</td>
<td>NVTAMBE</td>
</tr>
<tr>
<td>CSAOS00A</td>
<td>NVTDCERM</td>
<td>NVTDLBE1</td>
</tr>
<tr>
<td>CSAJXITA</td>
<td>NVTUSINT</td>
<td>NVTTAPRA</td>
</tr>
<tr>
<td>CSAURTNA</td>
<td>NVTUSTAE</td>
<td>NVTLRBKA</td>
</tr>
<tr>
<td>CSAEVALA</td>
<td>NVTSYSIN</td>
<td>NVTBRBKA</td>
</tr>
<tr>
<td>CSASNPA</td>
<td>NVTPLNVT</td>
<td>NVTHCHKI</td>
</tr>
</tbody>
</table>
CA IDMS - 19.0

<table>
<thead>
<tr>
<th>CSASWITR</th>
<th>NVTCTABA</th>
<th>***SASC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSAHOOK1</td>
<td>NVTRVIBA</td>
<td>NVTDBVAR</td>
</tr>
<tr>
<td>CSABMVTA</td>
<td>NVTUCFVC</td>
<td>NVTDYLTA</td>
</tr>
<tr>
<td>CSADBUG</td>
<td>NVTSRSVC</td>
<td>NVTSUBM</td>
</tr>
<tr>
<td>CSAULIO</td>
<td>NVTENVDA</td>
<td>NVTP58B</td>
</tr>
<tr>
<td>CSADSIRB</td>
<td>NVTUXITA</td>
<td>NVTDTSEH</td>
</tr>
<tr>
<td>CSALOAD1</td>
<td>NVTHCHKR</td>
<td>NVTDYLD2</td>
</tr>
<tr>
<td>CSALOAD2</td>
<td>NVTSCRN</td>
<td>NVTMSGM</td>
</tr>
<tr>
<td>CSAAIDMS</td>
<td>NVTDBLNK</td>
<td>NVTDTCM</td>
</tr>
<tr>
<td>CSLOCKM</td>
<td>NVTCFIMA</td>
<td>NVTTJSYN</td>
</tr>
<tr>
<td>CSADTOLA</td>
<td>NVTRRSA</td>
<td>NVTPLIPC</td>
</tr>
<tr>
<td>CSAHISTA</td>
<td>NVTDBIO3</td>
<td>NVTPLIAB</td>
</tr>
<tr>
<td>CSAUTIL</td>
<td>NVTDCMT</td>
<td>NVTDOLAA</td>
</tr>
<tr>
<td>CSADLIF</td>
<td>NVTSRTTA</td>
<td>NVTARBKA</td>
</tr>
<tr>
<td>CSADLRC</td>
<td>NVTSRTNA</td>
<td>NVTTKMRV</td>
</tr>
<tr>
<td>CSAAPPCC</td>
<td>NVTCMSG</td>
<td>NVTPRNTK</td>
</tr>
<tr>
<td>CSALRFA</td>
<td>NVTDSRVA</td>
<td>NVTJCLI</td>
</tr>
<tr>
<td>CSADSIRC</td>
<td>NVTRDDRV</td>
<td>NVTJSRVH</td>
</tr>
<tr>
<td>CSAQSR</td>
<td>NVTAEDTA</td>
<td></td>
</tr>
</tbody>
</table>

#HOOK Caller Example

⚠️ **Note:** In this example, the ISA size is 48.
Vector Override Program Example

VECPROG  TITLE 'VECPROG'
  #MOPT CSECT=VECPROG,ENV=SYS
  COPY #CSADS
  COPY #TCEDS

VECPROG  CSECT
VECPGEP1 #START MPMODE=CALLER
  LR    12,15
  USING VECPGEP1,R12
  USING CSA,R10
  USING TCE,R9

  *
  * VECTOR SPECIFIC OVERRIDE CODING HERE
  *
  VECTDONE DS 0H
  #RTN
END

Numbered Exits

Contents

- Calling Conventions for Numbered Exits (see page 258)
- Macros Required for DC/UCF Calling Conventions (see page 259)
  - MOPT Macro (see page 259)
  - MOPT Syntax (see page 259)
  - MOPT Parameter (see page 259)
  - START Macro (see page 259)
  - START Syntax (see page 260)
  - START Parameter (see page 260)
  - RTN Macro (see page 261)
  - RTN Syntax (see page 261)
  - RTN Parameter (see page 261)
  - RTN Example (see page 262)
  - GETSTK Macro (see page 262)
  - GETSTK Syntax (see page 262)
  - GETSTK Parameter (see page 262)
- User-invoked Numbered Exits (see page 263)
  - XIT exit Syntax (see page 263)
  - XIT exit Parameter (see page 263)
  - exit Examples (see page 263)
- Installing Numbered Exits in the System (see page 263)
  - Step 1 Code a DEFXIT Macro (see page 264)
  - DEFXIT Syntax (see page 265)
  - DEFXIT Parameters (see page 265)
  - Step 2 Assemble and Link Edit the RHDCUXIT Module (see page 266)
- System-invoked Numbered Exits (see page 268)
  - Exit 0 -- System Initialization Exit (see page 269)
- Exit 1 -- Signon Exit (see page 270)
- Exit 2 -- Signoff Exit (see page 270)
- Exit 4 -- New Task Exit (see page 270)
- Exit 5 -- Task Termination Exit I (see page 271)
- Exit 6 -- Task Termination Exit II (see page 271)
- Exit 7 -- Write-to-Log Exit (see page 271)
- Exit 8 -- Log Full Exit (see page 272)
- Exit 9 -- System Statistics Exit (see page 272)
- Exit 12 -- Terminal I/O Error Exit (see page 272)
- Exit 13 -- Shutdown Exit (see page 273)
- Exit 14 -- BIND RUN UNIT and READY AREA Exit (see page 273)
- BIND Parameters (see page 274)
- Exit 15 -- VIB Statistics Exit (see page 275)
- VIB Parameters (see page 275)
- Exit 16 -- Write Printer Exit (see page 275)
- Exit 17 -- Input Data Stream Exit (see page 276)
- Input Parameters (see page 276)
- Exit 18 -- Output Data Stream Exit (see page 277)
- Exit 18 Parameter (see page 278)
- Exit 19 -- Asynchronous Terminal Connection Exit (see page 279)
- Exit 20 -- Resource Limit Exit (see page 279)
- Exit 20 Parameters (see page 280)
- Exit 20 Examples (see page 281)
- Exit 21 -- SYSOUTL Report Termination Exit (see page 282)
- Exit 22 -- Report Security and Routing Exit (see page 283)
- Exit 23 -- Pre-BIND RUN UNIT Exit (see page 284)
- Exit 24 -- GET TIME Exit (see page 285)
- Exit 26 -- OLQ JCL exit (see page 285)
- Exit 27 -- Examining the ERE Extension Exit (see page 286)
- Exit 27 Examples (see page 287)
- Exit 28 -- Security Preprocessing Exit (see page 287)
- Exit 29 -- Security Postprocessing Exit (see page 288)
- Exit 30 -- Deadlock Victim Selection Exit (see page 289)
- Exit 31 -- Transaction Statistics Exit (see page 291)
- Exit 31 Parameters (see page 291)
- Exit 32 -- SYSOUTL Detail Record Exit (see page 291)
- Exit 33 -- Program Loader Exit (see page 292)
- Exit 33 Parameters (see page 292)
- Exit 34 -- Unqualified Dbkey FIND/OBTAIN Exit (see page 292)
- Exit 34 Parameters (see page 293)
- Exit 35 -- Stalled Task Information Exit (see page 294)
- Exit 36 -- Global Deadlock Victim Selection Exit (see page 294)
- Exit 36 Parameters (see page 295)
Numbered exits are user exits that you define by means of the RHDCUXIT module. RHDCUXIT is loaded at system startup as part of the nucleus and as part of the batch and non-TP monitor interfaces. Numbered exits are divided into two types, as described in the following table:

<table>
<thead>
<tr>
<th>Type of exit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System-invoked exits</td>
<td>System-invoked exits provide control over selected DC/UCF functions such as signon, signoff, and security checking DC/UCF determines the routine that calls the exit. These exits are invoked at a predefined logical point in a DC/UCF module. For example, the signon routine calls exit 1 (the signon exit). (Exits 0 through 255)</td>
</tr>
<tr>
<td>User-invoked exits</td>
<td>User-invoked exits can be invoked by a site-written Assembler program. For example, if you have added exit 256 to your system, an Assembler program at your site can call this exit by means of an #XIT statement. (Exits 256 and above)</td>
</tr>
</tbody>
</table>

**Flow of control**

When a numbered user exit routine is executed:

1. Control passes to the exit-routine entry point
2. The user exit routine is performed
3. Control returns to the calling module

**Execution mode of the exit routine**

When you code a numbered exit routine, you must consider the execution mode for the exit routine. Numbered exit routines can be defined to execute either in user or in system mode. Considerations for each mode are given in Installing Numbered Exits in the System.
The following sections describe:

- Calling conventions for all numbered exits
- The #MOPT, #START, #RTN, and #GETSTK macros
- Calling user-invoked exits
- Installing numbered user exits in the system
- System-invoked exits

### Calling Conventions for Numbered Exits

The routine that calls a numbered exit is determined by whether the exit is a system-defined or a user-defined exit.

The calling conventions that you use for a user exit routine determine the registers that are available to the exit routine. You can use either DC/UCF calling conventions or IBM calling conventions. Consult IBM documentation for information on IBM calling conventions.

The following table describes DC/UCF calling conventions and also gives general calling conventions that apply when you use either IBM or DC/UCF calling conventions. You use either these conventions or standard IBM conventions to call a given numbered exit routine. This table also describes general calling conventions that apply under either DC/UCF or IBM calling conventions.

<table>
<thead>
<tr>
<th>Calling conventions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC/UCF calling conventions</td>
<td>When you use these calling conventions, you must use the #START macro to begin the exit routine and the #RTN macro to end the exit routine. On entry to the exit routine, the following registers contain information: Register 13 contains the address of the next available entry in the TCE stack. The value in this register may only be changed through the use of CA IDMS macros, for example #GETSTK, described below. On XA machines, the TCE stack is located in XA storage. This means that the exit routine has to run in Amode 31. If it switches to Amode 24, the TCE stack is not addressable anymore. Therefore, all DB and DC calls must be done in Amode 31. Register 12 contains the base address. Register 10 contains the address of the common system area (CSA). The value in this register must not be modified. Register 9 contains the address of the task control element (TCE). The value in this register must not be modified.</td>
</tr>
<tr>
<td>General calling conventions (Apply under DC/UCF and under IBM calling conventions)</td>
<td>These calling conventions apply to all numbered exits. On entry to the numbered exit, register 1 points to a two-word parameter list: First word -- The address of a fullword containing the exit number. Second word -- Either the address of an exit-specific parameter list or 0 (zero) if no parameters are passed. On return from a system-invoked exit, register 15 contains the return code (if any). Only routines executed in system mode can process these Return codes.</td>
</tr>
</tbody>
</table>
Macros Required for DC/UCF Calling Conventions

When you use DC/UCF calling conventions to call a user exit, you must use these macros:

- **#MOPT** -- Required if a user exit is to run in system mode
- **#START** -- Use to begin the user exit routine
- **#RTN** -- Use to end a user exit routine and return control to the calling routine
- **#GETSTK** -- Use in a system mode exit to acquire storage from the TCE stack area; this can be useful in preserving reentrancy

**MOPT Macro**

The **#MOPT** macro sets up the options for the issuing module. The only code that should appear prior to **#MOPT** is 'TITLE', comments, or source macro definitions.

**MOPT Syntax**

```
label  #MOPT CSECT=csect-name,ENV=SYS USER
```

**MOPT Parameter**

- **CSECT=**
  Generates a named CSECT and a constant of the CSECT name. `Csect-name` specifies the name that will be generated for the CSECT statement and a 'DC CL8' of that name for identification.

- **ENV=**
  Indicates a DC/UCF system module (SYS) or a user module (USER). The macro sets certain globals for system modules to ensure proper calling sequence generation.

**START Macro**

The **#START** macro must be the first instruction in a user exit routine that uses DC/UCF calling conventions. Syntax and syntax rules for the **#START** routine are provided below, followed by considerations that apply at sites using multitasking support.

**Considerations**

When determining which MPMODE to assign, remember that:

- A system-mode exit routine that uses IBM calling conventions should always be assigned an MPMODE of DC.
- Any exit routine that uses DC/UCF calling conventions should be assigned an MPMODE consistent with the control blocks that the routine accesses.
Note: For detailed coding considerations for multitasking sites, see the section Extended Addressing and Multitasking (see page 298).

Examples

Establish an entry point with the name XTEP1:

XTEP1 #START

Start a numbered user exit routine without establishing an entry point:

XTEP4 #START INTERNAL

START Syntax

```
label #START   INTERNAL MPMODE=
```

START Parameter

- **INTERNAL**
  
  If omitted, generates an external entry point. The value coded in the label field of the macro is the name used for the entry point.

- **MPMODE**
  
  Applies only at multitasking sites.
  
  When included in a #START macro, the MPMODE clause specifies the MPMODE to be assigned to the exit routine. At a multitasking site, you must specify an MPMODE if the routine runs in system mode (ENV=SYS is specified in the #MOPT macro) and an external entry point is to be generated.

  - **ANY**
    
    The exit routine can assume any MPMODE.
    
    ANY is appropriate only for fully-reentrant routines that do not update storage associated with another task and that do not access control blocks that may be updated by another task. Typically, ANY mode is required for:
    
    - User exit 14
    - User exit 15
    - User exit 23
    - User exits 27, 28, and 29
    - User exit 31

  - **DB**
    
    The exit routine runs in DB mode. DB mode is appropriate for routines that reference system-wide database resources, such as system lock tables and currency tables.
- **DC**
  The exit routine runs in DC mode. DC mode is appropriate when the exit routine only references control blocks held by task executing the routine (for example, TCE, LTE, SSC, VIB, and so forth). It assumes the exit routine does not reference system-wide DB control blocks. Typically, DC mode is required for:
  - User exits 0 through 9
  - User exit 12
  - User exit 13
  - User exit 16
  - User exits 19 through 22
  - User exits 24, 25, and 26
  - User exit 30

- **CALLER**
  The exit routine can assume the MPMODE of the caller. CALLER means that the program does not do any specific DB- or DC-related activities. Therefore, the MPMODE does not need to be changed from that of the calling program.

- **DRIVER**
  The exit routine runs in DRIVER mode. All the CA IDMS line drivers (for example, VTAM and UCF) run in DRIVER mode. Exits that need to access driver-related control blocks should run in this mode.
  Typically, DRIVER mode is required for user exits 17, 18, and 32.

- **USER**
  The exit routine runs in USER mode. USER mode is appropriate for user programs with storage protection enabled; for example, user-invoked numbered exits.

**RTN Macro**

The `#RTN` macro terminates a routine and returns control to the calling routine. `#RTN` must be the last instruction executed in a user exit routine that uses DC/UCF calling conventions. The `#RTN` macro must execute in Amode 31.

**RTN Syntax**

```plaintext
label   #RTN
```

**RTN Parameter**

Inclusion of a `label` is optional.
RTN Example

In the following sample #RTN macro, the macro is labeled RTRN1:

RTRN1 #RTN

GETSTK Macro

The #GETSTK macro allows the issuing program to acquire a number of fullwords from the current stack area pointed to by Register 13. The #GETSTK macro must execute in Amode 31.

GETSTK Syntax

```
#GETSTK (register-number) (variable-name) =fullword-number
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>register-number</td>
<td>Is the register that contains the value of the number of fullwords.</td>
</tr>
<tr>
<td>variable-name</td>
<td>Specifies the symbolic name of a halfword or fullword field that contains the value of the number of fullwords.</td>
</tr>
<tr>
<td>fullword-number</td>
<td>Specifies the number of fullwords preceded by an equal sign (that is, specify =2 for 2 fullwords).</td>
</tr>
<tr>
<td>REG=register-number</td>
<td>Indicates the register in which DC/UCF will return the address of the fullwords requested. Registers 12, 13 and 14 are reserved. Register 11 is the default.</td>
</tr>
<tr>
<td>DBLWD=NO/YES</td>
<td>Specifies whether the stack address returned should be aligned (YES) or not aligned (NO) on a double word boundary.</td>
</tr>
<tr>
<td>CLEAR=NO/YES</td>
<td>Specifies whether the stack area should be initialized to nulls (YES) or not initialized (NO). NO is the default.</td>
</tr>
</tbody>
</table>
User-invoked Numbered Exits

When you write a user-invoked exit routine, you assign the routine a unique number. The number can be 256 or higher. Information about Return codes for user-invoked numbered exits is provided below, followed by information on calling user-invoked exits by means of the #XIT macro.

Return codes

Each site determines the meanings of Return codes passed back in register 15 to a user-invoked exit routine. However, Return codes 0 (zero) and 4 are reserved, and have the following meanings:

- 0 -- The exit routine did not encounter any errors.
- 4 -- The requested exit is not implemented at the site.

Site-written Assembler programs can call user-invoked numbered exits. To call a user-invoked exit, the Assembler program uses a #XIT macro.

XIT exit Syntax

```
►►──┬─────────┬─ #XIT exit-number ─┬────────────────────────────────────┬─────►◄
└─ label ─┘            └─ ,PARM=parameter-register-pointer ─┘
```

XIT exit Parameter

- **exit-number**
  Specifies the number of the #DEFXIT macro that defines the exit being called.

- **PARM**
  Specifies a parameter list (if any) to be passed to the exit routine. *Parameter-register-pointer* can be either a register pointing to the parameter list or the symbolic name of the parameter list.

exit Examples

Call exit 256 without passing any parameters:
```
#XIT 256
```

Call exit 256 and pass a parameter list in register 1:
```
#XIT 256,PARM=(R1)
```

Installing Numbered Exits in the System

To install a numbered user exit in the system:

1. Code the user exit routine based on information in this section.

2. Install the numbered exit by coding a #DEFXIT macro. Include the #DEFXIT macro in the system’s RHDCUXIT source module. Assemble and link edit the RHDCUXIT module. Optionally, link the exit routine itself with RHDCUXIT.
3. If the exit routine is not linked with RHDCUXIT, then define the exit routine to the system by using a system generation PROGRAM statement. You can temporarily define the program to the system by using the DCMT VARY PROGRAM command.

![Note: For more information on defining the exit routine to the system, see the Administrating section or the System Tasks and Operator Reference section.]

Steps 1 and 2 are discussed in the following sections.

**Step 1 Code a DEFXIT Macro**

The #DEFXIT macro specifies information on a numbered exit, including the name of the exit's entry point and user exit routine. All #DEFXIT macros for a system are included in the RHDCUXIT module for the system.

You must code a #DEFXIT macro for each numbered exit:

- Place #DEFXIT macros in numerical order in the RHDCUXIT module, according to the number of each associated exit.

- Include placeholder #DEFXIT macros for numbered exits that are not currently used. This is necessary because each #DEFXIT macro's position in the RHDCUXIT source module determines the numbered exit to which the macro applies.

For example, the sample #DEFXIT macros below install system-invoked exits 1 and 5 in the DC/UCF system. If the first placeholder #DEFXIT macro were deleted from the sample RHDCUXIT module, the module would add exits 0 and 4 to the DC/UCF system:

```
RHDCUXIT CSECT
.
.
.

#DEFXIT , EXIT 00
#DEFXIT MODE=SYSTEM,CALL=DC,EP=XSON,AMODE=ANY EXIT 01
#DEFXIT , EXIT 02
#DEFXIT , EXIT 03
#DEFXIT , EXIT 04

#DEFXIT MODE=SYSTEM,CALL=DC,EX=XTASK,AMODE=ANY EXIT 05
.
.
.
```

**Considerations**

The combination of #DEFXIT options that requires the least system overhead is:

`MODE=SYSTEM,CALL=DC,EP=entry-point-name`
The #DEFXIT options that you use determine whether the user exit routine will be executed in user mode or system mode. The table below lists considerations for exit routines written for execution in user or system mode.

- **System-mode exit routines**
  When you use DC/UCF macros within a system-mode exit routine, the RGSV parameter of the macro should be specified to ensure that register contents are not destroyed during execution.

### Execution Considerations

<table>
<thead>
<tr>
<th>Mode</th>
<th>Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>The routine can access only the storage pages associated with it unless storage protection has been disabled at the system or the program level. The exit routine: Runs as a program under the DC/UCF nucleus. Uses standard IBM calling conventions (except that no save area is passed in register 13). The exit is called by a #LINK statement. Cannot process register 15 Return codes.</td>
</tr>
<tr>
<td>System</td>
<td>The routine can access all storage pages in the region in which it executes. The exit routine: Uses either IBM or DC/UCF calling conventions: IBM conventions -- The routine is called by a BALR instruction. DC/UCF conventions -- The routine is called by a #CALL instruction. Must contain a #MOPT macro with ENV=SYS specified. Must save and restore any registers that are used by the exit. Any DC/UCF macros used must include a RGSV=(R2-R8) clause and ensure that R9=A (current TCE) and R10=A(CSA), when the macro is called; for example #GETSTG TYPE=(USER),LEN=R0, ADDR=(1),RGSV=(R2-R8) Can process register 15 Return codes.</td>
</tr>
</tbody>
</table>

### DEFXIT Syntax

```
#DEFXIT MODE=
```

#### DEFXIT Parameters

- **MODE=USER**
  Enables the exit routine to run in user mode. The routine can access only the storage pages associated with it unless storage protection has been disabled at the system or the program level.

**Note:** Storage protection is disabled at the system level by the NOPROTECT parameter of the SYSTEM statement and disabled at the program level by the NOPROTECT parameter of the PROGRAM statement.

- **NAME=**
  Identifies the name of the exit routine.
- **MODE=SYSTEM**
  Enables the exit routine to run in system mode. The routine can access all storage pages in the region in which it executes.

- **CALL**
  Specifies the calling conventions with which the exit routine is called:
  - **DC**
    The routine is called by means of a #CALL instruction, using DC/UCF calling conventions.
  - **IBM**
    The routine is called by means of a BALR instruction, using standard IBM calling conventions.

  **Note:** For more information on calling conventions, see *Calling Conventions for Numbered Exits* (see page 258).

- **EP=**
  Specifies the routine is called by the specified entry point.

- **AMODE=**
  Specifies the addressing mode for the exit routine. Valid options are:
  - **24** -- The exit routine executes in 24-bit addressing mode.
  - **ANY** (default) -- The exit routine executes in the same addressing mode as does the DC/UCF nucleus.

- **NAME=**
  Specifies the routine is loaded using the specified routine's name.

### #DEFXIT system calling methods

<table>
<thead>
<tr>
<th>Calling method</th>
<th>Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entry point (EP)</strong></td>
<td>The routine runs as part of the RHDCUXIT nucleus module. Therefore, you must link edit the routine with the RHDCUXIT module. Running the program in RHDCUXIT eliminates the overhead of using a program pool for the exit routine. This strategy is generally advisable for frequently called exits.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>The routine runs as a program under the DC/UCF nucleus. Therefore, you must define the routine to the DC/UCF system by using the system generation PROGRAM statement. At runtime, DC/UCF loads the exit routine into a program pool, as necessary. This strategy is often advisable for infrequently called exits.</td>
</tr>
</tbody>
</table>

---

**Step 2: Assemble and Link Edit the RHDCUXIT Module**

To make numbered exits available for use, assemble and link edit the RHDCUXIT module for the system. The RHDCUXIT module is loaded at system startup as part of the nucleus.
When link editing the RHDCUXIT module:

- Include in the link edit any exit routines for which the #DEFXIT macro specifies an entry-point name
- Specify that the RHDCUXIT module entry point is UXITEP1

**z/OS RHDCUXIT assembly and link edit**

**To create a RHDCUXIT load module**

1. Create a source module by modifying the sample RHDCUXIT source in CAGJSRC with appropriate #DEFXIT macros for your installation.

2. Save the source module in your custom source library.

3. Assemble and link it into your custom load library by executing the z/OS Assemble and Link-Edit JCL (see page 465).
   Substitute the name of your source member and insert the following binder statement:

   ```
   INCLUDE CUSTLIB(linked-exit) OPTIONAL
   INCLUDE CUSTLIB(linked-exit) OPTIONAL
   . . .
   SETOPT PARM(REUS=NONE)
   NAME RHDCUXIT(R)
   ```

   - **linked-exit**
     Specifies the name(s) of any user exit(s) for which the #DEFXIT macro contains the EP= parameter. The exit must have been previously compiled and linked into your custom loadlib.

**z/VSE RHDCUXIT assembly and link edit**

1. Assemble and catalog the object module using the sample JCL in z/VSE Assemble JCL (see page 476).
   Modify the JCL by substituting the following inplace of the Assembler input statements:

   ```
   PUNCH 'CATALOG rhdcuxit OBJ REPLACE=YES'
   END
   ```

   Add your RHDCUXIT source by modifying the sample source in RHDCUXIT.A in your IDMS sublibrary

2. Link the RHDCUXIT phase using the sample JCL in z/VSE Link JCL (see page 476).
   Modify the JCL by substituting the following statements in place of the Linkage editor control statements:

   ```
   PHASE RHDCUXIT,*
   INCLUDE rhdcuxit INCLUDE linked-exit OPTIONAL
   INCLUDE linked-exit OPTIONAL
   ENTRY UXITEP1
   ```

   rhdcuxit

   Specifies the name of your RHDCUXIT object module.

   linked-exit
Specifies the names of any user exits for which the #DEFXIT macro contains the EP= parameter. The exit must have been previously compiled and cataloged into your idms.custlib sublibrary.

*z/VM RHDCUXIT assembly and link edit*

GLOBAL MACLIB idmslib
FILEDEF TEXT DISK RHDCUXIT TEXT A
ASSEMBLE rhdcuxit.source (NODECK OBJECT

FILEDEF SYSLST PRINTER
FILEDEF SYSLMOD DISK idmslib LOADLIB a2 (RECFM V LRECL 1024 BLKSIZE 1024
LKED linkctl

Linkage editor control statements (linkctl):

INCLUDE RHDCUXIT
include exitnnep optional: for #DEFXITs with EP=exitnnep
ENTRY UXITEP1
NAME RHDCUXIT(R)

<table>
<thead>
<tr>
<th>rhdcuxit.source</th>
<th>filename of the file containing the RHDCUXIT source</th>
</tr>
</thead>
<tbody>
<tr>
<td>idmslib</td>
<td>filename of the CA IDMS MACLIB library</td>
</tr>
<tr>
<td>idmslib LOADLIB a2</td>
<td>file identifier of the CA IDMS LOADLIB library</td>
</tr>
<tr>
<td>linkctl</td>
<td>filename of the file containing the linkage editor control statements</td>
</tr>
</tbody>
</table>

**System-invoked Numbered Exits**

User exits 0 through 255 are reserved for system-invoked numbered exit routines. The currently-implemented exits are listed below and described individually after the table.

At runtime, a system-invoked numbered exit routine is called by a system module if the exit has been installed in the system by means of the #DEFXIT macro. For details, see Installing Numbered Exits in the System.

<table>
<thead>
<tr>
<th>Exit</th>
<th>Description of exit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit 0</td>
<td>System initialization exit</td>
</tr>
<tr>
<td>Exit 1</td>
<td>Signon exit</td>
</tr>
<tr>
<td>Exit 2</td>
<td>Signoff exit</td>
</tr>
<tr>
<td>Exit 4</td>
<td>New task exit</td>
</tr>
<tr>
<td>Exit 5</td>
<td>Task termination exit I (before statistics are written)</td>
</tr>
<tr>
<td>Exit 6</td>
<td>Task termination exit II (after statistics are written)</td>
</tr>
<tr>
<td>Exit 7</td>
<td>Write-to-log exit</td>
</tr>
<tr>
<td>Exit 8</td>
<td>Log full exit</td>
</tr>
<tr>
<td>Exit 9</td>
<td>System statistics exit</td>
</tr>
<tr>
<td>Exit 10</td>
<td>Reserved</td>
</tr>
<tr>
<td>Exit 11</td>
<td>Reserved</td>
</tr>
<tr>
<td>Exit 12</td>
<td>Terminal I/O error exit</td>
</tr>
<tr>
<td>Exit</td>
<td>Description of exit</td>
</tr>
<tr>
<td>------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td>Exit 13</td>
<td>Shutdown exit</td>
</tr>
<tr>
<td>Exit 14</td>
<td>BIND RUN UNIT and READY AREA exit</td>
</tr>
<tr>
<td>Exit 15</td>
<td>VIB statistics exit</td>
</tr>
<tr>
<td>Exit 16</td>
<td>Write printer exit</td>
</tr>
<tr>
<td>Exit 17</td>
<td>Input data stream exit</td>
</tr>
<tr>
<td>Exit 18</td>
<td>Output data stream exit</td>
</tr>
<tr>
<td>Exit 19</td>
<td>Asynchronous terminal connection exit</td>
</tr>
<tr>
<td>Exit 20</td>
<td>Resource limit exit</td>
</tr>
<tr>
<td>Exit 21</td>
<td>SYSOUTL report exit</td>
</tr>
<tr>
<td>Exit 22</td>
<td>Report security and routing exit</td>
</tr>
<tr>
<td>Exit 23</td>
<td>Pre-BIND RUN UNIT exit</td>
</tr>
<tr>
<td>Exit 24</td>
<td>GET TIME exit</td>
</tr>
<tr>
<td>Exit 26</td>
<td>OLQ JCL exit</td>
</tr>
<tr>
<td>Exit 27</td>
<td>ERE Extension Examiner exit</td>
</tr>
<tr>
<td>Exit 28</td>
<td>Security Preprocessing exit</td>
</tr>
<tr>
<td>Exit 29</td>
<td>Security Postprocessing exit</td>
</tr>
<tr>
<td>Exit 30</td>
<td>Victim selection for deadlock detection exit</td>
</tr>
<tr>
<td>Exit 31</td>
<td>Transaction statistics exit</td>
</tr>
<tr>
<td>Exit 32</td>
<td>SYSOUTL detail record</td>
</tr>
<tr>
<td>Exit 33</td>
<td>Program loader exit</td>
</tr>
<tr>
<td>Exit 34</td>
<td>Unqualified dbkey FIND/OBTAIN exit</td>
</tr>
<tr>
<td>Exit 35</td>
<td>Stalled task information exit</td>
</tr>
<tr>
<td>Exit 36</td>
<td>Global deadlock victim selection exit</td>
</tr>
<tr>
<td>Exit 37</td>
<td>Recovery wait exit</td>
</tr>
<tr>
<td>Exit 38</td>
<td>Quiesce area exit</td>
</tr>
<tr>
<td>Exit 39</td>
<td>SQL Syntax Collection exit</td>
</tr>
</tbody>
</table>

**Exit 0 -- System Initialization Exit**

This exit is called by RHDCMSTR immediately after internal run units are signed on and common work area (CWA) storage is initialized.

**Sample uses**

Exit 0 can be used to initialize the CWA with site-specific information.

**Parameters**

None.

**Return codes**
Exit 1 -- Signon Exit

This exit is called after the signon routine (RHDCSNON) has verified the user identifier and password and has built a signon element (SON).

Sample uses

Exit 1 can be used to display messages and to capture signon information whenever a user signs on to the DC/UCF system.

Considerations

This user exit routine must be written to execute in system mode. Additionally, the #DEFXIT macro that defines the routine must specify MODE=SYSTEM.

Parameters

One parameter is passed: the address of the SON.

Return codes

The following values can be returned in register 15:

- A return code of greater than 4 directs the calling routine to abort the signon request.
- A return code of 4 or less directs the calling routine to accept the signon request.

Exit 2 -- Signoff Exit

This exit is called before the central security system signs off a user.

Sample uses

Exit 2 is useful for billing purposes and to capture signon information when the user signs off.

Parameters

One parameter is passed: the address of the signon element (SON).

Return codes

None.

Exit 4 -- New Task Exit

This exit is called whenever a new task is first dispatched. The exit is called after the task is initialized but before control is passed to a processing program.

Sample uses
This exit can be used to monitor task usage.

Parameters

None.

Return codes

None.

Exit 5 -- Task Termination Exit I

This exit is called whenever a task terminates, before task statistics (if any) are written. When the exit is called, the VB50 has been freed. Task statistics must be obtained from the task statistics table.

Sample uses

This exit can be used to control the writing of task statistics.

Considerations

Accounting data of interest can be extracted from the #STLDS DSECT and processed, as appropriate. For example, the accounting data can be written to an output file in addition to being written to the DC/UCF log.

This user exit routine must be written to execute in system mode. Additionally, the #DEFXIT macro that defines the routine must specify MODE=SYSTEM.

Parameters

None.

Return codes

A return code greater than 4 prevents the system from writing task statistics to the log file.

Exit 6 -- Task Termination Exit II

This exit is called whenever a task terminates, after task statistics (if any) are written.

Parameters

None.

Return codes

None.

Exit 7 -- Write-to-Log Exit

This exit is called whenever a message line is retrieved from the message dictionary by a #WTL (WRITE LOG) request.
This exit is called after the message is received and before it is sent to its destination.

Sample uses
Exit 7 can be used to modify the text of a message before the message is sent to its destination.

Parameters
One parameter is passed: the address of the message (133 bytes), starting with the ASA control character.

Return codes
None.

Exit 8 -- Log Full Exit
This exit is called whenever the DC/UCF log area or file becomes full.

Sample uses
This exit can be used to submit an ARCHIVE LOG utility statement to the internal reader and offload the log area.

Parameters
None.

Return codes
None.

Exit 9 -- System Statistics Exit
This exit is called whenever RHDCSTTS is invoked to write system statistics:
- At DC/UCF shutdown
- At the statistics interval
- When a DCMT WRITE STATISTICS is issued

The exit is called before statistics are written.

Sample uses
This exit permits your site to monitor the writing of system statistics.

Parameters
None.

Return codes
None.

Exit 12 -- Terminal I/O Error Exit

This exit is called whenever a permanent terminal I/O error is returned to a task by a line driver module.

Sample uses
Exit 12 can be used to vary a physical terminal offline after a designated number of I/O errors.

Parameters
None.

Return codes
None.

Exit 13 -- Shutdown Exit

This exit is called whenever a shutdown request is issued. The exit is called before the system is terminated (that is, immediately before the database resource control task (RHDCDBRC) terminates).

This exit is not called when an abort request is issued.

Sample uses
Exit 13 can be used to record the status of the system at shutdown time.

Parameters
None.

Return codes
None.

Exit 14 -- BIND RUN UNIT and READY AREA Exit

This exit is called whenever a BIND RUN UNIT or READY AREA is performed. Exit 14 is invoked after exit 23.

Sample uses
Exit 14 can be used to perform security checking routines to determine whether the user is authorized to access the requested database or area.

Considerations
This user exit routine must be written to execute in system mode. Additionally, the #DEFXIT macro that defines the routine must specify MODE=SYSTEM.
Information required by the user exit routine is available as follows:

- For BIND requests, the address of the subschema name is contained in the SSCPARM3 field in the IDMS communications block (#SSCDS).
- For READY AREA requests, the address of the area name is contained in the SSCAREA field in the IDMS communications block.
- For any request, user information is available from the signon element (#SONDS) by issuing an ACCEPT USERID navigational DML statement (#ACCEPT TYPE=USERID in Assembler).
- For any request, security information is available from the signon element (#SONDS) by invoking the #SECHECK macro.

\textbf{Note:} For more information on the #SECHECK macro, see the \textit{Security Administering section}.

Return codes

The following values can be returned in register 15:

- A return code greater than 4 directs the calling routine to abort the BIND or READY.
- A return code of 4 or less directs the calling routine to continue processing the BIND or READY.

To indicate that an error has occurred, the exit routine must both set the return code (register 15) to greater than 4 \textit{and} set an error code in the SSCSTAT field of the IDMS communications block.

\textbf{BIND Parameters}

The following parameters are passed:

- A fullword, the value of which indicates the function to be performed:

  \begin{center}
  \begin{tabular}{|c|c|}
  \hline
  Value & Function \tabularnewline
  \hline
  36 & READY USAGE - MODE UPDATE \tabularnewline
  37 & READY USAGE - MODE RETRIEVAL \tabularnewline
  38 & READY USAGE - MODE PROTECTED UPDATE \tabularnewline
  39 & READY USAGE - MODE PROTECTED RETRIEVAL \tabularnewline
  40 & READY USAGE - MODE EXCLUSIVE RETRIEVAL \tabularnewline
  41 & READY USAGE - MODE EXCLUSIVE UPDATE \tabularnewline
  59 & BIND RUN UNIT \tabularnewline
  97 & No lock for transient isolation \tabularnewline
  \hline
  \end{tabular}
  \end{center}

- The address of the IDMS communications block (#SSCDS).
- The address of the DLIM control block, which contains the address of the external request element (ERE) if the request is external. This address can be used to access the batch accounting data in the ERE-SVC-EXTENSION if the SVC exit has been implemented to provide such data.
Exit 15 -- VIB Statistics Exit

This exit is called whenever system statistics and/or task statistics (if requested at system generation time) are written to the variable information block (VIB). This exit is called after the statistics are written but before the VIB is released.

Sample uses

This exit can be used to examine collected statistics.

Return codes

None.

VIB Parameters

The following parameters are passed:

- A fullword value:
  - 0 -- Task statistics are not being collected.
  - 1 -- Task statistics are being collected.
- The address of the VIB

Exit 16 -- Write Printer Exit

This exit is called by the #PRINT (WRITE PRINTER) request handler at the first print request for a report.

Sample uses

Exit 16 can be used to examine and/or modify the report request block (RRB).

Considerations

This user exit routine must be written to execute in system mode. Additionally, the #DEFXIT macro that defines the routine must specify MODE=SYSTEM.

Parameters

One parameter is passed: The address of the RBB associated with the requesting task.

Return codes

None.
Exit 17 -- Input Data Stream Exit

This exit is called by the I/O request handler after the check for a read has been posted, the completion code has been checked, and line editing has been performed.

Sample uses

Exit 17 can be used to examine and/or alter the contents of the input data stream.

Considerations

If you need to increase the size of the data stream, create a new input buffer. To acquire storage for building buffers, use the get-buffer routine whose address is passed in the fourth word in the parameter list.

Invoke the get-buffer routine by using the following instruction:

```
BALR R14, R15
```

In this BALR instruction, register 15 contains the address of the get-buffer routine. Register 1 contains the address of the parameter list.

All registers are preserved by the get-buffer routine. The #IBHDS DSECT provides a map of the buffer header. The id of the task's physical terminal (PTERM) can be found in the PTEID field of the physical terminal element (PTE). The PTE layout is given by the #PTEDS DSECT.

An exit routine that creates new buffers can be:

- Written to execute in system mode. In this case, the #DEFXIT macro that adds the routine to the system must specify MODE=SYSTEM.

- Written to execute in user mode. The user exit routine must run with storage protection disabled. The #DEFXIT macro that adds the routine to the system must specify MODE=USER and must call the routine by name.

If you create a new input buffer, make sure that the exit 17 routine does not free the old buffer.

Make sure that the exit 17 routine returns values as follows:

- The first parameter must return the address of the first input buffer.

- The second parameter must return the total length of the input data stream.

Return codes

None.

Input Parameters

The parameter lists for the exit routine and for the get-buffer routine are each given below.

The exit 17 routine receives and can return changed values by means of the following five-fullword parameter list:
The address of the first input buffer. If no buffer exists, the address is zeros.

The total length of the input data stream. This includes all buffers.

The address of the task’s physical terminal element (PTE).

The address of a get-buffer routine.

A fullword in which the first byte contains:

- X’80’ -- End of parameter list.
- X’40’ -- The buffer contains the SNA functional management header (for lines defined with TYPE=VTAMLU).
- X’20’ -- The physical terminal expects a 3270-type data stream (for lines defined with TYPE=VTAMLU).

The get-buffer routine receives and returns the following fifteen-fullword parameter list that you must build:

- The address of the buffer on return
- The length of buffer needed
- The address of the task’s physical terminal element (PTE)
- A twelve-fullword work area

**Exit 18 -- Output Data Stream Exit**

This exit is called by the I/O request handler before the line driver is posted to perform an output operation. The exit is invoked after compaction is performed (if applicable).

**Sample uses**

Exit 18 can be used to examine and/or alter the contents of the output data stream.

**Considerations**

If the exit routine alters the contents of the data stream, you must return the length of the altered data stream in the second parameter.

You can build buffers by using the get-buffer routine whose address is passed in the fourth word in the parameter list. All registers are preserved by the get-buffer routine.

Invoke the get-buffer routine by using the following instruction:

```
BALR R14, R15
```

In this BALR instruction, register 15 contains the address of the get-buffer routine. Register 1 contains the address of the parameter list.
The id of the task's physical terminal (PTERM) can be found in the PTEID field of the physical terminal element (PTE). The PTE layout is given by the #PTEDS DSECT.

An exit routine that builds buffers can be:

- Written to execute in system mode. In this case, the #DEFXIT macro that adds the routine to the system must specify MODE=SYSTEM.

- Written to execute in user mode. The program must run with storage protection disabled. The #DEFXIT macro that adds the routine to the system must specify MODE=USER and call the routine by name.

If you create a new input buffer, make sure that the exit 18 routine returns values as follows:

- The first parameter must return the address of the first buffer.
- The second parameter must return the total length of the output data stream.

**Return codes**

None.

**Exit 18 Parameter**

The parameter lists for the exit routine and for the get-buffer routine are each given below.

The exit 18 routine receives and can return changed values by means of the five-fullword parameter list:

- The address of the output buffer. This parameter contains zeros if no buffer exists.
- The length of the output data stream.
- The address of the task's physical terminal element (PTE).
- The address of a get-buffer routine.
- A fullword in which the first byte contains:
  - X'80' -- End of parameter list.
  - X'40' -- The buffer contains the SNA functional management header (for lines defined with TYPE=VTAMLU).
  - X'20' -- The task's physical terminal (PTERM) expects a 3270-type data stream (for lines defined with TYPE=VTAMLU).

If the buffer contains converted data that originally was in 3270-type format, make sure the fifth parameter does not return X'20'. (X'20' causes 3270 WRITE commands to be prefixed to the buffer.)

The get-buffer routine receives and returns the following fifteen-fullword parameter list that you must build:
The address of the buffer on return.

The length of buffer needed.

The address of the task’s physical terminal element (PTE). This value is required.

A twelve-fullword work area.

Exit 19 -- Asynchronous Terminal Connection Exit

This exit is called by the asynchronous (TTY and 2741) terminal drivers. This exit is called after the line driver has validated the terminal type code and has copied the 1-to-8-character identifier into the PTE6NAME field of the physical terminal element (PTE).

Sample uses

Exit 19 can be used to examine and/or alter the contents of the PTE and its device-dependent extension (for example, the exit can treat the character string in the PTE6NAME field as a system password).

Considerations

This user exit routine must be written to execute in system mode. Additionally, the #DEFXIT macro that defines the routine must specify MODE=SYSTEM.

Parameters

One parameter is passed: the address of the physical terminal element (PTE).

Return codes

The following values can be returned in register 15:

- A return code greater than 0 directs DC/UCF to continue the normal connection process.

- A return code less than 0 directs DC/UCF to disconnect the terminal and to re-enable the line for connection.

- A return code of 0 indicates that register 1 points to an EBCDIC message and register 0 contains the message length. After truncating the message (if longer than 100 bytes) and translating it into the appropriate transmission code, DC/UCF writes the message to the terminal and waits for the user to reenter a terminal identifier.

Exit 20 -- Resource Limit Exit

This exit is called by RHDCLIMIT whenever a resource limit is exceeded by a task. The exit routine is invoked before the request that exceeds the limit is satisfied.

Sample uses

This exit can be used to examine other limits or to increase the limit that has been exceeded.
Exit 20 permits the user to continue or abend a task based on the type of task. For example, you may choose to continue a payroll task that has exceeded the storage limit while abending any other task that exceeds the same limit.

Considerations

DC/UCF enforces limits on task resource usage as described below:

- When a task is initiated at runtime, the system constructs a resource limit block (RLB) for the task. The RLB specifies the specified limit for each resource type. Additionally, the RLB points to an extension in which the system maintains a count of the number of times each limit has been exceeded.

- When the task issues a request involving a resource for which a limit has been established, the system checks the current value for the resource in the task statistics table (TST) for the task.

- When the resource request would exceed the limit specified in the RLB, (the value in the TST would exceed the limit in the RLB):
  - If exit 20 is defined, the system either continues or abends the task, as directed by the exit routine. Exit 20 can be used to perform various functions, such as examining other limits, writing messages to the system log, and altering the limits in the RLB for the task.
  - If exit 20 is not defined, the system abends the task.

DC/UCF continues to check limits when the task resumes execution. Therefore, if the task is to continue, the exit routine must alter the limit in the resource limit block (RLB).

Return codes

The following values can be returned in register 15:

- 0 directs the calling routine to continue the task without issuing any messages.
- 4 directs the calling routine to abort the task after issuing message DC244003.
- 8 directs the calling routine to continue the task after issuing message DC244002.

Exit 20 Parameters

The following parameters are passed:

- A fullword, the last byte of which indicates the resource that caused the exit to be invoked, as follows:
  
<table>
<thead>
<tr>
<th>Value</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'01'</td>
<td>Storage</td>
</tr>
<tr>
<td>X'08'</td>
<td>Database I/O operations</td>
</tr>
<tr>
<td>X'10'</td>
<td>Database key locks</td>
</tr>
<tr>
<td>X'20'</td>
<td>System service calls</td>
</tr>
</tbody>
</table>
The offset from the beginning of the resource limit block (RLB) of the limit that has been exceeded.
The address of the RLB is provided by the TCERLBA field of the task control element (TCE). The RLBEA field of the RLB points to an extension in which the system maintains a count of the number of times each limit has been exceeded.

Exit 20 Examples

The following sample user exit is called by RHDCLIMIT whenever a resource limit is exceeded. The exit is invoked by modifying RHDCUXIT for exit 20 in either of the following ways:

```
#DEFXIT MODE=SYSTEM, CALL=DC, EP=UX20EP1  EXIT 20
#DEFXIT MODE=SYSTEM, CALL=DC, NAME=RHDCUX20  EXIT 20
```

Sample user exit 20 -- resource limit exit

```
***********************************************************************
*         THIS SAMPLE USER EXIT 20 HANDLES RESOURCE LIMIT OVERFLOWS. *
*         IT IS INVOKED BY RHDCLIMIT WHENEVER A LIMIT IS FOUND TO BE *
*         EXCEEDED. THE CONDITIONS HANDLED ARE AS FOLLOWS:            *
*         STORAGE LIMITS: ALL TASKS ARE ALLOWED TO CONTINUE            *
*         LOCK  LIMITS: ALL TASKS ARE ABENDED                         *
*         CALL   LIMITS: EXTERNAL REQUEST UNITS (ERUS) ARE ABENDED    *
*         DBIO   LIMITS: ALL TASKS ARE ABENDED                         *
*                                                                *
***********************************************************************
UX20      TITLE 'RHDCUX20 -- EXAMPLE RESOURCE LIMIT EXIT'
* RHDCUX20 EP=UX20EP1
EJECT
#MOPT CSECT=RHDCUX20, ENV=SYS
COPY   #CSADS
COPY   #LTEDS
COPY   #RLBDS
COPY   #TCEDS
COPY   #TSTDS

*****************************************************************************
*     WORK AREA DSECT FOR LIMIT ROUTINE.                                   *
*****************************************************************************
WORKD   DSECT
SYSPLIST DS   12A
WORKDL EQU (*-WORKD+3)/4 LENGTH IN WORDS.
   SPACE
XIT     DSECT
XIT00   DS   H        FILLER
XITCD   DS   0H       RESOURCE CODE
   DS   X        (RESERVED AT THE MOMENT)
XITSTG  #FLAG X'01'     CHECK STORAGE LIMIT
XITDBIO #FLAG X'08'     CHECK DBIO LIMIT
XITLOCK #FLAG X'10'     CHECK DB LOCKING LIMIT
XITCALL #FLAG X'20'     CHECK SERVICE CALL LIMIT
XITFLG1 DS   X        *
XITOF   DS   F        LIMIT OFFSET IN TASK RLB
EJECT
*****************************************************************************
*     RHDCUX20 - USER EXIT TO HANDLE RESOURCE LIMITS                       *
*     ON ENTRY R1 WILL POINT TO A TWO WORD PLIST, FORMATTED                *
*     AS FollowS:                                                         *
```
Exit 21 -- SYSOUTL Report Termination Exit

Exit 21 is called by the SYSOUTL line driver (RHDCD07Q) after the last line of each print report has been written to the spool file.

A sample exit routine for exit 21 on z/VM systems is provided on the integration tape used to install CA IDMS.

⚠️ Note: For details on implementing the sample routine at z/VM sites, see the Installing section -- z/VM.

Sample uses

A DC system can include one or more lines for spooled output. Such lines are defined at system generation time with a type of SYSOUTL. Reports sent to the spool file for a SYSOUTL line are not printed until the file is closed. The spool file is closed:
• When the SYSOUTL line driver is disabled, for example, at DC system shutdown
• When a CP command to close the file is issued from the DC operator’s console

At z/VM installations, exit 21 can be used to print spooled output on completion of each report. If appropriate, the exit routine can reroute the spool file (for example, through RSCS) before closing the file. A sample user exit routine for exit 21 can be found in the Installing section -- z/VM.

Considerations

At a z/VM site with guest operating systems, define the SYSOUTL line by:

1. Making a direct unit assignment in the guest operating system
2. Using a FILEDEF in z/VM to associate that unit with a z/VM print device

If the SEND command sends messages to the SYSOUT printer, the report element for each message will have a report name (RPERPTNM) of binary zeros. The exit routine can then:

1. Determine the appropriate RSCS destination for the report
2. Issue the TAG, SPOOL CLOSE, and SPOOL OPEN commands to provide support for VM/RSCS operations

This user exit routine must be written to execute in system mode. Additionally, the #DEFXIT macro that adds the exit routine to the system must specify MODE=SYSTEM.

Parameters

One parameter is passed: the address of the physical terminal element (PTE) associated with the SYSOUTL line. The PTE points to the logical terminal element (LTE), which points to the report element (RPE). The RPE contains such information as the report identifier and the number of copies to be printed.

Return codes

None.

Exit 22 -- Report Security and Routing Exit

This exit can be used to store user-related information to be used at print time. Print module RHDCPRNT calls exit 22 when the user makes an initial print request, after RHDCPRNT has assigned a unique report identifier to the print request.

Sample uses

During this exit’s processing, you can write a queue record that contains security or routing information. You can modify the RHDCBANR routine so that, at print time, it extracts the saved information and uses it to write the report header. This header can contain the appropriate security or routing information.

Considerations
This exit routine must be written to execute in system mode. Additionally, the #DEFXIT macro that adds the exit to the system must specify MODE=SYSTEM.

Parameters

The following parameters are passed:

- The address of the report print element (RPE)
- The address of the report request block (RRB)

Return codes

None.

Exit 23 -- Pre-BIND RUN UNIT Exit

User exit 23 is called before a BIND RUN UNIT is performed. Exit 23 is invoked before exit 14. Exit 23 has addressability to the task control element (TCE).

Sample uses

User exit 23 can be used to override any specifications passed in the parameter list, as specified below.

For example, exit 23 can be used to determine the node on which a database resides and the database name based on the subschema name. If a database changes location, you need not recompile the program with the new node or database name on the BIND. The exit can change these parameters instead.

Considerations

This exit routine must be written to execute in system mode. Additionally, the #DEFXIT macro that adds the exit to the system must specify MODE=SYSTEM.

Note: If you invoke EXIT 23 under CICS, the exit must be linked with RHDCUXIT and the #DEFEXIT macro must specify EP=entry-point-name.

Parameters

Register 1 points to two fullwords that contain the exit number and the address of a 40-byte data area that holds the values coded in the BIND RUN UNIT request. If the user has set any database or dictionary overrides (such as by using DCUF SET DICTNAME), the overrides are reflected in the data area. The format of the data area is:

- Subschema name (8 bytes)
- Database node (8 bytes)
- Database name (8 bytes)
- Dictionary node (8 bytes)
Dictionary name (8 bytes)

The address of the task control element (TCE) is passed through register 9. The exit can change these values. These changes are then reflected in the IDMS communications block.

Return codes

None.

Exit 24 -- GET TIME Exit

Exit 24 provides a means to return a different value for GET TIME requests than the current value returned by the operating system.

Except for SQL, Exit 24 is called whenever a DC GETIME is issued to obtain the time and date from the operating system.

Parameters

To use this feature, you must write a user exit routine with the following attributes:

- System mode
- No storage protect
- Amode 31

On entry to the exit:

- R1 = Address of two (2) word parm list
- +0 = Address of fullword containing exit number
- +4 = Address of a doubleword date and time

The doubleword date and time contains the packed date, and the binary absolute time in 0000 seconds. This is identical to the values normally found in CSATIME and CSADATE fields. The date is expressed as OnYYDDDF (for 1900 n is 0; for 2000 n is 1).

To set a different time to be returned, simply store a date and time in the doubleword pointed at by R1.

Return codes

Return codes are ignored.

Exit 26 -- OLQ JCL exit

User Exit 26 is called when OLQ has built the JCL line.

Sample uses

Exit 26 can be used to alter the JCL and/or to output it.
Considerations

None.

Parameters

A single parameter is passed to exit 26. It contains the address of the JCL record.

Return codes

The following Return codes can be returned in register 15:

- A return code of 0 indicates that the exit output the record itself. OLQ itself will continue with the next record if any.
- A return code of 4 indicates that the exit examined and maybe altered the JCL record, but wants OLQ to output it.
- Any other return code directs OLQ to abort.

Exit 27 -- Examining the ERE Extension Exit

User Exit 27 is called after DC/UCF receives a new external request for services, but before the system performs any processing for the request. For example, the exit is called before DC/UCF processes:

- A request for database services from a batch program
- A request to initiate a new UCF session
- A request for services from another node in the DC/UCF communications network provided that the connection is an SVC connection.

Sample uses

You can examine information in the ERE or in the ERE extension. You can also specify the user id which will be used to sign on to the DC/UCF system.

Considerations

The following considerations apply to user exit 27:

- In the #DEFXIT macro, user exit 27 must be defined using MODE=SYSTEM.
- Exit 27 is invoked before exits 14 and 23.
- When you return from Exit 27, register 1 should point to a field containing an 18-character user identifier, right-padded with blanks. Otherwise, default user identifier PUBLIC may be substituted when ERUS signon takes place.

Parameters

A single parameter is passed to exit 27. It contains the address of the ERE.
Return codes

The following Return codes can be returned in register 15:

- A return code of 0 if register 1 points to a valid user identifier.
- A return code of 4 if register 1 does not point to a valid user id.

Exit 27 Examples

```
... L R2,4(R1) Get address of ERE
    USING ERE,R2
L R3,ERESEA Get address of ESE
    USING ESE,R3
LH R4,ESESPNOOF Get offset to ERE extension.
    USING EREXSOF,R4
LA R4,ERE(R4) Get address of ERE extension.
    USING EREXSOF,R4
...
* Test for user id modification criteria
  CLI EREXUSER,C User id blank of hex zero
  BH NOSET No.
* Other user id modification criteria:
  ...
* We have determined that this exit is to specify the
  * signon user id.
  LA R1,DEFAULTU Set R1 for caller
  XR R15,R15 Clear R15 return code
  B RETURN
NOSET DS 0H
RETURN DS 0H #RTN
DEFAULTU DC CL16'SITEUSERID'
#RTN
COPY #EREDS
COPY #ESEDS
COPY #UCFDS
```

Exit 28 -- Security Preprocessing Exit

Exit 28 allows you to examine all security requests, including user signon and signoff, before the request is processed by the CA IDMS centralized security facility.

Exit 28 is called after the security system has validated the function code, but before it performs any other processing for the security request. When you set a flag (SRBFAB) in the Security Request Block (SRB), the exit can request that access be denied.

⚠️ Note: For more information on the Security Request Block (#SECRB DSECT), see the CA IDMS Security Administering section.

Sample uses

You can examine the information in the SRB before CA IDMS processes the security request. For example, you might alter the required authorities based on site specific requirements.
Considerations

Exit 28 cannot force the centralized security facility to allow the requested access. If the exit does not abort the request by setting SRBXFAB, the centralized security facility will process the request normally.

This exit routine must be written to execute in SYSTEM MODE. The #DEFXIT macro that adds the exit routine to the system must:

- Specify MODE=SYSTEM
- Call the routine using either DC or IBM calling conventions
- Call the routine by entry point

Parameters

The following parameters are passed:

- The address of the SRB
- The length of the SRB

Return codes

None.

Exit 29 -- Security Postprocessing Exit

Exit 29 allows you to examine all security requests, including user signon and signoff, after the request is processed by the CA IDMS centralized security facility.

Exit 29 is called after the centralized security facility has completed processing for a security request. When you set a flag (SRBXFAB) in the security request block (SRB), the exit can request that access be denied.

Sample uses

You can use exit 29 to log security violations or to implement site specific security enforcement requirements.

Considerations

- Exit 29 cannot override a security violation.
- If your site uses a security package external to DC/UCF (for example, CA ACF2 for z/OS), DC/UCF calls Exit 29 after signon to the external security package and again after signon to DC/UCF. External signon occurs if any resource, including signon itself, is externally secured. DC/UCF always attempts an internal signon. Because DC/UCF calls the exit two times, you can customize internal signon to use information from the external signon. For example, CA ACF2 for z/OS users may want to move a Release 10.2 bit map from the CA ACF2 for z/OS logon ID (LID) record to the SONSECTY field in the signon control block.
DC/UCF uses the contents of the SONSECTY field as the default activity bit map for the user. To do this, set the SRBXSGN flag in the SIGNON function of the security request block (SRB). If signon is secured internally, then you must:

- Define the user in the USER catalog
- Turn off the SRBXSGN flag in the SIGNON function of the security request block (SRB).
- This exit routine must be written to execute in SYSTEM MODE. The #DEFXIT macro that adds the exit routine to the system must:
  - Specify MODE=SYSTEM
  - Call the routine using either DC or IBM calling conventions
  - Call the routine by entry point

More Information

- For information on the signon control block (#SONDS DSECT), see the DSECT Reference section.
- For information on the security request block (#SECRB DSECT), see the Security Administering section.

Parameters

The following parameters are passed:

- The address of the SRB
- The length of the SRB

Return codes

None.

Exit 30 -- Deadlock Victim Selection Exit

User exit 30 lets users implement a site-specific deadlock victim selection algorithm. User exit 30 is called after the deadlock detector has scanned all tasks in the system that are waiting and has eliminated those that cannot be involved in a deadlock situation.

After identifying waiting tasks that are involved in a deadlock, the deadlock detector processes these tasks in pairs. A victim is chosen from the initial pair of tasks and then compared to the next task involved in the deadlock. This process of examining each task in the deadlock and selecting a new victim continues until all deadlocked tasks have been examined. The last victim chosen is then terminated and the process is repeated until no more deadlocks exist.

If exit 30 is not installed, the deadlock detector chooses the victim from each pair of tasks examined as follows:

- If one of the tasks is COND=NONE and the other task is COND=DEAD, the COND=DEAD task will be selected as the victim.
If both tasks have the same COND specification, the task having the lower priority will be chosen.

Sample uses

User exit 30 lets users bypass the default victim selection logic and gain more control over how deadlocks are resolved. For example, some users may prefer that victims be selected:

- As external request units when deadlocks between online and external request units are detected.
- Based on what an application does rather than by its task priority.

You can apply exit 30 to a subset of applications or databases, by using Return codes (described below). This feature allows the exit to selectively apply its logic to pairs of deadlocked tasks based on criteria determined by the implementer.

Considerations

Exit 30 is passed the address of the Dispatch Control Elements (DCE) representing the two tasks from which to choose a victim. The DCE is used by the system to control how tasks are dispatched and contains the task dispatching priority and the address of the Task Control Element (TCE).

As with all user exits, minimize the exposure of this exit to system control blocks since these control blocks can change from release to release.

This exit routine must be written to execute in system mode. Additionally, the #DEFXIT macro that adds the exit to the system must specify:

- MODE=SYSTEM
- AMODE=ANY

Additionally, this exit should be reentrant and should be coded to handle 24-bit and 31-bit addresses.

Parameters

Exit 30 is passed the address of a two fullword parameter list consisting of the:

- DCE address of the first deadlocked task
- DCE address of the second deadlocked task

Return codes

Before it terminates, exit 30 should put the DCE address of the selected victim in register 15. If you want the exit to use the default victim selection logic in the deadlock detector, rather than a site-specific logic:

- Return a value of 0 in register 15 to use the default victim selection logic and continue to call exit 30.
- Return a value of 4 in register 15 to use the default victim selection logic and discontinue call to exit 30.
Exit 31 -- Transaction Statistics Exit

This exit is called by the Transaction Manager whenever statistics are written from the transaction block. This exit is called after statistics have been written but before the transaction block has been released. Note that these transaction statistics are not those maintained by the system and are not those maintained as the result of the BIND, ACCEPT, or END TRANSACTION STATISTICS DML verbs.

Sample uses

This exit can be used to examine collected statistics.

Return codes

None.

Exit 31 Parameters

The following parameters are passed:

- A fullword value:
  - 0 indicates task statistics are not being collected
  - 1 indicates task statistics are being collected
- The address of the transaction block (TBK)

Exit 32 -- SYSOUTL Detail Record Exit

This exit is called by the SYSOUTL line driver (RHDCD07Q) when a record is about to PUT to the output file.

Sample uses

This exit can be used to:

- Examine and modify every record
- Insert records using the available DCB address

Considerations

This user exit routine must be written to execute in SYSTEM mode. Additionally, the #DEFXIT macro that adds the exit routine to the system must specify MODE=SYSTEM.

Parameters

The following parameters are passed:

- The output record address
- The DCB address
Return codes
None.

Exit 33 -- Program Loader Exit
This exit is called after a program load request has been completed.

Sample use
This exit can be used to extract program name and type to use for statistics.

Considerations
This exit routine must be written to execute in SYSTEM mode. The #DEIXIT macro that adds the exit routine to the system must:

- Specify MODE=SYSTEM
- Call the routine using either DC or IBM calling conventions
- Call the routine by entry point

Exit 33 Parameters
The following parameters are passed:

- Fullword 1 -- Sets a type code of 1 to indicate a PDE
- Fullword 2 -- Address of the PDE for the program loaded
- Fullword 3 -- Unused
- Fullword 4 -- Unused

Exit 34 -- Unqualified Dbkey FIND/OBTAIN Exit
Exit 34 helps identify and correct applications that may require modification to function correctly when the Mixed Page Group BINDS ALLOWED feature is enabled.

Exit 34 is provided to allow the runtime detection of unqualified dbkey retrievals when "Mixed Page Group Binds Allowed" is specified for the DBNAME. The exit may be used to display messages on the console and/or abend the task. Furthermore, this exit can provide the correct page group and radix value for the passed dbkey to enable the application to run correctly without requiring source changes.

Exit 34 is invoked by IDMSDBMS whenever a rununit issues a FIND DB-KEY or OBTAIN DB-KEY verb with no record name specified. However, alteration of a rununit's current page group or page radix will not be honored unless mixed page group support is fully implemented by specifying 'Mixed Page Group Binds Allowed' for the DBNAME. The exit is not invoked for rununits accessing the dictionary or catalog.
It is possible that an unqualified FIND DB-KEY or OBTAIN DB-KEY command may not retrieve the desired record when Mixed Page Group Binds are allowed. When this feature is enabled, IDMSDBMS will use the current page and radix value for the dbkey. If the last DML operation referenced a page group other than the one desired, then the wrong record may be retrieved. If the unqualified retrieval is the first DML operation for the rununit, then there is no current of page group and a 0326 status code is returned.

For the long term, it is recommended that applications with unqualified FIND DB-KEY or OBTAIN DB-KEY commands be enhanced to specify a record name or exploit the PAGE-INFO parameter rather than use this exit.

Considerations

You must write the Exit 34 routine to execute in SYSTEM mode. The #DEFXIT macro that adds the exit routine to a system must:

- Specify MODE=SYSTEM
- Call the routine using either DC or IBM calling conventions
- Call the routine by entry point

Return codes

Return codes are ignored.

Exit 34 Parameters

The following parameters are passed:

- Fullword 1 -- Address of a five-word save area. The area will remain consistent for the life of the rununit. Data stored here will remain until the rununit finishes.
- Fullword 2 -- Address of the IB50 Control Block. (See macro #FIBDS.)
- Fullword 3 -- Address of the Subschema Control Block. (See copy book #SSC120.) Fields: SSCPGRUP and SSCRADIX will contain the current page group and radix value for the rununit (these may be changed). Upon return from the exit IDMSDBMS will make the changed values current for the rununit.

Note: Sample exit RHDCUX34 has been supplied as part of CA IDMS. This program will display a message on the console when called. It also contains examples of abending a task and of modifying the current page group. To use the sample exit as written, RHDCUXIT must define a #DEFEXIT as follows:

```
#DEFEXIT MODE=SYSTEM,CALL=DC,EP=UX34EP1
```

and RHDCUX34 must be linked with RHDCUXIT.
Exit 35 -- Stalled Task Information Exit

This exit is called to gather information on a stalled task for use during deadlock victim selection. It will be invoked only in a data sharing environment during global deadlock management.

This exit can be used to pass information to user exit 36 to assist in selecting a victim task in a global deadlock situation.

Considerations

This exit is passed the addresses of two control blocks. The first control block contains information on the stalled task, the second is an output area in which the exit can place information for use by exit 36 when selecting a deadlock victim. The address of the stalled task's Dispatch Control Element (DCE) is passed in the first control block. This can be used to locate other task-related control blocks.

Addresses should not be stored in the 32-byte output area, since exit 36 may execute on a different system from that on which exit 35 is executing.

This exit must be written to execute in SYSTEM mode. The #DEFXIT macro that adds the exit routine to the system must specify:

- MODE=SYSTEM
- AMODE=ANY

Additionally, this exit should be reentrant and should be coded to handle 31-bit addresses.

Parameters

The following parameters are passed:

- Fullword 1 - The address of an area described by DSECT #X35PL
- Fullword 2 - The address of a 32-byte output area in which the exit may save information to be passed to User Exit 36

Return codes

Set register 15 to 0.

Exit 36 -- Global Deadlock Victim Selection Exit

This exit is called in a global deadlock situation to select a task to be cancelled. If exit 36 is not installed, a global deadlock will be resolved by choosing the task with the lowest priority that was initiated last. When comparing two tasks, CA IDMS will always select as a victim a task that specified COND=DEAD over one that specified COND=NONE.

Exit 36 allows site-control over which task is chosen as a victim in a global deadlock situation.

Considerations
This exit is passed the addresses of two sets of control blocks, each of which is associated with a stalled task. Within each set, the first control block contains information supplied by CA IDMS about the stalled task, the second is an area that potentially contains information passed from exit 35.

Exit 36 must be written to execute in SYSTEM mode. The #DEFXIT macro that adds the exit routine to the system must specify:

- **MODE=SYSTEM**
- **AMODE=ANY**

Additionally, this exit should be reentrant and should be coded to handle 31-bit addresses.

**Return codes**

Set register 15 to the address of the control block described by DSECT #X36PL for the task that is to be selected as the victim. If the exit does not select a victim, it should set the return code to 0.

**Exit 36 Parameters**

The following parameters are passed:

- **Fullword 1** - The address of a control block described by DSECT #X36PL that describes the first of two deadlocked tasks.

- **Fullword 2** - The address of a 32-byte area containing information passed from user exit 35 for the first deadlocked task.

- **Fullword 3** - The address of a control block described by DSECT #X36PL that describes the second of two deadlocked tasks.

- **Fullword 4** - The address of a 32-byte area containing information passed from user exit 35 for the second deadlocked task.

**Exit 37 -- Recovery Wait Exit**

This exit is called when a task is about to wait on a global resource that requires recovery by a failed member of a data sharing group.

This exit can override the current recovery wait setting for the system.

**Considerations**

The exit can specify whether the task should be aborted or whether it should wait for the failing member to be recovered. If the task should wait, the exit specifies the length of time the task should wait.

This exit must be written to execute in SYSTEM mode. The #DEFXIT macro that adds the exit routine to the system must specify:

- **MODE=SYSTEM**
AMODE=ANY

Additionally, this exit should be reentrant and should be coded to handle 31-bit addresses.

Parameters

The following parameter is passed:

Fullword 1 - The address of an area described by DSECT #X37PL

Return codes

By setting an appropriate return code in Register 15, the exit can specify what action CA IDMS should take with regard to the task. The possible choices are:

- 0 - the task should wait. Register 0 must contain the amount of time that the task is permitted to wait. Valid values for register 0 are:
  - 1 through 32767 specifying the number of seconds that the task is permitted to wait
  - -1 indicating that the task should wait indefinitely
  - 0 indicating that the task should not wait. A value of 0 is equivalent to a return code value of 8.
- 4 - the system’s recovery wait setting determines what action will be taken
- 8 - the task should be aborted

Exit 38 -- Quiesce Area Exit

This exit is invoked when a quiesce point has been reached in the processing of a DCMT QUIESCE command. Its purpose is to allow additional site-specific actions to be taken in response to the quiesce.

Considerations

This exit is passed the quiesce identifier, an indication of what is being quiesced and a list of files and their data set names that are impacted by the quiesce. With this information, the exit can take additional action, such as constructing JCL or loading predefined JCL for a batch job to be submitted through the internal reader.

Through Return codes, the exit can direct IDMS to terminate or continue the quiesce operation, or proceed as specified in the original DCMT QUIESCE command.

Return codes

By setting an appropriate return code, the exit can specify what action IDMS should take with regard to the quiesce operation. The possible choices are:

- 0, to continue or terminate the quiesce operation as specified on the DCMT QUIESCE command
- 8, to continue the quiesce operation, overriding the option specified on the DCMT QUIESCE command
- 12, to terminate the quiesce operation, overriding the option specified on the DCMT QUIESCE command

Exit 38 Parameters

The exit is passed a single parameter described by DSECT #X38PL. This structure contains the following information:

- The nodename on which the quiesce command originated
- The quiesce operation identifier
- An indication of what is being quiesced (area, segment or DBNAME) and its name
- An array of file entries containing the following information for each file involved in the quiesce:
  - File name (<segment-name>.<area-name>)
  - VOLSER
  - DDNAME
  - Data set name

Exit 39 -- SQL Syntax Collecting Exit

Exit 39 allows you to collect audit information from the system about the usage of SQL.

Exit 39 is called after the centralized security facility completes processing of a security request and after calling exit 29. Exit 39 is only called when the following conditions are met:

- SAVE_SQL_SYNTAX=ON is specified in the SYSIDMS parameter file.
- The security request is issued for an SQL check, that is when the function in the Security Request Block (SRBFUNC) has a value of 12 (SQL Schema check).

Sample uses

You can use exit 39 to log the SQL syntax that is used by BCF, OCF, ODBC, and JDBC users.

Considerations

1. To obtain the storage containing the SQL syntax, follow these steps:
   - Get the address of the sLTE from the TCECLTE field at TCE+X'94'.
   - Get the address of the SQL syntax buffer from field LTESVSQL at LTE+X'150'.
The layout of the SQL syntax buffer is:

- 0 - 3: Eyecatcher 'SQL#'
- 4 - 8: Fullword containing the length of the SQL syntax buffer.
- 9 -12: Fullword containing the length of the current SQL command.
- 13-nnn: Current SQL command without the default or customized delimiter.

2. Execute exit 39 in SYSTEM MODE. For the #DEFXIT macro that adds exit 39 to the system, it is necessary to:

- Specify MODE=SYSTEM.
- Call the routine using either DC or IBM calling conventions.
- Call the routine by entry point.

⚠️ **Note:** For more information on the security request block (#SECRB DSECT), see the CA IDMS Security Administering section.

**Parameters**

The following parameters are passed:

- The address of the SRB.
- The length of the SRB.

**Return codes**

None.

**Extended Addressing and Multitasking**

- Extended Addressing Considerations (see page 298)
- Multitasking Support (see page 301)

**Extended Addressing Considerations**

**Contents**

- AMODE and RMODE Assignment (see page 299)
- Program Pool Usage (see page 299)
- Storage Pool Usage (see page 301)
Treatment of Dynamically-built Control Blocks (see page 301)

When you run a DC/UCF system, program execution is affected by the presence of multiple program and/or storage pools. This section discusses the aspects of program execution.

AMODE and RMODE Assignment

Each program must be assigned the following:

- An addressing mode (AMODE)
- A residency mode (RMODE)

The following table summarizes ways to assign AMODE and RMODE to programs or modules.

<table>
<thead>
<tr>
<th>Module location</th>
<th>Location mode specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load library</td>
<td>Specify AMODE and RMODE according to instructions provided in your operating system dependent documentation on the linker/loader provided by your operating system vendor.</td>
</tr>
<tr>
<td>Data dictionary (DDLDCL area)</td>
<td>Specify AMODE and RMODE in the IDD LOAD MODULE statement as follows: MOD LOAD MODULE module-name. AMODE=ANY RMODE=ANY. The default for both AMODE and RMODE is ANY.</td>
</tr>
</tbody>
</table>

Program Pool Usage

Each program defined to the DC/UCF system is assigned a primary program pool based on the program’s reentrancy and residency mode and on the types of pools defined. All IDMS-created programs (for example, maps, dialogs, subschema, and code and edit tables) have AMODE 31 and RMODE ANY. A program assigned a 31-bit primary pool is also assigned a 24-bit alternate pool.

Program loading sequence

The table below shows the primary and alternate pools assigned to each type of program. When a task invokes a program, DC/UCF loads the program in a pool or waits for enough space to load the program:

- DC/UCF tries to load the program as follows:
1. Into its primary pool

2. Into its alternate pool if the primary pool is short on space and an alternate pool is assigned to the program

- DC/UCF waits for space in either pool when both the primary pool and the alternate pool are short on space.

Additionally, if a program assigned a 31-bit primary pool is invoked by a task for which the location is BELOW, the program is loaded into its 24-bit alternate pool. DC/UCF does not try to load the program into a 31-bit pool. This ensures the program's addressability by other programs running under the same task.

**Primary and alternate program pool assignments**

DC/UCF always tries to load a program into the program's primary pool. If the primary pool is short on space, DC/UCF loads the program in the alternate pool. The existence of an alternate pool depends on the characteristics of the program and what pools have been defined in addition to the required 24-bit program pool.

<table>
<thead>
<tr>
<th>Program characteristics</th>
<th>Conditions</th>
<th>Primary pool</th>
<th>Alternate pool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonreentrant or quasi-reentrant (RMODE=24)</td>
<td>--</td>
<td>24-bit program pool</td>
<td>--</td>
</tr>
<tr>
<td>Reentrant (RMODE=24)</td>
<td>24-bit reentrant pool defined</td>
<td>24-bit reentrant pool defined</td>
<td>24-bit reentrant pool</td>
</tr>
<tr>
<td>Reentrant (RMODE=24)</td>
<td>24-bit reentrant pool not defined</td>
<td>24-bit reentrant pool</td>
<td>--</td>
</tr>
<tr>
<td>Nonreentrant or quasi-reentrant (RMODE=ANY)</td>
<td>31-bit program pool defined</td>
<td>31-bit program pool</td>
<td>24-bit program pool</td>
</tr>
<tr>
<td>Nonreentrant or quasi-reentrant (RMODE=ANY)</td>
<td>31-bit program pool not defined</td>
<td>24-bit program pool</td>
<td>--</td>
</tr>
<tr>
<td>Reentrant (RMODE=ANY)</td>
<td>31-bit and 24-bit reentrant pools defined</td>
<td>31-bit program pool</td>
<td>24-bit reentrant pool</td>
</tr>
<tr>
<td>Reentrant (RMODE=ANY)</td>
<td>31-bit reentrant pool defined; 24-bit reentrant pool not defined</td>
<td>24-bit reentrant pool</td>
<td>24-bit program pool</td>
</tr>
<tr>
<td>Reentrant (RMODE=ANY)</td>
<td>31-bit reentrant pool defined; 24-bit reentrant pool not defined</td>
<td>24-bit reentrant pool</td>
<td>--</td>
</tr>
<tr>
<td>Reentrant (RMODE=ANY)</td>
<td>31-bit reentrant pool defined; 24-bit reentrant pool not defined</td>
<td>24-bit reentrant pool</td>
<td>--</td>
</tr>
</tbody>
</table>
Storage Pool Usage

The location of storage acquired during the execution of a program depends on the following factors:

- For user mode storage requests:
  - The location (BELOW or ANY) of the task that invoked the program
  - The specification (BELOW or ANY) in the LOC parameter of the \#GETSTG statement used to request the storage
  - The pools defined at DC/UCF system generation time

- For system mode storage requests, default is 31-bit storage

- For all storage requests, space availability in the eligible pools and the types of storage that the pools accommodate

XA storage pool 255, which you define on the SYSTEM system generation statement, is reserved for system storage only. It is the XA equivalent of storage pool 0.

Storage is allocated as follows:

<table>
<thead>
<tr>
<th>TASK</th>
<th>Storage Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(BELOW)</td>
<td>24-bit storage allocated</td>
</tr>
<tr>
<td>TASK(ANY); #GETSTG</td>
<td>24-bit storage allocated</td>
</tr>
<tr>
<td>(BELOW)</td>
<td>24-bit storage allocated</td>
</tr>
<tr>
<td>TASK(ANY); #GETSTG</td>
<td>If any 31-bit pools contain the requested type of storage, 31-bit storage allocated, if defined and available</td>
</tr>
<tr>
<td>(ANY)</td>
<td>Otherwise, 24-bit storage allocated</td>
</tr>
</tbody>
</table>

Treatment of Dynamically-built Control Blocks

Dynamically-built control blocks, for example program definition elements (PDEs) and task definition elements (TDEs) can be built in an XA storage pool. Therefore, if you do not define PDEs at system generation using either the UNDEFINED PROGRAM COUNT parameter of the SYSTEM statement or the PROGRAM statement itself, DC/UCF will build the PDEs dynamically in an XA storage pool.

DC/UCF may allocate from an XA storage pool all system-allocated storage that will not be passed back to a user program regardless of the program’s residency mode. For control blocks that must reside below the 16-megabyte line, DC/UCF explicitly requests storage below the line.

Multitasking Support

Contents
CA IDMS is a multithreaded system that supports multiple diverse units of work called tasks. In the past, these tasks shared one operating system subtask. With multitasking support, several CA IDMS tasks can execute concurrently and each task uses a different predefined operating system subtask.

Multitasking support can be implemented only at z/OS sites that have either uniprocessors or multiprocessors.

Multitasking support can provide for better CPU utilization in a heavily used multi-processor environment; for example, shops with dyadic or quadratic processors that implement multitasking can run closer to full capacity. For example, user task A may be issuing database operations (such as, an OBTAIN) and can be executing at the same time as a DC/UCF user task B activity (such as, a GET STORAGE request). Meanwhile, user task C is executing its user coded routines.

The following table contrasts nonmultitasking with z/OS multitasking environments:

<table>
<thead>
<tr>
<th>Nonmultitasking</th>
<th>Multitasking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks execute consecutively.</td>
<td>Tasks execute simultaneously.</td>
</tr>
<tr>
<td>No distinction is made between the types of tasks performed.</td>
<td>Tasks are subdivided into families based on the control blocks updated.</td>
</tr>
</tbody>
</table>

How DC Handles Concurrency

In a multitasking environment, the system must be able to allow multiple tasks to update the same information. This is achieved in the following ways, depending on the update type and amount:

- Under very specific circumstances, updating a single field with length 4 or 8 bytes can be done with a single instruction (CS or CDS). No control block locking nor task serialization is needed.

- If the update consists of multiple field updates, without calls to different components, control block locking is used. This means that a lock is acquired, the updates are made and the lock is released. The time that the lock is held is very short, minimizing the chance on lock collisions.

- In all other circumstances, task serialization is used. This means that the system must control the order in which tasks are executed, so that no two tasks simultaneously do the same work.

Most of the executable coding in CA IDMS runs without task serialization.
Task serialization

To achieve this control, DC/UCF uses task serialization, a technique that forces tasks to take turns based on a predefined class or family. Code running in a mode type of ANY is not serialized. Much of the executable code in CA IDMS runs in mode type ANY.

Work modes

Work is divided into families of executable code called modes. CA IDMS uses a symmetrical design that allows each mode of work to be scheduled on any available subtask, allowing the full capacity of the hardware to be used most efficiently.

CA IDMS multitasking supports the following types of work modes:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Code description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANY</td>
<td>Assigned to all user code and system code that doesn’t require serialization. ANY is for programs that will not update any storage associated with another program. Therefore, it is safe for a program assigned MPMODE ANY to run simultaneously with other programs.</td>
</tr>
<tr>
<td>DB</td>
<td>Assigned to modules that perform database activities such as record processing, record locking, and concurrency control. Use the #START macro to assign this MPMODE to user exits performing database functions.</td>
</tr>
<tr>
<td>DC</td>
<td>Assigned to all non-DB modules, including most user-written programs running without storage protection. Use the #START macro to assign this MPMODE to user exits performing DC functions. (CA ADS dialogs and COBOL programs can be assigned MPMODE=ANY).</td>
</tr>
<tr>
<td>DRIV ER</td>
<td>Assigned to all CA IDMS line drivers (for example, VTAM and UCF). Use the #START macro to assign this MPMODE to user exits that need to access driver-related control blocks.</td>
</tr>
<tr>
<td>LOAD ER</td>
<td>Assigned to program loading routines; user programs cannot be assigned this mode.</td>
</tr>
<tr>
<td>USER</td>
<td>Assigned to user programs executing with storage protection enabled and to VS COBOL programs that are non-protected.</td>
</tr>
</tbody>
</table>

CALLER command

CALLER is a command that assigns the mode of the calling program to fully-reentrant programs. CALLER means that the called program accepts the MPMODE of the calling program and assumes full responsibility for all MPMODE serialization rules. Therefore, the MPMODE does not need to be changed from that of the calling program. User programs cannot be assigned MPMODE=CALLER.

You can specify at system generation that an application will run with an MPMODE of ANY or SYSTEM (the default). SYSTEM means that DC/UCF determines the MPMODE at runtime.

Example

Suppose four user tasks are executing concurrently. Tasks 1 and 4 are executing CA ADS user code in ANY mode. Task 2 is executing DB mode and task 3 is executing DC mode. All four tasks execute on individual subtasks concurrently, assuming your machine has four available processors:
How to Implement z/OS Multitasking

You implement multitasking support in system startup execution JCL through the use of the PARM parameter of the EXEC statement for the startup routine.

You use the PARM parameter to do the following:

- Enable multitasking support
- Optionally specify the number of operating system subtasks to use

Note: For more information on the PARM parameter, see Specifying Runtime Options.
Coding Considerations

Assembler programs need to exercise caution when using shared storage. This is because, in multitasking systems, this storage can change while the program executes, even if the program does not request any DC/UCF services. These programs must follow site-dependent mechanisms and standards that keep user programs from simultaneously updating shared storage.

Assembler programs should use #ENQ to acquire shared storage and #DEQ to release the storage. The #ENQ and #DEQ statements are described in the CA IDMS DML Reference section for Assembler.

System mode user-exit routines

System mode user-exit routines at a DC/UCF site must be recompiled if they are already installed to support multitasking execution. The #START macro for each system mode exit routine must declare an MPMODE. Use the following modes:

<table>
<thead>
<tr>
<th>Mode</th>
<th>For these user exits</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANY</td>
<td>Exits 14, 15, 23, 27, 28, 29, 31, 37</td>
</tr>
<tr>
<td>DC</td>
<td>Exits: 0 - 9, 12, 13, 16, 19 - 22, 24 - 26, 30, 35 - 36</td>
</tr>
<tr>
<td>DB</td>
<td>Exit 38</td>
</tr>
<tr>
<td>DRIVER</td>
<td>Exits 17, 18, 32</td>
</tr>
<tr>
<td>None</td>
<td>Exits 10, 11</td>
</tr>
</tbody>
</table>

System mode exit routines with IBM calling conventions are invoked by DC/UCF in CALLER mode. Since these exits are called by a routine with an MPMODE of DC, such user exits must be able to execute in DC mode. To handle this, you can begin with NOOP (X'47000004').

⚠️ **Note:** For more information on user-exit routines and the #START macro, see User Exits.

Programs that execute with storage protection

At runtime, DC/UCF always assigns an MPMODE of USER to a user-mode program that is invoked with storage protection regardless of the system generation MPMODE specification. For a program to execute with storage protection enabled, storage protection must be enabled both at the system level and at the program level.
Monitoring Multitasking Performance

Under z/OS, DC/UCF supports up to 99 subtasks. At heavily used systems, it may be helpful to run DC /UCF with more subtasks available than there are processors in the system. This strategy can be used to provide for an overlap in subtask processing during page fault processing.

Task waits may occur at runtime when serialization has to take place.

Available monitoring information

You can obtain information on multitasking performance using the following tools:

<table>
<thead>
<tr>
<th>Command</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCMT DISPLAY SUBTASK</td>
<td>Use to observe the frequency with which subtasks are being dispatched. CA IDMS multitasking will only dispatch an additional subtask if the work load dictates it. Select fewer subtasks if you want to restrict the amount of CPU CA IDMS consumes; increase the number of subtasks if you observe all the subtasks in a busy condition.</td>
</tr>
<tr>
<td>DCMT DISPLAY MPMODE</td>
<td>Displays, for each mode, how often the mode was requested and how often a mode had to wait if another subtask was executing in that mode. The highest throughput is achieved when the request counts for ANY mode is very high and the values in the Wait Counts are very low.</td>
</tr>
<tr>
<td>DCMT DISPLAY MT Q DEPTH</td>
<td>Displays the number of DC tasks that have to be queued for system services before an additional subtask is woken up. This number is called the multitasking queue depth (MT Q DEPTH). Its default value is 2. DC/UCF only restarts (wakes up) a system subtask when the MT queue depth is exceeded. This strategy is defined to reduce overhead.</td>
</tr>
<tr>
<td>DCMT VARY MT Q DEPTH</td>
<td>Allows dynamic changes to the MT queue depth. Specifying a low value will cause more usage of subtasks. A too-low value will cause subtasks to wake up and go back to sleep again without having done any work because the queue was already emptied by one or more other subtasks. A too-high value will cause most or all work to be processed by only one subtask, that is, it disables multitasking. The optimum value for the MT queue depth is dependent on factors outside the control of DC (other work on the CPUs; operating system dispatcher parameters; paging rate; ...). Therefore, it is advised to experiment with the value and watch the results. The value must be in a range of 0 to 255; however, the advised value is in a range of 0 to 9. The default is 2.</td>
</tr>
<tr>
<td>CA IDMS Performance Monitor</td>
<td>Use the online realtime and interval monitors to observe the impact of multitasking on your runtime production system. For example, you may need to adjust your DC/UCF system to accommodate the higher level of multiprogramming introduced by multitasking. The interval monitor also provides the time spent waiting on mode locks. Use PMARPT02, the Task Summary Report, to measure changes in overall response time before and after implementing multitasking.</td>
</tr>
</tbody>
</table>
Multitasking Information in Dumps

In this section, you are shown how to find multitasking specific information in dumps. For these tasks, you should use an operating system dump, instead of a snap dump. With a snap dump, information has already changed due to the snap processing as follows:

- The current task could have switched to another operating system subtask.
- The MPMODE of the task has changed to the MPMODE of RHDCSNAP.
- All locks were released. For more information, see Determining the Locks Held by a Task.

The following control blocks, which are described in the CA IDMS DSECT Reference section, are referenced as follows:

- CSA: #CSADS
- ILE: #ILEDS
- SCA: #SCADS
- TCE: #TCEDS

Locating the SCA

z/OS System

At a z/OS multitasking site, the subtask control area (SCA) of the TCB active at the time of the dump can be located using the following procedure:

1. Find location X'10'. The value at this location is the address of the communication vector table (CVT).
2. At address X'00' into the CVT is the address of the TCB list.
3. At X'04' into the TCB list is the address of the currently active TCB.

![Note: The address of the first ready TCB is located at X'1C' into the ASCB.]

4. At X'70' into the TCB is the address of the first TCB save area.
5. At X'00' into the save area is the address of the SCA assigned to the TCB.

SCA Information

- Once you located the SCA, you must determine if this SCA is associated with the abending task. To do this, check flag SCAACTV (SCA + X'1C', bit X'80'). If the flag is set, the SCA is active.
SCACURTK (SCA + X'08') is the address of the TCE the active SCA is executing.

The location of all SCAs can be determined by walking the chain of SCAs, beginning at the DC/UCF common system area (CSA). The address of the SCA area is found in CSASCAA (CSA + X'518').

The address of the next SCA is computed by adding SCALEN (half word at (SCA + X'0E') to the address of the current SCA. The number of SCA’s can be found in CSASCA# (half word at CSA + X'480')

On occasion, there will be a dummy SCA for the ticker subtask. This occurs when the ticker task has to wait for a lock for the internal control element (ICE) chain. In this case, the SCA is used as working storage.

**Determining the Task Using a Subtask**

To determine the DC task using a particular subtask, perform the appropriate operation:

- If you have the task’s SCA address, look at SCACURTK (offset X'08' in the SCA). This field contains a pointer to the active TCE, if there is one.
- If you have the task’s TCE address, look at TCESCAA (offset X'134' in the TCE). The value at this location points to the SCA when the task is active.

**Determining the Task's MPMODE**

To determine the MPMODE a task was running in, use an operating system dump. Examine the half word value of the TCEMPMOD field (TCE + X'13C'). Possible values are the following:

- X'0000' -- MPMODE=ANY
- X'0004' -- MPMODE=DC
- X'0008' -- MPMODE=DB
- X'000C' -- MPMODE=USER
- X'0010' -- MPMODE=LOADER
- X'0014' -- MPMODE=DRIVER

**Determining the Locks Held by a Task**

The locks mentioned in this section are used to serialize access to control blocks. These types of locks should not be confused with database locks.

To determine the locks held by the task from an operating system dump, locate the internal lock elements (ILEs) that document the locks held by the task as follows:

1. TCEILINU (offset X'148' into the TCE) -- Contains the current TCE lock count.
2. TCELKILE (offset X'144' into the TCE) -- Contains the address of the first ILE.
3. ILE's form a linked list, which can be walked using ILENEXT (offset X'08' into the ILE).

Routing System Snaps to a Sequential File

System snaps are routed to a sequential log file in a multitasking environment. The system snaps are routed even when you have directed the log to the data dictionary DDLDCLOG area in the system definition.

Routing system snaps to a sequential file provides reliable system snaps. While the DC/UCF system is taking a system snap for one subtask, other subtasks are temporarily frozen. This helps ensure the integrity of the storage being snapped.

DC/UCF uses the MPMODE DC to take a system snap. Because the MPMODE DB may be held by one of the frozen subtasks, the system may not be able to write the snap to the data dictionary log area. Therefore, DC/UCF attempts to write system snaps to a sequential file in a multitasking environment.

Steps to route system snaps

To route system snaps to a sequential file, perform the following steps:

1. Specify LOG DATABASE in the system generation SYSTEM statement.

2. Allocate a sequential disk file using the following DCB specifications:
   - RECFM must be FBA.
   - LRECL must be 133.
   - BLKSIZE must be a multiple of 133. A large block size requires fewer I/O operations to complete the system snap and, therefore, results in a faster snap.

3. Include the following statement in the JCL to start up the DC/UCF system:
   ```
   //CDMSLOGA DD DSN=idms.syssnap,DISP=SHR
   ```

   *idms.syssnap*
   Specifies data set name of the sequential log file for system snaps.

Considerations

The following are general considerations for routing system snaps to a sequential file:

- If the log is directed to a sequential file in the system definition (LOG FILE1/FILE2), system snaps are written to that file.

- If the log is directed to the data dictionary log area in the system definition (LOG DATABASE), system snaps are written to a sequential file with a ddbname of CDMSLOGA. If the system startup JCL does not include a DD statement for CDMSLOGA, DC/UCF does not take system snaps.

- The size in bytes of the sequential disk file should be at least three times the size in bytes of the DC/UCF region. If the space allocated to the sequential file is not large enough to hold an entire system snap, DC/UCF terminates abnormally with an x37-type system abend code.
- DC/UCF opens and closes the sequential disk file for each system snap. If the startup JCL specifies DISP=SHR for the file, DC/UCF writes to the beginning of the file each time a snap occurs. To copy each system snap to another file before the snap is overwritten, you can use the WTOEXIT user exit. Write the exit routine to:

1. Intercept the message that is written to the log when the snap is complete. The message number is DC009101.

2. Submit a job that copies the system snap file to another file.

This use of the WTOEXIT user exit is similar to using the exit to submit jobs for the ARCHIVE JOURNAL and ARCHIVE LOG utility statements.

Statistics

DC/UCF statistics report on system resource usage. These statistics are written to the system log file. The table below lists the different types of DC/UCF statistics that can be collected. This section discusses the following:

- Statistics written to the system log at runtime. Separate information is provided for the following:
  - System-wide statistics
  - Task and external request unit statistics
  - Transaction statistics
  - CA ADS dialog statistics
  - Histograms
- User-written statistics support available through use of user exits that gain control during various statistics collection activities.
- Options for examining statistics that have been written to the system log.

Types of DC/UCF statistics

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Information collected</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>General resource-usage statistics for the entire system (for example, program pool, queue area, line, physical terminal, and task statistics). These statistics are always collected.</td>
</tr>
<tr>
<td>Task and external request</td>
<td>Detailed statistics for each executing task or external request unit. System and database usage is tracked. These statistics are optional.</td>
</tr>
<tr>
<td>Transaction</td>
<td>Detailed statistics for each executing transaction. System and database usage is tracked. These statistics are optional.</td>
</tr>
<tr>
<td>Dialogs</td>
<td></td>
</tr>
</tbody>
</table>
Statistic | Information collected
---|---
| Detailed statistics for all or selected CA ADS dialogs. Control commands, system usage, and database usage are tracked. These statistics are optional.
| Histograms | Numerical totals of how many times specific events or limits occur at runtime. These statistics are optional.

System-Wide Statistics

DC/UCF always collects system-wide statistics because they require minimal overhead and they provide valuable information for tuning and maintaining the DC/UCF system. DC/UCF system-wide statistics are grouped into six categories. The table below lists statistics collected for each category. The categories are as follows:

- System statistics
- DC/UCF statistics
- Non-SQL DB statistics
- SQL DB statistics
- Line statistics
- Physical terminal statistics
- Program statistics
- Queue statistics
- Task statistics

Note: Statistics can be cumulative or interval-based.

- When System-Wide Statistics are Written (see page 311)
- Categories of Statistics (see page 312)

When System-Wide Statistics are Written

System-wide statistics and histograms can be automatically written to the log and cleared at a specified time and day. This process ensures that the values of these statistics are synchronized by avoiding overflow. It also makes analysis easier since statistics are gathered over a standard time interval.

Use interval roll time to specify the time of day and day interval when the statistics are written and reset. When the interval roll time is reached, current system-wide statistics and histograms are written to the CA IDMS log, and then cleared in preparation for the next collection interval.
Set and maintain your write or write and roll interval time with the following statement and commands:

- **SYSGEN SYSTEM statement** -- used to establish various characteristics of the DC/UCF runtime system.

- **DCMT VARY STATISTICS command** -- changes the interval at which DC/UCF statistics are written to the system log file or written and reset.

- **DCMT DISPLAY STATISTICS command** -- displays the interval or interval roll time.

- **DCMT WRITE STATISTICS command** -- provides the ability to both write and clear the statistics.

System-wide statistics are written to the DC/UCF system log at each of the following times:

- At normal system shutdown.

- At the statistics interval specified in the system generation SYSTEM statement:

  ```
  SYSTEM
  .
  .
  STATISTICS INTERVAL interval-time
  ```

  The statistics interval can be varied at runtime by means of the DCMT VARY STATISTICS command.

- At the statistics interval roll specified in the system generation SYSTEM statement:

  ```
  SYSTEM
  .
  .
  ```

  **STATISTICS ROLL TIME interval-roll-time FREQUENCY day-frequency**

  The statistics interval roll can be varied at runtime by means of the DCMT VARY STATISTICS ROLL command.

  - On explicit request by means of a DCMT WRITE STATISTICS command.
  
  - On explicit request by means of a DCMT WRITE STATISTICS ROLL command.

### Categories of Statistics

This section lists the information collected for each category of system-wide statistics. For more information on interpreting these statistics, see the following:

- **CA IDMS Reporting section**

- **CA ADS Reference section**

- **CA IDMS Administrating section**
CA IDMS - 19.0

- **CA IDMS Performance Monitor Using section**
  - System Performance

**System statistics**

- Number or amount of:
  - Journal writes
  - Program pool loads
  - Program pool waits
  - Pages loaded into the program pool
  - Logical terminal autotasks started
  - Requests for storage satisfied in the first and second scan of the storage pools (collected separately)
  - SET TIME WAIT requests issued
  - SET TIME POST requests issued
  - SET TIME START requests issued
  - SET TIME CANCEL requests issued
  - Tasks processed
  - Tasks abended
  - System tasks currently active
  - Tasks currently active (both system and user mode)
  - Times maximum-tasks condition detected
  - Task threads aborted for exceeding runaway time
  - System tasks processed
  - PGFREE requests
  - Fixed pages freed
  - Reentrant pool loads
  - Reentrant pool waits
  - Pages loaded into the reentrant pool
- Pages that became eligible for release
- Pages actually released
- XA program pool loads
- XA program pool waits
- Pages loaded into the XA program pool
- XA reentrant pool loads
- XA reentrant pool waits
- Pages loaded into the XA reentrant pool
- Storage pool waits
- PGFIX requests
- Pages fixed

- Number of times:
  - DPE threshold exceeded
  - RCE threshold exceeded
  - RLE threshold exceeded
  - ILE threshold exceeded
  - Short-on-storage conditions detected
  - DPEs in use concurrently
  - RCEs in use concurrently
  - RLEs in use concurrently
  - Entries in the TCE stack (that is, TCE stack high-water mark)

**Deadlock statistics**

- Total:
  - Deadlock detector dispatch count
  - Pass 1 (stalled task) dispatch count
  - Pass 2 (stalled task) dispatch count
Total number of:
- Deadlocked tasks with COND=NONE
- Deadlocked tasks with COND=DEAD
- Deadlock victims
- Deadlock victims with COND=DEAD
- Deadlock victims with COND=NONE

Maximum number of:
- COND=DEAD tasks deadlocked in a single pass
- COND=NONE tasks deadlocked in a single pass

**DC/UCF statistics**

Number or amount of:
- Programs called
- Program loaded
- Terminal reads
- Terminal writes
- Terminal I/O errors
- GET TIME requests
- SET TIME requests
- Database calls
- Queue GETs (DC/UCF)
- Queue PUTs (DC/UCF)
- Queue DELETEs (DC/UCF)
- Scratch GETs
- Scratch PUTs
- Scratch DELETEs
- #GETSTG requests
- #FREESTG requests
- CPU time spent in user mode. On systems utilizing zIIP processors, CPU time includes time on the zIIP processor normalized to standard processor speed.

- CPU time spent in system mode. On systems utilizing zIIP processors, CPU time includes time on the zIIP processor normalized to standard processor speed.

- Wall-clock time spent in wait state

- Maximum number or amount of:
  - Entries in the TCE stack (that is, TCE stack high water mark)
  - Storage held at one time (that is, task storage high water mark)

- Total number of system service calls

**Non-SQL DB statistics**

- Number or amount of:
  - Pages read
  - Pages written
  - Pages requested
  - CALC records stored with no overflow
  - CALC records stored with overflow
  - VIA records stored with no overflow
  - VIA records stored with overflow
  - Records requested
  - Records current of transaction
  - Database calls
  - Fragments stored
  - Records relocated
  - Locks acquired
  - Share locks held
  - Non-share locks held
  - Locks freed
  - SR8 splits
- SR8 spawns
- SR8's stored
- SR8's erased
- SR7's stored
- SR7's erased
- Btree searches
- Btree levels searched
- Orphans adopted
- Levels searched (best case)
- Levels searched (worst case)

Miscellaneous DB statistics
- Number or amount of:
  - Records updated
  - Pages found in cache
  - Pages found in prefetch buffer

SQL DB statistics
- Number or amount of:
  - SQL commands executed
  - Rows fetched
  - Rows inserted
  - Rows updated
  - Rows deleted
  - Sorts performed
  - Minimum rows sorted
  - Maximum rows sorted
  - AM recompiles

Line statistics
For each line, the number of:
- Reads performed
- Read errors that occurred
- Writes performed
- Write errors that occurred

Physical terminal statistics

For each physical terminal, the number of:
- Reads performed
- Read errors that occurred
- Writes performed
- Write errors that occurred

For each physical terminal, the total or cumulative:
- Number of responses
- Response time
- Terminal I/O time

Program statistics

For each program, the number of:
- Times called
- Times loaded
- Times waited to load
- Program check errors

Queue statistics

The number of times each queue's associated task was invoked to process queue records.

Task statistics

Number of times each task code was invoked.
Task and External Request Unit Statistics

Contents

- Enabling Statistics Collection (see page 319)
- How Statistics Accumulate at Runtime (see page 320)
  - DC/UCF Statistics (see page 321)
  - Non-SQL DB Statistics (see page 322)
  - SQL DB Statistics (see page 323)

A task is the basic unit of work under DC/UCF. Each task consists of one or more programs. An external request unit is a unit of program activity initiated from outside the DC/UCF region/partition. For example, external transactions are initiated to handle a batch program's database requests.

You enable task and external request unit statistics to do the following:

- Monitor and tune individual application programs.
- Enforce limits on task resource usage.

⚠️ **Note:** For more information on resource limits, see the *Administrating section.*

Collection of these statistics is optional because additional overhead is required to collect the statistics and because a large volume of data is generated. One statistics record is maintained per task thread for the life of the task or external request unit. These statistics records can quickly fill the system log file.

For tasks, it is sometimes sufficient to collect task histograms instead of full task and external request unit statistics. However, when you are collecting task histograms, you cannot also collect task and external request unit statistics.

⚠️ **Note:** For more information on histograms, see Histograms.

### Enabling Statistics Collection

Enable collection of statistics for tasks and external request units in the system generation SYSTEM statement as follows:

```
SYSTEM version-number .
  .
  STATISTICS TASK WRITE NOUSER/USER
```
The STATISTICS keyword controls statistics collection.

TASK WRITE enables collection of both task and external request unit statistics.

The NOUSER/USER clause specifies how CPU-time statistics are maintained:

- NOUSER -- Statistics for system-mode and user-mode execution are maintained as a single statistic. This statistic represents total task or external request unit execution time.
- USER -- System-mode and user-mode statistics are maintained separately.

How Statistics Accumulate at Runtime

Task and external request unit statistics are accumulated in control blocks at runtime. Once allocated, control blocks for a task or external request unit are maintained as long as the task or transaction is active.

Statistics in the control blocks are written to the DC/UCF log file when the task or external request unit terminates, or when you issue a DCMT WRITE STATISTICS command at runtime.

System-wide statistics and histograms are automatically written to the log and cleared at a specified time each day. This process ensures that the values of these statistics are synchronized by avoiding overflow. It also makes analysis easier since statistics are gathered over a standard time interval.

The interval roll time specifies the time of day when the statistics are written and reset. When the interval roll time is reached, current system-wide statistics and histograms are written to the CA IDMS log, and then cleared in preparation for the next collection interval.

You can set the interval roll time using the parameters on the SYSGEN SYSTEM statement and the DCMT VARY STATISTICS command. The DCMT DISPLAY STATISTICS command displays the interval roll time, and the DCMT WRITE STATISTICS command provides the ability to both write and clear the statistics.

The following fields describe the header portion of the statistics records written to the log, and are available in the #STLDS DSECT command:

- The time at the start of the interval, in internal timestamp format
- The time when the DC/UCF system started, in internal timestamp format
- The job name of the DC/UCF system
- The Central Version (CV) number

Task and external request unit statistics are sorted as follows:

- Task statistics are sorted by user identifier, by logical terminal identifier, and by task code.
- External request unit statistics are sorted by accounting data and by program name.

Statistics for tasks, external requests, and transactions
The following sections summarize the statistics collected when you enable statistics collection for tasks, external request units, and transactions. Note that even when SQL is used to access data, non-SQL database statistics (designated non-SQL IDMS statistics in the table) are still collected.

**DC/UCF Statistics**

- Number or amount of:
  - Programs called
  - Program loaded
  - Terminal reads
  - Terminal writes
  - Terminal I/O errors
  - GET TIME requests
  - SET TIME requests
  - Database calls
  - Queue GETs (DC/UCF)
  - Queue PUTs (DC/UCF)
  - Queue DELETEs (DC/UCF)
  - Scratch GETs
  - Scratch PUTs
  - Scratch DELETEs
  - #GETSTG requests
  - #FREESTG requests
  - CPU time spent in user mode. On systems utilizing zIIP processors, CPU time includes time on the zIIP processor normalized to standard processor speed.
  - CPU time spent in system mode. On systems utilizing zIIP processors, CPU time includes time on the zIIP processor normalized to standard processor speed.
  - Wall-clock time spent in wait state

- Maximum number or amount of:
  - Entries in the TCE stack (that is, TCE stack high water mark)
  - Storage held at one time (that is, task storage high water mark)
Non-SQL DB Statistics

- Number or amount of:
  - Pages read
  - Pages written
  - Pages requested
  - CALC records stored with no overflow
  - CALC records stored with overflow
  - VIA records stored with no overflow
  - VIA records stored with overflow
  - Records requested
  - Records current of transaction
  - Database calls
  - Fragments stored
  - Records relocated
  - Locks acquired
  - Share locks held
  - Non-share locks held
  - Locks freed
  - SR8 splits
  - SR8 spawns
  - SR8's stored
  - SR8's erased
  - SR7's stored
  - SR7's erased
  - Btree searches
  - Btree levels searched
Transaction Statistics

Contents

- Enabling Statistics Collection (see page 324)
- How Statistics Accumulate at Runtime (see page 325)

A transaction is a series of tasks that perform one logical activity under DC/UCF. For example, a transaction typically includes all tasks that display, retrieve, and process data on a single map.

⚠️ **Note:** This type of transaction differs from the transaction CA IDMS uses to manage resources and to control recovery.

Transaction statistics are initiated and terminated under the control of the application program and as such have no impact on recovery. The same type of information is collected for transactions as is collected for individual tasks. Statistics that are collected for transactions are listed in the previous table.
Enabling Statistics Collection

To collect transaction statistics, it is necessary to:

1. Prepare the system to collect transaction statistics at runtime
2. Prepare tasks in the transaction for which the statistics are to be collected

Preparing the system

To prepare the system to collect transaction statistics, perform the following steps:

1. Enable task statistics, since transactions are composed of tasks. For more information, see Task and External Request Unit Statistics.
2. Enable transaction statistics in one of the following ways:
   - At system generation time, specify the following in the SYSTEM statement:
     ```
     SYSTEM version-number.
     .
     .
     STATISTICS TASK TRANSACTION
     ```
     The keyword TRANSACTION enables collection of transaction statistics.
   - At runtime, issue the following DCMT command:
     ```
     DCMT VARY STATISTICS TRANSACTION ON
     ```
     The keywords TRANSACTION ON enable collection of transaction statistics.

Preparing tasks in the transaction

A transaction can contain several tasks. To collect statistics for a transaction, do the following:

1. Enable transaction statistics in the first task of the transaction. To do this, include the appropriate command in one of the programs of the task:
   - BIND TRANSACTION STATISTICS (COBOL, PL/I)
   - #TRNSTAT TYPE=BIND (Assembler)

Within a COBOL or PL/I runtime environment, transaction statistics are collected from the point in the current task at which the BIND command is issued. Within an Assembler #TRNSTAT macro, TASK=NO causes statistics to be collected from the point at which the #TRNSTAT macro is executed. TASK=YES is the default setting for the #TRNSTAT macro; in this case statistics are collected from the beginning of the current task.

2. Disable transaction statistics by including the appropriate command in the program that ends the transaction:
   - END TRANSACTION STATISTICS (COBOL, PL/I)
   - #TRNSTAT END (Assembler)
Statistics collection ends when this command is issued. To write the collected transaction statistics to the system log at the same time, include the WRITE clause in the command that ends the transaction.

**Note:** For more information on COBOL, PL/I, or Assembler commands, see the *CA IDMS DML Reference section for COBOL*, *CA IDMS DML Reference section for PL/I*, and *CA IDMS DML Reference section for Assembler*, respectively.

### How Statistics Accumulate at Runtime

Transaction statistics accumulate in system control blocks at runtime. The statistics are sorted by user and by logical terminal. As shown in the following diagram, transaction statistics accumulation occurs as follows:

1. For COBOL or PL/I programs, accumulation begins when the BIND command is issued. For Assembler programs, accumulation begins when the `#TRNSTAT TYPE=BIND` macro is issued if the TASK=NO parameter is included, otherwise accumulation starts at the beginning of the current task.

2. Stops when a program in the transaction issues a command to end the transaction.

![How Statistics Accumulate at Runtime](image-url)
Copying to the DC/UCF log or program variable storage

Accumulated transaction statistics can be copied at the discretion of the application to the DC/UCF system log file and/or to a specified location in the application program’s variable storage. Transaction statistics are copied as follows:

- Optionally during the transaction when requested by a command in one of the transaction's programs:
  - ACCEPT TRANSACTION STATISTICS (COBOL, PL/I)
  - #TRNSTAT ACCEPT (Assembler)

  The command can be issued at any point in the transaction. The copied statistics remain in the transaction statistics control blocks.

- Automatically during the transaction when successive BIND TRANSACTION STATISTICS commands are issued. In this case, transaction statistics blocks are cleared for continued use after the statistics are copied.

- Optionally at the end of the transaction, based on the command that ends the transaction. In this case, transaction statistics blocks are freed after the statistics are copied.

Note: For more information on the layout of transaction records in the log, see the CA IDMS Reporting section.

CA ADS Dialog Statistics (IDMS)

Contents

- Enabling Statistics Collection (see page 326)
- How Statistics Accumulate at Runtime (see page 327)
- Categories of CA ADS Statistics (see page 328)

Statistics for CA ADS Batch dialogs are collected in a separate log file.

More Information

- For more information on CA ADS dialogs, see the CA ADS Reference section.
- For more information on CA ADS dialog statistics, see the CA ADS Batch Using section.

Enabling Statistics Collection

To collect statistics for CA ADS dialogs, do the following:
1. Enable task statistics as discussed in Task and External Request Unit Statistics.

2. Enable transaction statistics as discussed in Transaction Statistics.

3. Enable dialog statistics in one of the following ways:
   - At system generation time, specify the following in the ADSO statement:
     ```
     ADSO
     .
     DIALOG STATISTICS ON ALL/SELECTED
     ```
     The keywords STATISTICS ON enable collection of dialog statistics. The keyword SELECTED directs collection of statistics for selected dialogs only (see below).
   - At runtime, issue the following DCMT command:
     ```
     DCMT VARY ADSO STATISTICS ON ALL/SELECTED DIALOGS
     ```
     The keywords ADSO STATISTICS ON enable collection of dialog statistics. The keyword SELECTED directs collection of statistics for selected dialogs only (see below).

The ALL/SELECTED clause specifies the following:
- **ALL** -- Statistics are collected for all dialogs that are executed.
- **SELECTED** -- Statistics are collected only for preselected dialogs when they are executed. You preselect dialogs for statistics in one of the following ways:
  - At system generation time, name the dialog in a PROGRAM statement and specify that ADSO statistics are to be collected for the dialog as follows:
    ```
    PROGRAM dialog-name
    .
    ADSO DIALOG STATISTICS
    ```
  - At runtime, name the dialog in a DCMT VARY PROGRAM command and specify that ADSO statistics are to be collected for the dialog as follows:
    ```
    DCMT VARY PROGRAM dialog-name ADSO STATISTICS
    ```

### How Statistics Accumulate at Runtime

When dialog statistics collection is enabled, the DC/UCF system collects statistics as follows:
- Statistics for overhead activity are collected and written to the DC/UCF system log whenever overhead activity is performed. This occurs once at the beginning of a CA ADS application and once at the end.
  The CA ADS statistics block identifier for statistics accumulation is either the dialog name (if statistics were explicitly enabled for the dialog) or @ADS@@AO for a catch-all statistics block.
Statistics for dialogs are collected each time a dialog issues a control command. These statistics are not written immediately to the system log. Dialog statistics are sorted by user and by logical terminal. Dialog statistics accumulate in transaction statistics blocks (TSBs) and CA ADS statistics blocks (ASBs).

**Note:** For more information on how statistics accumulate in TSBs and ASBs, see the *CA ADS Reference section*.

Each time dialog statistics are written to the system log, the TSBs and ASBs that contained the newly written statistics are initialized. The TSBs and ASBs are freed only when the application terminates. Note, however, that during a pseudo-converse they may be written to the scratch area along with record buffer blocks.

**When statistics are written to the system log**

Dialog statistics are written to the system log when one of the following occurs:

- The number of statistics accumulations equals the predefined checkpoint interval
- The application terminates

**Checkpoint intervals**

To establish a checkpoint interval for dialog statistics, perform one of the following steps:

- At system generation time, specify the following in the ADSO statement:
  
  ADSO
  
  DIALOG STATISTICS CHECKPOINT interval-number

- At runtime, issue the following DCMT command:
  
  DCMT VARY ADSO STATISTICS CHECKPOINT INTERVAL interval-number

**Categories of CA ADS Statistics**

This section identifies statistics for CA ADS dialogs. At dialog execution time, statistics are collected on control commands, general dialog execution, and record buffer block (RBB) usage:

**Explicitly coded control commands**

Statistics collected are the number of times these commands are executed:

- DISPLAY
- INVOKE
Implicitly coded control commands

Statistics collected are the number of times these commands are executed:

- DISPLAY
- DISPLAY CONTINUE
- INVOKE
- LINK TO DIALOG
- LINK TO PROGRAM
- RETURN
- RETURN CONTINUE
- TRANSFER
- LEAVE ADS
- LEAVE APPLICATION
- ABORT

General dialog execution

Statistics collected are:

- Number of:
  - Premap process executions
  - Response process executions
  - Calls to the CA ADS statistics accumulation block (ASA)
Explicit GET SCRATCH commands
Explicit PUT SCRATCH commands
Explicit DELETE SCRATCH commands
Explicit WRITE TO PRINTER commands
PUT NEW DETAIL commands
PUT CURRENT DETAIL commands
GET DETAIL commands

Size of:
- Fixed dialog block (FDB)
- Variable dialog block (VDB)

Highest and lowest link level at which dialog executed

**Record buffer block (RBB) usage**

Statistics include:

- Number of times RBBs are put in scratch records
- Greatest amount of:
  - RBB storage used for all dialogs
  - RBB free space acquired for the dialog
- Least amount of RBB storage used for all dialogs
- Amount of space when:
  - Greatest amount of RBB storage used
  - Least amount of RBB storage used
- Highest and lowest number of RBBs used

**Histograms**

**Contents**

- Overview of Histograms (see page 331)
- How Histograms Accumulate at Runtime (see page 332)
- System-wide Histograms (see page 332)
Histograms track the frequency of specific events. For example, one histogram tracks the number of
programs loaded into the program pool.

**Note:** To collect task histograms, you must disable collection of task and external request
unit statistics.

### Overview of Histograms

Histograms track how many times a specific event occurs at runtime. Histograms organize
information according to subcategories (that is, value ranges) meaningful to the event being tracked.

For example, the histogram for program pool usage keeps track of the size of programs loaded into
the pool. The following sample histogram counts the number of programs smaller than 4096 bytes,
between 4097 and 16384 bytes, between 16385 and 66536 bytes, and above 66537 bytes:

**Sample histogram:**

<table>
<thead>
<tr>
<th>How many programs loaded:</th>
<th>14</th>
<th>32</th>
<th>9</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of programs (bytes)</td>
<td>0</td>
<td>4096</td>
<td>16384</td>
<td>66536</td>
</tr>
</tbody>
</table>

Each range (for example, 0-4096) defines the bin
in which the range counter is kept.

The range counter for each bin is automatically incremented by 1 whenever an event in the
appropriate range occurs. To continue the previous example, assume that two programs between
4096 and 16384 bytes are loaded in the program pool:

The histogram is updated:

<table>
<thead>
<tr>
<th>How many programs loaded:</th>
<th>14</th>
<th>34</th>
<th>9</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of programs (bytes)</td>
<td>0</td>
<td>4096</td>
<td>16384</td>
<td>66536</td>
</tr>
</tbody>
</table>

Two more programs were loaded in the 4096-16384 range.

**Note:** For more information and a sample of SREPORTs, see the *Reporting section*.

### Types of histograms

- Task Histograms (see page 333)
- Line Histograms (see page 334)
- Overriding Histogram Defaults (see page 335)
  - Coding HSTDEF Macros (see page 335)
  - HSTDEF Syntax (see page 335)
  - HSTDEF Parameters (see page 335)
  - HSTDEF Examples (see page 336)
- Creating the RHDCCHIST Module (see page 336)
System-wide histograms, task histograms, and line histograms can be collected. For each of these types of histogram, one or more classes of events can be tracked simultaneously. Statistics are maintained separately for each class.

**Bins allocated at runtime**

Bins for each histogram class are allocated at runtime based on the low value and range increment specified for the class at startup time:

1. The low value for the histogram class specifies the start of the first bin in that class. For example, assume that a histogram is being kept of the database calls per task:

   First bin:
   
   \[
   \begin{array}{|c|c|c|c|c|c|c|c|c|c|}
   \hline
   & 5 & \ldots & x \\
   \hline
   \end{array}
   \]

   - A low value of 5 is established; information is not collected for tasks that issue four or fewer database calls.
   - The range for this bin is determined by the range increment (see below).

2. The range increment for the histogram class specifies the range of values, starting from the low value, to be collected by each bin in the class. To continue the previous example:

   The range increment is set at 3:

   \[
   \begin{array}{|c|c|c|c|c|c|c|c|c|c|}
   \hline
   5 & 8 & 11 & 14 & 17 & 20 & 22 & 25 & 28 & 31 & 34 \\
   \hline
   \end{array}
   \]

   The first bin counts tasks that issue from 5 to 7 database calls.

**How Histograms Accumulate at Runtime**

Histogram statistics accumulate in system control blocks at runtime. Histograms are written to the DC/UCF log at the following times:

- At normal system shutdown
- At the statistics interval established by the STATISTICS parameter of the system generation SYSTEM statement
- On explicit request by means of a DCMT WRITE STATISTICS command

**System-wide Histograms**

System-wide histograms are always collected. The following table lists the classes of histograms automatically collected for the system, along with the default low value and increment for each class. Bins for each histogram class are built based on the low value and increment for the class. By default, ten bins are maintained for each histogram class.
Size (in bytes) of

<table>
<thead>
<tr>
<th>Class name</th>
<th>Low value</th>
<th>Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-defined records written to the journal because of PUTJRNLR requests</td>
<td>JRNLSIZE</td>
<td>100</td>
</tr>
<tr>
<td>Programs loaded into the program pool</td>
<td>PROGSIZE</td>
<td>250</td>
</tr>
<tr>
<td>Queue records written to the queue data set</td>
<td>QUESIZE</td>
<td>100</td>
</tr>
<tr>
<td>Scratch records written to the scratch data set</td>
<td>SCRSIZE</td>
<td>100</td>
</tr>
<tr>
<td>Storage requested by #GETSTG TYPE=USER</td>
<td>USTGSIZE</td>
<td>50</td>
</tr>
<tr>
<td>Storage requested by #GETSTG (all types)</td>
<td>GSTSIZE</td>
<td>50</td>
</tr>
</tbody>
</table>

Task Histograms

Collection of task histograms is optional.

At system runtime, DC/UCF maintains a histogram for each of the classes of task histogram listed in the table below. DC/UCF collects histogram information for all tasks executed on the entire system.

To enable collection of task histograms, specify the following in the the system generation SYSTEM statement:

```
SYSTEM version-number
   .
   .
   STATISTICS TASK COLLECT NOUSER/USER
```

The STATISTICS keyword controls statistics collection.

TASK COLLECT enables collection of task histograms.

The NOUSER/USER clause specifies how CPU-time statistics are maintained:

- **NOUSER** -- Statistics for system-mode and user-mode execution are maintained as a single statistic. This statistic represents total task or external request unit execution time.
- **USER** -- System-mode and user-mode statistics are maintained separately.

Task histogram classes

Bins for each histogram class are built based on the low value and increment for the class. Default values are shown in the following tables. By default, ten bins are maintained for each histogram class.

<table>
<thead>
<tr>
<th>Number per task of:</th>
<th>Class name</th>
<th>Low value</th>
<th>Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programs called</td>
<td>PROGCALL</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Programs loaded</td>
<td>PROGLOAD</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Terminal reads issued</td>
<td>TERMREAD</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Terminal writes issued</td>
<td>TERMWRIT</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Number per task of:</td>
<td>Class name</td>
<td>Low value</td>
<td>Increment</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>Terminal I/O errors</td>
<td>TERMERR</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>#GETSTG requests issued</td>
<td>GSTGCNT</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>#GETSCR requests issued</td>
<td>GSCRCNT</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>#PUTSCR requests issued</td>
<td>PSCRCNT</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>#DELSR requests issued</td>
<td>DSCRCNT</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>#GETQUE requests issued</td>
<td>GQUECNT</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>#PUTQUE requests issued</td>
<td>PQUECNT</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DELQUE requests issued</td>
<td>DQUECNT</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>#GETIME requests issued</td>
<td>GTIMCNT</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>#SETIME requests issued</td>
<td>STIMCNT</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Database calls issued</td>
<td>DBCALLS</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Maximum number of entries in the TCE stack</td>
<td>HISTACK</td>
<td>50</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Amount of time per task in:</th>
<th>Class name</th>
<th>Low value *</th>
<th>Increment*</th>
</tr>
</thead>
<tbody>
<tr>
<td>User mode</td>
<td>USERTIME</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>System mode</td>
<td>SYSTIME</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Wait state</td>
<td>WAITTIME</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

* Expressed in units of ten-thousandths of a second

<table>
<thead>
<tr>
<th>Number per task of the following in concurrent use:</th>
<th>Class name</th>
<th>Low value</th>
<th>Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource control elements (RCEs)</td>
<td>HIRCE</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Resource link elements (RLEs)</td>
<td>HIRLE</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Deadlock prevention elements (DPEs)</td>
<td>HIDPE</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

**Line Histograms**

Collection of line histograms is optional. Line histograms collect information for each DC/UCF line defined at system generation time. Reports can be written at a site to extract line histogram information.

To collect line histograms, specify the following in the the system generation SYSTEM statement:

```plaintext
SYSTEM version-number .
  .
    STATISTICS LINE
```

The keyword STATISTICS controls statistics collection, and the keyword LINE enables collection of line histograms.

**Line histogram classes**
The following table displays line histogram classes. Bins for the LINETIME class are built based on the low value and increment for the class. Default values are shown in this table. By default, ten bins are maintained for LINETIME.

<table>
<thead>
<tr>
<th>Description</th>
<th>Class name</th>
<th>Low value</th>
<th>Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cumulative response time per line</td>
<td>LINETIME</td>
<td>50*</td>
<td>50*</td>
</tr>
</tbody>
</table>

* Expressed in units of ten-thousandths of a second

Overriding Histogram Defaults

You can override the default low value, range increment, and number of bins for each histogram class. To do this, perform the following steps:

1. Code a #HSTDEF macro for each histogram class whose defaults are to be overridden.
2. Create an RHCHIST load module from the assembled #HSTDEF macros and save it in your custom.loadlib.

Coding HSTDEF Macros

This section presents #HSTDEF macro syntax and parameters, followed by two examples.

**HSTDEF Syntax**

```
#HSTDEF histogram-class
  .LOW= low-range-value-number
  .INCR= range-increment-number
  .BINS= bin-count
  .LINE=line-id
```

**Note:** The defaults for the #HSTDEF macro do not necessarily coincide with defaults for a given histogram class (for example, LINETIME has a default low value of 50).

**HSTDEF Parameters**

- `histogram-class`
  Names the histogram class (for example, PROGCALL or LINETIME) whose defaults are to be overridden.
  Valid histogram class names are given in the discussions of system-wide, task, and task histograms earlier in this section.
- **LOW**
  Specifies the low value for the histogram's value range. *Low-range-value-number* must be a positive integer.
  Considerations: Low values for classes USERTIME, SYSTIME, WAITTME, and LINETIME represent units of one ten-thousandth of a second.

- **INCR**
  Specifies the increment added to range values for successive bins. *Range-increment-number* must be a positive integer.
  Considerations: Range increments for classes USERTIME, SYSTIME, WAITTME, and LINETIME represent units of one ten-thousandth of a second.

- **BINS**
  Specifies the number of bins in the histogram. *Bin-count* must be a positive integer less than or equal to 56.

- **LINE**
  (LINETIME class) Specifies the line whose histogram quantities are being overridden. *Line-id-a* is a line identifier specified at system generation time by a LINE statement.
  Considerations: If LINE is omitted, the #HSTDEF macro applies to all lines in the DC/UCF system.

**HSTDEF Examples**

**Examples**

The following example shows how to override defaults for statistics class PROGSIZE:

```
#HSTDEF PROGSIZE, LOW=20, INCR=20, BINS=15
```

The following example shows how to override line statistics (LINETIME) defaults for line VTAM80:

```
#HSTDEF LINETIME, LOW=2, INCR=100, BINS=12, LINE=VTAM80
```

**Creating the RHDCHIST Module**

To create the RHDCHIST module, assemble and link edit the #HSTDEF macros. To do this, use the appropriate JCL or commands, as provided below, for z/OS, z/VSE, and z/VM operating systems.

**z/OS RHDCHIST assembly and link edit**

**To create a RHDCHIST load module**

1. Create a source module as follows:

   ```
   RHDCHIST CSECT
   #HSTDEF macros
   END
   ```

2. Save the source module in your custom source library.

3. Assemble and link it into your custom load library by executing the **z/OS Assemble and Link-Edit JCL** (see page 465).
4. Substitute the name of your source member and insert the following binder statement:

   NAME  RHDCHIST(R)

**z/VSE RHDCHIST assembly and link edit**

1. Assemble and catalog the object module using the sample JCL in **z/VSE Assemble JCL (see page 476)**.
   Modify the JCL by substituting the following in place of the Assembler input statements:
   
   ```
   PUNCH 'CATALOG rhdchist.OBJ REPLACE=YES'
   RHDCHIST CSECT
   #HSTDEF macros
   END
   ```

2. Link the RHDCHIST phase using the sample JCL in **z/VSE Link JCL (see page 476)**.
   Modify the JCL by substituting the following statements in place of the Linkage editor control statements:
   
   ```
   PHASE RHDCHIST,*
   INCLUDE rhdchist
   ENTRY RHDCHIST
   ```
   
   **rhdchist**
   Specifies the name of your RHDCHIST object module.

**z/VM RHDCHIST assembly and link edit**

```
GLOBAL MACLIB idmslib
FILEDEF TEXT DISK RHDCHIST TEXT A
ASSEMBLE hstdef (NODECK OBJECT

FILEDEF SYSLST PRINTER
FILEDEF SYSLMOD DISK idmslib LOADLIB a2 (RECFM V LRECL 1024 BLKSIZE 1024
LKED linkctl
Linkage editor control statements (linkctl):

INCLUDE RHDCHIST
NAME RHDCHIST(R)
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hstdef</td>
<td>filename of the file containing the #HSTDEF macro statements</td>
</tr>
<tr>
<td>idmslib</td>
<td>filename of the CA IDMS MACLIB library</td>
</tr>
<tr>
<td>idmslib LOADLIB a2</td>
<td>file identifier of the CA IDMS LOADLIB library</td>
</tr>
<tr>
<td>linkctl</td>
<td>filename of the file containing the linkage editor control statements</td>
</tr>
</tbody>
</table>

**User-Written Statistics Support**

DC/UCF provides user exits that allow the user to gain control at selected points in the statistics-writing process. These exits can be used, for example, to accumulate statistics of special interest in a separate file or to take a specified action in response to a statistical value.

The following are the statistics user exits:
### Examining Statistics

#### Contents
- Current Runtime Statistics (see page 339)
- Logged Statistics (see page 339)

You can examine statistical information both before and after it is written to the DC/UCF log. For example:

- Current execution can be traced by examining statistics as they occur, before they are written to the log.
- Specific runtime events can be evaluated by examining logged statistics.
- Runtime trends can be traced by examining logged statistics.

The following pages present information on examining current runtime statistics and logged statistics.

<table>
<thead>
<tr>
<th>User exit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDMSSVCCX (SVC exit)</td>
<td>Called at external request unit signon. Used to extract job card information.</td>
</tr>
<tr>
<td>Exit 5 (task termination exit I)</td>
<td>Called whenever a task terminates, before task statistics (if any) are written.</td>
</tr>
<tr>
<td>Exit 6 (task termination exit II)</td>
<td>Called whenever a task terminates, after task statistics (if any) are written.</td>
</tr>
<tr>
<td>Exit 9 (system statistics exit)</td>
<td>Called whenever system-wide statistics are to be written, before they are written.</td>
</tr>
<tr>
<td>Exit 15 (VIB statistics exit)</td>
<td>Called whenever system and/or task statistics are written to the variable information block (VIB). The exit is called after the statistics are written, but before the VIB is released.</td>
</tr>
<tr>
<td>Exit 31 (transaction statistics exit)</td>
<td>Called whenever system and/or task statistics are written from the transaction block. The exit is called after the statistics are written but before the transaction block is freed.</td>
</tr>
</tbody>
</table>

⚠️ **Note:** For more information on user exits, see User Exits.
Current Runtime Statistics

You can use any of the following methods to examine statistical information that is not yet written to the system log:

- DCMT commands -- Provide online summaries of statistics currently contained in active control blocks
- OPER task -- Permits dynamic monitoring of various statistics currently contained in active control blocks

⚠️ **Note:** For more information on DCMT commands and the OPER task, see the *System Tasks and Operator Reference section*.

Logged Statistics

DC/UCF provides utilities for use when the system log file is assigned to the database (that is, the DDLDCLOG area).

If the log is not assigned to the database, use the appropriate operating system utility to examine the system log. For example, use IEBGENER for z/OS systems, DITTO for z/VSE systems, or the COPYFILE command for z/VM systems.

Use the following methods to examine logged statistics:

- The PRINT LOG utility statement prints the current contents of the system log file, including statistics written to the file.
- The ARCHIVE LOG utility statement archives the system log file and reports on the contents of the archived log.

⚠️ **Note:** For more information on how to execute the print log utility, see the *Administrating section*.

- Statistics reports (SREPORTs) summarize data contained in the archived system log file. The following table lists the available SREPORTs.

⚠️ **Note:** For more information on SREPORTs, see the *Reporting section*. 
System Performance

DC/UCF system performance largely depends on site-specific factors. Factors such as the system environment (both hardware and software), the number of systems running concurrently, and the type of work being done by each system make it difficult to prescribe any single method of system tuning.

However, system performance problems generally are the result of resources not being available as required at runtime. This section discusses performance issues that you can address by adjusting parameters specified at system generation time.

To monitor performance trends, you can use DC/UCF statistics and histograms, as described in section 9, Statistics. You can display current performance information at runtime by using DCMT commands and the OPER task. To view the contents of the system log, you can use the OLP system task.
Database Operations

Contents

- Waiting on Full Journal Message (see page 342)

This section presents system generation parameters for locking and journaling that affect database performance.

Note: For a complete discussion about locking and journaling, see the Database Administering section.

Locking

<table>
<thead>
<tr>
<th>Sysgen parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCK LIMIT (SYSTEM and TASK statement s)</td>
<td>Limits the total number of record locks that can be held concurrently by specific types of transactions.</td>
</tr>
<tr>
<td>SYSLOCKS (SYSTEM statement )</td>
<td>Specifies the maximum number of record locks that the system can maintain concurrently for all transactions. If this value is too low, the lock manager allocates secondary blocks of lock storage out of the first available storage pool. DC/UCF releases the storage when it is no longer required; however, you may notice an impact on overall system performance due to unplanned demand on the storage pool and increased CPU usage by the lock manager.</td>
</tr>
<tr>
<td>AREA ACQUISITION ON THRESHOLD (SYSTEM and TASK statement s)</td>
<td>Defines the point at which, during ready processing, the system will begin to accumulate area locks for a database transaction if the system is readying multiple areas at the same time.</td>
</tr>
</tbody>
</table>

Journaling
<table>
<thead>
<tr>
<th>Sysgen parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOURNAL FRAGMENT NUMBER (SYSTEM statement)</td>
<td>Defines an interval at which DC/UCF writes a dummy journal record to the journal for the purpose of improving warmstart performance.</td>
</tr>
<tr>
<td>JOURNAL TRANSACTION LEVEL (SYSTEM statement)</td>
<td>Defines the number of transactions over which DC/UCF will defer writing the buffer to the journal in order to produce fuller journal buffers and to decrease journal I/O.</td>
</tr>
<tr>
<td>NOJOURNAL RETRIEVAL (SYSTEM statement)</td>
<td>Suppresses the writing of checkpoints for retrieval transactions.</td>
</tr>
<tr>
<td>ON COMMIT ON ROLLBACK (SYSTEM &amp; TASK statements)</td>
<td>Controls behavior upon COMMIT and ROLLBACK CONTINUE statements; may improve performance during warmstart and rollback operations.</td>
</tr>
</tbody>
</table>

**Waiting on Full Journal Message**

CA IDMS provides enhanced handling for journal files that fill because long running transactions do not commit their changes. Such transactions can fill the journals because the ARCHIVE JOURNAL utility is unable to remove the BFOR images for uncommitted transactions. When the journals fill, the system comes to a halt. In order to correct the situation, the task that is filling the journals must be canceled.

To assist in this process, CA IDMS writes the following message for each task that is waiting to write to a full journal file:

DC205024 Journal Write waiting on full Journal

The message is repeated every few seconds until tasks are no longer waiting on a full journal.

To recover from this situation, do the following:

1. Identify the task that is filling the journal files and abort the task.

2. After its changes are rolled out and an ABRT checkpoint is written, issue a DCMT VARY JOURNAL command so the central version swaps to a new journal and the full journal can be offloaded and condensed by ARCHIVE JOURNAL.

It is likely that DCMT VARY JOURNAL will need to be issued more than once, since several journal files may have filled and require offloading.
Once the system swaps back to the initial journal file on which tasks waited, processing should continue without the need for further intervention.

**Deadlock Detector Performance Management**

You can set the following parameter on the SYSTEM statement at system generation to control the CA IDMS deadlock manager:

**DEADLOCK DETECTION INTERVAL**

This parameter establishes a time interval for invoking the deadlock detector.

You can also set this value dynamically using the DCMT VARY DEADLOCK command.

⚠️ **Note:** For more information, see the *System Tasks and Operator Reference section.*

**Monitoring deadlock performance**

Statistics identifies the deadlock statistics DC/UCF writes to the DDLDCLOG area when the system is shut down. The following table identifies how to interpret those statistics:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of deadlocks detected</td>
<td>This number is the sum of the total number of COND=DEAD and COND=NONE deadlocks. This value should be close to zero. If a system is experiencing no deadlocks, you could set the deadlock detection interval to a very high value (for example, 10 minutes) in order to reduce resource consumption by the deadlock detector.</td>
</tr>
<tr>
<td>The ratio of tasks processed in pass 1 to the number of times the deadlock detector was dispatched</td>
<td>This ratio signifies the number of tasks that are potentially deadlocks to the number of times a deadlock check was performed. The ratio should be at least greater than 1 and should be as high as possible. You can increase the value of this statistic by raising the detection interval value.</td>
</tr>
<tr>
<td>The ratio of tasks processing in pass 2 to the total number of deadlocks encountered.</td>
<td>This ratio indicates how effective the deadlock detector interval is because it indicates the average number of tasks examined per deadlock. This ratio should be greater than 1. In a system experiencing few deadlocks, this ratio should be as high as possible. In a system experiencing many deadlocks, this ratio should be smaller (but still greater than 1). You can increase or decrease the ratio by raising or lowering the deadlock detection interval.</td>
</tr>
</tbody>
</table>

**Program Loading**

Contents
By improving program loading performance, you can reduce the amount of time and the number of I/O operations required to load programs.

General Strategies

To enhance DC/UCF program loading performance, use the following strategies, as appropriate at your site:

- Tailor the block size specified for the DDLDCLOD and DDLCATLOD load areas and for load (core-image) libraries based on the typical size of programs in the area or library and on the device type. Because programs are loaded one block at a time, this strategy can help reduce the amount of I/O required to load programs. For example, assume that program X is about 200K bytes in size. If the block size is 1K, 200 I/O operations will be required to load the program. If the block size is 20K, 10 I/Os will be required to load program X.

- Evaluate programs for loading efficiency. For example, you may reduce the number of loads required for a program if you use internal subroutines rather than external subroutines.

- Define frequently used programs at system generation time as resident. For example, define frequently used subschemas, DML programs, maps, and CA ADS dialogs. Assign the NODYNAMIC parameter to production programs defined at system generation time. This parameter prevents users from defining additional versions of the program at runtime. Be sure to define at system generation time only those programs that you expect to execute on a regular basis. Infrequently used programs can be defined at system runtime.

- Define subschemas for a production environment at system generation time, using the following additional strategies:
  - Put subschemas in the CDMSLIB load (core-image) library. Put the dictionary network subschema (for example, IDMSNWKA) at the front of the CDMSLIB concatenation.
  - Exclude SUBSCHEMA from the UNDEFINED PROGRAM COUNT clause in the system generation SYSTEM statement. This causes the system to skip the overhead associated with using null PDEs when subschemas are loaded.
  - Use XA program pools or XA reentrant pools to increase the amount of pool space available at runtime.

**Note:** For more information, see Extended Addressing and Multitasking.

- Predefine load area run units to handle program loading operations for the primary dictionary. You also can precondition load run units for secondary dictionaries.

**Note:** For more information on predefined run units, see System Run Units.
Clear out the data dictionary DDLDCLOD area by using the CLOD task. This strategy applies only at sites where the DC/UCF system remains running for long periods of time. The CLOD task is executed automatically by DC/UCF at startup time.

Reduce the search path required to load frequently used programs. You have the option to define alternate search paths, called load lists. More information on load lists is presented below.

**Defining Load Lists**

A load list defines the search path that DC/UCF uses to locate and load programs for execution at runtime. For compatibility with Release 10.2, DC/UCF provides the following SYSLOAD load list:

```
ADD LOADLIST SYSLOAD
  DICTNAME IS USER-DEFAULT VERSION IS USER-DEFAULT
  DICTNAME IS SYSTEM-DEFAULT VERSION IS USER-DEFAULT
  LOADLIB IS USER-DEFAULT
  DICTNAME IS USER-DEFAULT VERSION IS 1
  DICTNAME IS SYSTEM-DEFAULT VERSION IS 1
  LOADLIB IS SYSTEM-DEFAULT
```

**Customized load lists**

A production system should not use the SYSLOAD load list. For performance reasons, create a customized load list that contains the minimum number of entries required to support the production environment. For example, you can create a load list that includes only the load area and load libraries appropriate for a given application.

A site defines load lists at DC/UCF system generation time by using the LOADLIST statement. When you define a load list, you specify the following:

- The dictionaries and load libraries that are to be searched
- The order in which dictionaries and load libraries are to be searched

**Specifying the runtime load list**

You specify the load list to be used at runtime in one of the following ways:

- At system generation time, specify which load list is to be used by the system by including the LOADLIST parameter in the SYSTEM statement. This load list is the system default load list.
- At runtime, specify the load list to be used for your current terminal session by using the DCUF SET LOADLIST command. The load list specified in this way is the terminal session default load list.

**More Information**

- For more information on system generation specifications, see the *Administrating section*.
- For more information on DCUF commands, see the *System Tasks and Operator Reference section*.

**How DC/UCF uses load lists**

DC/UCF selects the location from which a program is loaded based on the following:
1. Criteria specified on the #LOAD statement. If a dictionary name is specified on the #LOAD statement, DC/UCF loads the program from that dictionary.

2. The current load list. If a terminal session default load list has been specified by using a DCUF SET LOADLIST command, DC/UCF searches dictionaries and load libraries in the specified order.

3. The system default load list. DC/UCF searches dictionaries and load libraries in the order specified in the system default load list when both of the following are true:
   - No dictionary name is specified on the #LOAD statement.
   - No terminal session default load list has been specified by using a DCUF SET LOADLIST command.

Note: For more information on load lists, see the Administrating section.

System Resource Management

Fine tuning how DC/UCF controls resources can improve and speed task execution. This section discusses the following topics:

- Task Resource and Deadlock Management (see page 346)
- Resource Limits (see page 347)

Task Resource and Deadlock Management

A task is the basic unit of work under DC/UCF. A task consists of the execution of one or more programs.

DC/UCF manages task resources by using resource control elements (RCEs), resource link elements (RLEs), and deadlock prevention elements (DPEs). At system generation time, you specify how many of each element can be allocated at runtime. At runtime, the system uses these preallocated elements to track task resources usage.

If the TIMES THRESHOLD EXCEEDED system statistic for RCEs, RLEs, or DPEs is high at your site, you can improve system performance by allocating more of the indicated element. It can be especially useful to examine trends in your system’s use of DPEs. If an increasing number of DPEs is used at runtime, contention for resources is increasing.
Resource Limits

You define resource limits to specify the maximum amount of specific resources a given task can use at runtime. The table below lists the system generation parameters that you use to define and enforce resource limits.

More Information

- For more information on system generation parameters, see the Administrating section.
- For more information on DCMT LIMITS commands that you can use to control resource limits at runtime, see the System Tasks and Operator Reference section.

Resource limit considerations

When specifying resource limits, keep in mind the following:

- Limits on task resource usage can be enforced only when task statistics are being collected.
- User exit 20 can be used to handle situations when task resource limits are exceeded. For example, user exit 20 can continue the task or abend it, examine other limits, or write messages to the system log. For more information on user exit 20, see User Exits.
- The MAXIMUM CONCURRENT THREADS parameter of the system generation TASK statement can be used to control the number of threads that can be active concurrently for a given task code. This limit is particularly appropriate for tasks that consume large amounts of system resources.
  At runtime, you can override this specification by using the DCMT VARY TASK command.
- The SYSLocks parameter of the system generation SYSTEM statement and the AREA ACQUISITION THRESHOLD parameter of the SYSTEM or TASK statement affect task usage of record locks. However, implementing explicit limits for record locks as described in the table below provides more control over this resource. This is because user exit 20 is called based only on resource limit violations.

Parameters used to set task resource limits

The following table identifies parameters you can use to establish task resource usage limits. You establish the limits at system generation time. You can change limit specifications at runtime by using DCMT commands. For information on setting limits for external request units, see External Request Units.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL/DBIO/LOCKS/STORAGE LIMIT</td>
<td>For online and/or for ERUS tasks, defines limits for the following: System service calls Database I/O operations Record locks Storage</td>
</tr>
<tr>
<td>(SYSTEM statement)*</td>
<td>At runtime, you can override the limits by using the DCMT VARY TASK or DCMT VARY ACTIVE TASK commands.</td>
</tr>
</tbody>
</table>
Parameter | Description
---|---
LIMITS FOR ONLINE | Enforces the limits defined for online tasks in the above parameter.
(System statement) | At runtime, you can override this specification by using the DCMT VARY LIMITS command.

* CALL/DBIO/LOCKS/STORAGE LIMIT also is a parameter of the system generation TASK statement. You can use the TASK statement to override limits for online and ERUS tasks.

### Tasks

The maximum number of tasks that can be executed at a given time is determined by a number of factors. A value is determined at startup based on various values specified in the system generation parameters and certain startup parameters. This value can be modified with a DCMT command. In addition, the system will automatically prevent new tasks from starting when it encounters a shortage of certain system resources such as storage, RCEs, or RLEs.

Regulating the maximum number of concurrent tasks is one of the most important tools in maximizing system performance. It is not easy to predict the optimal value for a given system. A good strategy is the following:

- Set the system generation MAXIMUM TASK and/or MAXIMUM ERUS values to a high number.
- Immediately after startup, use the DCMT VARY ACTIVE TASK MAX TASK command to set the value to a relatively low number.
- Observe system performance using the measures discussed below.
- Use DCMT to increase or decrease MAX TASK value.

The DCMT command can be repeated as often as necessary to determine optimal system performance. It is possible that the parameters will need periodic adjustment during different processing periods when a different mix of applications puts different demands on the system. Once a pattern is established, it will probably be possible to set the parameters at startup and only vary them when a known change in processing occurs.

The following indicators can be used to aid in determining whether the maximum number of concurrent tasks should be increased or decreased:

- If the system is experiencing a shortage of resources, the amount of that resource should be increased or the MAX TASK value should be decreased. Indicators of resource shortage include:
  - Message DC015007 appears on the console.
  - Message DC002003, DC002004, DC002005, or message DC244003 appears on the console or in the CA IDMS log.
  - PMRM shows excessive wait time.
- If PMRM or an external monitor shows that active tasks are experiencing long wait times, it may be desirable to decrease the MAX TASK value.
Observe the total number of tasks that execute in a given time interval. Vary the MAX TASK value either up or down and observe throughput over the same time interval.

It is also important to avoid deadlocks. If frequent deadlocks are occurring, determine the resource involved in the deadlocks. Either:

- Increase the availability of that resource, for example, by increasing the size of the storage pool
- Reduce the MAX TASK parameter
- Modify the involved application programs to avoid concurrent use of the resource. This can be done through the use of the DML verbs ENQUEUE and DEQUEUE command. These DML commands are particularly useful for applications experiencing deadlocks involving DBKEYs. Another technique that can be used for this purpose is to use the MAXIMUM CONCURRENT clause on the appropriate TASK definition statement(s) or the NONCONCURRENT clause on the appropriate PROGRAM definition statement(s) in the system generation parameters.

Terminal Exception Response Protocol

Users can reduce I/O overhead significantly by assigning VTAM exception response protocol to certain tasks or lines. Exception response protocol reduces CPU time because terminals are not required to respond with a completion status when a terminal write is requested from the DC/UCF system.

You enable exception response protocol at system generation for specific lines and tasks. To do this, you use the system generation LINE and TASK statements.

Note: For more information on these statements, see the Administrating section.

Considerations

The following considerations apply when using VTAM exception response protocol:

- If exception response protocol is specified for a task, that task runs with exception response protocol.
- If exception response protocol is specified for a line, all tasks performed on that line run with exception response protocol.
- If the terminal sends any error notification, DC reports it on the next user request.
- Printers always use definite response protocol so that buffers can be saved if print errors occur.
File Cache in Memory

Contents
- Terminology (see page 350)
- Exploiting File Cache in Memory (see page 350)

With the introduction of 64-bit hardware and the operating systems which could exploit it, the theoretical amount of virtual storage available to an application increased to 16 exabytes. CA IDMS can exploit this high amount of storage by caching entire database files in memory.

This feature has the following benefits:
- Reduced number of I/Os
- Increased throughput
- Less CPU usage

Note: For more information on 64-bit addressing, see the IBM manuals, z/OS MVS Extended Addressability Guide or z/VSE Extended Addressability.

Terminology

The following terms are used in this discussion of file cache in memory:

- The bar -- The bar marks the 2-gigabyte limit of 31-bit addressing. This is analogous to the line, which marks the 16-megabyte limit of 24-bit addressing.

- Z-storage -- Virtual storage above the bar.

Exploiting File Cache in Memory

Database files with a high number of I/Os are good candidates for the file cache in memory feature. The DBA should use standard performance-monitoring tools to determine which database files these are. Once the decision is made as to which files will use this feature, the DBA should perform these steps:

Compute the total amount of storage that is needed to cache the selected files. To do this, for each file multiply the number of blocks in the file by the file's block size and sum all results. This sum is the total amount of Z-storage needed for file caching. Be aware that if the "Scratch in Storage" SYSGEN option is specified, it may also use Z-Storage.
- Make sure that the jobs that use the modified DMCL have enough Z-storage (at least the amount computed above) at their disposal. The amount of Z-storage available to a job is limited by the MEMLIMIT parameter. For an explanation of MEMLIMIT, see the IBM manuals, z/OS MVS Extended Addressability Guide or z/VSE Extended Addressability.

For z/OS you can set MEMLIMIT in the following ways:

- Through an installation default. For more information, see the IBM manual, z/OS MVS Initialization and Tuning Reference.

- In the JOB and EXEC statements. For more information, see the IBM manual, z/OS MVS JCL Reference.

- Through an installation exit. For more information, see the IBM manual, z/OS MVS Installation Exits.

For z/VSE you can set the MEMLIMIT through the SYSDEF MEMOBJ command. For more information on this command, refer to the IBM manual z/VSE System Control Statements.

- Change the DMCL definition for each file to specify MEMORY CACHE YES.

  **Note:** For more information on changing the DMCL definition, see the Database Administering section. Note: It is also possible to dynamically change the MEMORY CACHE attribute for a file using the DCMT VARY FILE command. For more information on DCMT commands, see the System Tasks and Operator Reference section.

  **Note:** On z/VSE, CV must run in a single-partition address space to use Z-storage.

---

### Parallel Access Volume Exploitation

This feature provides CA IDMS I/O performance improvements through exploitation of the Parallel Access Volume feature on Enterprise Storage System DASD devices, such as IBM’s Shark. This feature allows multiple users and multiple jobs to simultaneously access the same logical volume and perform concurrent I/Os to a file.

PAV devices support multiple concurrent I/Os against the same disk unit. However, by default PAV devices ensure that multiple I/Os to the same disk extent are single-threaded. This is known as collision checking. Because CA IDMS routinely issues concurrent I/Os to the same extent, collision checking prevented full exploitation of PAV devices. Since CA IDMS ensures that the I/O requests it issues do not conflict with each other, CA IDMS is able to disable collision checking, allowing PAV devices to be fully exploited.

When disk I/Os for the same file are waiting because of disk extent collision checking, implementing PAV support reduces I/O wait times. Reduced I/O wait times should increase transaction throughput and improve response times.
PAV support occurs automatically when a file is on a properly defined PAV device and does not occur otherwise. The systems programmer is responsible for defining the device to the operating system correctly. For example, if no alias Unit Control Blocks (UCBs) are defined for a PAV device, the I/Os are single threaded on the primary UCB and negate the advantage of no collision checking.

**PDSE Support**

PDSEs provide the following capabilities:

- It is the only library capable of containing load modules that are greater than 16 megabytes.

  **Note:** CA IDMS does not require such support.

- A PDSE does not require condensing.

CA Technologies recommends the use of STEPLIB rather than the CDMSLIB for CA IDMS load libraries used in batch jobs. No authorization is required.

UCFTSO can use PDSEs in STEPLIB or CDMSLIB.

The CA IDMS online system supports loading CA IDMS programs from a PDSE. This requires early initialization of the CA IDMS program call (PC) environment. Establishing the PC environment requires authorization, which can be obtained using one of the following methods:

- Running CA IDMS as an authorized program.

- By specifying a CA IDMS SVC number as a PARM parameter in the startup JCL. This SVC acquires sufficient authorization to construct a PC environment, without requiring you to start CA IDMS as an authorized program.

  **Note:** The SVC number specified can be the SVC used by DC/UCF, but it is not required. This option is not valid if the specified SVC was created with the #SVCOPT parameter AUTHREQ=YES. **Note:** For more information on the PARM parameter, see the section Specifying Runtime Options (see page 508).

  **Note:** When defining the datasets for your SMP/E environment, there is the option DSN TYPE. The default option in CSM is LIBRARY. If LIBRARY is used, the SMP/E datasets will be created as type PDSE. There are special considerations when using PDSE datasets. See **PDSE Support (see page 352)** for more information.
Journaling Performance

Journaling performance can be improved by setting a non-zero journal transaction level.

⚠️ Note: For more information on specifying a journal transaction level, see the Database Administering section.

Improving Recovery Performance

CA IDMS allows you to control the following commit and rollback behavior:

- The type of journal record written on a commit
- Whether a new local transaction ID is assigned on a rollback continue or commit

Exploiting these capabilities may improve recovery time during warmstart and rollback operations and reduce the likelihood of duplicate transaction IDs when the local transaction ID values wrap.

⚠️ Note: For more information on specifying commit and rollback behavior, see the following guides:

- Database Administering section
- Administrating section

System Tracing and the System Log

The DC/UCF system log records information on system activities. A separate journal is used to record database activities.

⚠️ Note: For more information on journal files, see the Database Administering section.

The system log contains the following types of information on DC/UCF system operations:
Abends
Cancellations
Messages
Normal shutdowns
Snap dumps (for abends)
Startups
Statistics
Trace records
Transaction and task starts
Warmstarts

Messages are sent to the log when they have a destination of LOG. A snap dump is sent to the log for an actual or assumed abend in a transaction or task.

**Note:** For more information on statistics in the system log, see the Reporting section.

**Defining the system log**

You define the system log at system generation time, by using the LOG parameter of the SYSTEM statement. For more information on the SYSTEM statement, see the Administrating section. You can define the system log in the following areas:

- The database, by assigning the log to the DDLDCLOG area of the data dictionary
- Sequential files, by assigning the log to one or more sequential files

You name sequential log files for a system in the JCL or commands that you use to start up the system, as discussed in System Startup.

Under z/VSE, also define sequential log files by using the #DVFILE macro discussed in Setting Up Interpartition Communication and the SVC (see page 58).

**Enhanced System Tracing**

**Contents**
- Trace Tables and Entry Types (see page 355)
- Saving Trace Data (see page 355)
- Specify Trace Options (see page 356)
CA IDMS - 19.0

- How to Reduce the Number of Missed Entries (see page 356)
- Printing and Archiving Trace Information (see page 356)
- Format of Trace Entries in Snap Dumps (see page 357)
- Enhanced Utility Support (see page 357)

Enhanced tracing provides improved diagnostic tools for faster problem resolution by technical support. The following table lists the enhanced capabilities:

- Optional merging of system and extended (formerly called DBTRACE) trace information for easier determination of the events leading up to a problem.
- The ability to save trace data in a persistent data store (either a new trace area or the log area) so that more information is available for problem diagnosis.
- The ability to report on saved trace information and chronologically merge information from multiple data sharing members.
- Syntactic control over the inclusion of trace information in task snaps, which eliminates the need for optional APAR flag 246.

Trace Tables and Entry Types

You can allocate two types of system-wide trace tables: a system trace table and an adjunct trace table. The two types of tables are used as follows:

- The system trace table records the standard system trace (SYSTRACE) entries. As in prior releases, standard system tracing is enabled using either the SYSGEN SYSTEM statement SYSTRACE parameter or the DCMT VARY SYSTRACE command. If system tracing is enabled dynamically and no system trace table exists, one is allocated automatically.

- Extended trace entries are variable in length and are generated only if certain CSA flags are turned on using either SYSIDMS parameters or the DCMT VARY CSAFLAGS command. Extended trace entries are recorded in the adjunct trace table if one exists; otherwise, they are recorded in the system trace table. If neither table exists, extended trace entries are not generated even if the CSA flags are on.

Saving Trace Data

Saving system trace data increases the information available for problem diagnosis. When saving trace data, CA IDMS periodically writes trace entries to a persistent data store: either a trace area or a log area.

A trace area is a type of system area that can be defined. It is similar to a log area and, to be usable, it must either be included in the SYSTEM segment or be accessible through the SYSTEM DBNAME. It differs from a log area in that its name is DDLCTRC instead of DDLDCLOG. The new trace area is archived and printed using the new ARCHIVE TRACE and PRINT TRACE utility statements. Just as for a log area, a job to archive the trace area can be automatically submitted using the WTO exit.

When you enable saving trace data, consider the following:
If a trace area is defined in the run time DMCL, trace entries are written to the trace area rather than the log area. Trace information is never written to a sequential log file.

If large volumes of trace data are to be saved, you should define a trace area to minimize the impact on system throughput.

If an adjunct table exists, only its contents are saved.

If a system trace table exists without an adjunct table, the contents of the system trace table are saved. This can generate high volumes of trace output.

Specify Trace Options

Tracing options include the sizes of the system and adjunct trace tables, and whether trace information is saved or not. You can specify these options using SYSGEN syntax, DCMT commands, and SYSIDMS parameters.

Note: Options specified through SYSIDMS parameters generally override those specified on the SYSGEN statement. Options specified through DCMT commands dynamically change the attributes currently in effect for a CV.

How to Reduce the Number of Missed Entries

Eliminating missed trace entries can be difficult; however, there are steps you can take to reduce the number of missed entries. In the trace information output, if the value for number of times entries missed is large compared to the value for number of save requests, consider taking one or more of the following actions:

- Save trace information to the trace area rather than the log area.
- Reduce the amount of trace information being saved. If only extended trace information is of interest, be sure to allocate an adjunct table so only extended trace information is saved.
- Increase the size of the trace or adjunct table.
- Ensure that the appropriate archive utility is executed often enough that the trace area does not fill. The easiest way to do this is to automate the submission of the archive job using a WTO exit.

Printing and Archiving Trace Information

The following features facilitate the printing and archiving of trace information:

- Trace entries in snap dumps include additional information.
- Utilities let you the print and archive saved trace information.
- The sample WTOEXIT program supports automatic archiving of trace information.
- The supplied WTOEXIT checks for message DC050004 (LOG FULL) in addition to DC050001 (% free halved).
Format of Trace Entries in Snap Dumps

The format of trace entries in snap dumps shows information for system trace entries and displays extended trace entries, if they exist in the system trace table.

Entries appear in newest to oldest order. Each entry includes the UTC time and date it was generated, the address of the primary LTE associated with the task, and the TOD clock value at the time it was generated.

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Release</th>
<th>Trace Entries, Ordered Newest To Oldest.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>012610</td>
<td>10.09.37</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Release</th>
<th>Trace Entries, Ordered Newest To Oldest.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>012610 10.09.37 TRACE ENTRIES, ORDERED NEWEST TO OLDEST.</td>
</tr>
</tbody>
</table>

Enhanced Utility Support

The following new utilities manipulate saved trace information:
The PRINT TRACE utility reports on trace information that resides in the DDLDCTRC area, the DDLDCLOG area, or a trace or log archive file. You can use the PRINT TRACE utility to report chronologically merged trace information from multiple data sharing members.

**Note:** The ARCHIVE LOG utility archives trace information written to the DDLDCLOG area, but will not print it. You must use the PRINT TRACE utility for that purpose.

The ARCHIVE TRACE utility archives and optionally reports on the contents of the DDLDCTRC area.

## Maintaining the System Log

The way you maintain the system log depends on the log assignment. Strategies are discussed below for the database log area of the data dictionary and for sequential log files.

### Database Log (DDLDCLOG Area)

DC/UCF makes it easy to maintain the database log (DDLDCLOG) area and the trace log (DDLDCTR) areas so that they do not become full. During runtime, you maintain each log by:

1. Monitoring the amount of available space that remains in the log
2. Archiving (offloading) the log before it becomes full

When the DDLDCLOG or DDLDCTR area becomes full, the DC/UCF system halts execution and waits for system operators to offload the area.

### Monitoring available log space

Available space in the system log can be monitored as follows:

Automatically by the system -- At runtime, DC/UCF monitors the amount of available unused space in the DDLDCLOG or DDLDCTR area. Each time the amount of available space in the log halves, DC/UCF sends a message to the operator's console. This message indicates the percentage of used space in the log area.

Selectively by users -- Authorized users can monitor the percentage of *used space* in the log area by using one of the following commands:

- `DCMT DISPLAY LOG`
- `DCMT DISPLAY TRACE`
- `DISPLAY LOG` (console operator command)
Archiving the log

You use the ARCHIVE LOG utility statement to archive the system log. You use the ARCHIVE TRACE to archive the trace log. You can use the PRINT LOG or PRINT TRACE utility statement to format reports of the active log file. For more information on these utilities, see the CA IDMS Administrating section.

You can archive (offload) the DDLDCLOG or DDLDCTRC area as follows:

- While the DC/UCF system is active. In this case, the system proceeds unless the log is full.
- At system shutdown. It is not mandatory to offload a partially full log at shutdown. At system startup, DC/UCF begins writing to the unused portion of the log area.

You can use the WTOEXIT user exit to automatically submit an ARCHIVE LOG or ARCHIVE TRACE utility statement that archives the log or trace area before it becomes full. The sample WTOEXIT includes support for both commands.

Sequential Log Files

Sites that assign the system log to one or more sequential files need to develop a strategy for maintaining log files.

This is particularly true when you need to save the contents of sequential disk log files. Considerations that apply to sequential disk files are presented below, followed by considerations that apply to print and tape sequential log file assignments.

Considerations for disk log files

You can assign the system log to a single sequential file or to two alternate sequential files. At system startup time, the DC/UCF system writes to the beginning of the first (or only) log file.

Your strategy for maintaining log files depends on the actions that DC/UCF takes when the current log file becomes full, as discussed in the table below.

You archive sequential log files by using the appropriate operating system utility as follows:

- z/OS -- IEBGENER
- z/VSE -- DITTO
- z/VM -- COPYFILE command

Maintaining sequential disk files

The appropriate strategy for maintaining and archiving sequential disk log files depends on whether you log to single or alternate log files, as summarized in the following table:

<table>
<thead>
<tr>
<th>Sequential disk log</th>
<th>Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single File</td>
<td></td>
</tr>
</tbody>
</table>
Sequential disk log

Considerations
To save log records, you print or archive the log file when you shut down the system. When the log file becomes full at system runtime, DC/UCF wraps around to the beginning of the full log file and writes over old log records.

Alternate Files
To save the contents of a full log file, offload the file immediately. You can do this while the system logs to the alternate log file. When the alternate log file becomes full, DC/UCF switches back to the beginning of the old log file.

Considerations for other log files

You can assign the system log to a print device under z/OS and z/VSE. Additionally, you can assign the system log to a tape device under z/VSE by assigning the log SYSLST to the tape device in system-startup JCL statements. The following considerations apply to these log file assignments under z/OS and z/VSE:

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Log file considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS</td>
<td>When the log is assigned to a print device, the log is not output until the system is shut down.</td>
</tr>
<tr>
<td>z/VSE</td>
<td>When the log is assigned to a print or tape device by means of SYSLST, the log normally is not output until the system is shut down. When the log is assigned to a specific print device, the log file can be printed while the system is active as follows: When logging to a single file, DC/UCF closes the active log file when it is full and then immediately reopens the file. When logging to alternate files, DC/UCF logs to the alternate file automatically when the first log file is full. To save the old file's contents, print the contents of the log file immediately.</td>
</tr>
</tbody>
</table>

Accessing Logged Information

If you are logging to sequential files, you access logged information from the printed or archived log.

⚠️ Note: For more information, see Sequential Log Files.

If you are logging to the database (the DDLDCLOG area), you can access archived or online log information as described in the following table:

<table>
<thead>
<tr>
<th>Method</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLP (online PLOG)</td>
<td>Displays online the current contents of the log. This utility accesses the current, unarchived log in the DDLDCLOG area. Prints reports about all or selected parts of the current or archived log.</td>
</tr>
</tbody>
</table>
### How the System Logs Errors

DC/UCF reports on errors by recording in the system log messages, snap photos, snap dumps, system trace records, and information on DCMT commands issued at runtime.

A message is written to the log if the message destination specifies LOG. A snap photo contains a list of tasks that were active when the snap photo was requested, along with information on the resources held by the tasks.

A snap dump records the contents of memory at a particular time, such as when an abend occurs. System trace records are written if the system trace facility is enabled for DC/UCF by the SYSTRACE parameter of the system generation SYSTEM statement.

DCMT commands entered by users are recorded in the log. System administrators can use the #CTABGEN macro to keep all but significant DCMT commands from being sent to the log.

DC/UCF also records errors by writing to the operator’s console and/or to a user terminal messages issued by the system and by application programs at runtime, as specified in the message definitions.

#### Additional error information

Additional information on errors is written to the system log in the following situations:

- **Deadlock situations.**

- **Timeouts.**

- **#FREESTG violations.** These occur when a task issues a #FREESTG request for storage that has been violated (either by the issuing task or by another task). In this case, DC/UCF snaps the violated storage. The snap dump written to the log includes additional bytes of storage from the immediately preceding and immediately following areas. The message following the snapped storage indicates the starting address of the requested storage.

- **Abend conditions.** When a task is abended and an abend control element (ACE) exists for the abend, the ACE is written to the log. The ACE includes the following information:
The program status word (PSW) at the time of the abend; used to determine the point in the program at which the abend occurred. If the ACE does not contain a PSW, the ACE is written to the log without a PSW.

The Breaking Event Address (BEA). This address determines the last instruction in a program to cause a break in sequential instruction execution before the program check.

- Interrupt code
- Data at the PSW
- Next instruction address
- Contents of registers

**DCMT SNAP**

DCMT SNAP commands control the writing of snap dumps and snap photos to the DC/UCF log file. DCMT parameters are provided for the following:

- DCMT VARY PROGRAM and DCMT VARY TASK commands let you dynamically enable snaps for an individual program or task.
- System Generation SYSTEM parameters let you globally disable or enable snaps for system or task abends.

**More Information**

- For more information on the #CTABGEN macro, see the *Security Administering section*.
- For more information on error messages, see the *Messages section*.

⚠️ **Note:** You can review explanations of error messages online using the DCMT DISPLAY MESSAGE command.

### Applying Optional Functionality

Optional functionality is provided in one of the following ways:

- **Type 1** -- Can be activated by setting one bit in an optional functionality bitmap table. The bitmap table is generated in a new RHDCOPTF module that is loaded during startup processing and anchored in the CSA.

  ⚠️ **Note:** RI29610 lists all of the current RHDCOPTF bits that are available.

- **Type 2** -- Implements a new SYSIDMS parameter.
Type 3 -- Implements new syntax such as new SYSGEN syntax.

**Note:** No functionality is provided through optional APARs. If a PTF introduces new functionality, it is a required PTF. This functionality is controlled via one of these three methods. Any options previously provided via an optional APAR were converted to one of these three methods. For a list of optional APARs which were replaced, see the *Release Notes for Version 18.0.00*.

Optional functionality applied through any of these methods can be easily added or removed and can be preserved across maintenance updates.

### Creating an RHDCOPTF Module

**Contents**
- DEFOPTF Macro (see page 363)
- Example (see page 363)

To create a new RHDCOPTF module, assemble and link a RHDCOPTF source module that contains #DEFOPTF macros that will activate optional functionality.

#### DEFOPTF Macro

The #DEFOPTF defines which optional functionality is activated. It accepts the following parameters:

- One is positional and can be an internal function number, in the form OPTnnnnn, or a list of internal function numbers, allowing functions to be grouped by subject or module in one macro.
  
  An internal function number is associated with each optional function that can be activated this way.

- The second parameter is the TYPE=keyword that can get the value DEFINE, which is the default, or GENERATE, in the last input macro.

The last #DEFOPTF macro must contain the TYPE=GENERATE parameter in order to generate the code.

#### Example

```
TITLE 'User optional bitmap table'
#DEFOPTF OPT00002
#DEFOPTF OPT00010,OPT00011
#DEFOPTF (OPT00020,OPT00021,OPT00022)
#DEFOPTF TYPE=GENERATE
```

### CA IDMS in a Sysplex Environment

- Using Shared Cache (see page 364)
- Using Dynamic Database Session Routing (see page 371)
Using Shared Cache

CA IDMS exploits Coupling Facility technology by allowing multiple central versions (CVs) running in a Sysplex environment to share database buffers for one or more files using a shared cache. Each CV continues to maintain its own local copy of the buffers. CA IDMS maintains data consistency in the shared cache and across CVs and databases. Using Coupling Facility technology, data is accessed using less overhead than would otherwise be incurred doing a direct I/O to disk.

This section describes how CA IDMS supports and manages a shared cache and how to implement a shared cache in your CA IDMS systems running in a Sysplex environment.

- About Shared Cache (see page 364)
  - About Shared Cache Examples (see page 367)
- Deciding to Use Shared Cache (see page 367)
- Implementing Shared Cache (see page 368)
  - Defining Shared Cache to the Coupling Facility (see page 368)
  - Defining Shared Cache in CA IDMS (see page 369)
- Monitoring Shared Cache (see page 369)
- Tuning a Shared Cache (see page 369)
  - What You Can Do (see page 370)

About Shared Cache

You can run CA IDMS systems in a Sysplex environment for the purpose of sharing current data in database buffers across multiple CVs running in a Sysplex. Buffers are shared across CVs using a shared cache in a Coupling Facility.

What is a shared cache

A shared cache is basically a large, high-speed buffer in a Coupling Facility. It contains database pages from files assigned to the cache and accessed by the CVs running in a Sysplex.

Because a Coupling Facility can provide high-speed access to systems to which it is connected, accessing data from a shared cache is faster and uses less overhead than would otherwise be used doing a direct I/O to disk.

How shared cache works

CA IDMS manages one or more cache structures that contain the actual data and are used to manage the updates made to pages in the cache.
CA IDMS connects to a cache structure the first time a file assigned to it is opened. Disconnecting from a cache structure happens only at shutdown or as a result of an explicit DCMT VARY SHARED CACHE OFF command.

The following describes how update and retrieval access to the shared cache and local buffers works at runtime:

- When an update CV writes a page to disk, it also updates the page in the shared cache. This causes invalidation of that page in all other CVs that ever read the page. The following illustration shows how an update CV updates a shared cache and invalidates the updated page in all CV buffers in which it exists.

About Shared Cache

- When a CV needs to access a database page and shared cache is turned on for a file, a current copy of a page is obtained by performing one or more of the following:
  - If the page in the local buffer is still valid, it is retrieved.
  - If the page is not in the local buffer or is no longer valid, the CV looks for it in the shared cache.
  - If the page is found in the shared cache, it is retrieved.
  - If it is not found in the shared cache, it is read from disk and placed in both the shared cache and the local buffer.

Benefits

Using a shared cache in a Sysplex environment offers the following benefits:

- **Multiple CVs can share database buffers** -- CVs look for data in the shared cache before doing an I/O to disk.
• **Shared buffers contain current data** -- Using a buffer invalidation mechanism, the operating system informs the CV that a database page has been updated. This causes the CV to access a newer copy of the page the next time it is requested from the shared cache or disk.

• **The number of I/O operations to disk is reduced** -- Because access to data in the Coupling Facility uses high-speed fiber optic links, accessing pages from the shared cache greatly reduces the overhead that would be incurred by doing an I/O to disk.

**XA storage and shared cache**

All control blocks related to shared cache are allocated from XA storage.

The amount of storage needed depends on the following:

- The number of shared caches
- The size of the shared caches
- The database page size of the areas put in shared cache
- The load on the central version

The following two formulas can be used to compute an approximation of the amount of storage that will be used:

\[
\text{MAXCONC} \times \text{MAXPGSZ} \\
\text{RNDUP(\text{RNDUP}(\text{cache-size}/\text{MEANPGSZ}) / 818) * 16K}
\]

Formula one is global for the central version and independent of the number of shared caches in use. Formula two provides an approximation of the amount of storage needed for a given shared cache.

The following are the formula parameter descriptions:

- **RNDUP**
  A function that rounds up a value to the next higher integer.

- **cache-size**
  Size of the XES structure as allocated in the Coupling Facility. If there is no space constraint on the Coupling Facility, this matches the *init-size* of the structure definition in the Sysplex couple data set.

- **MEANPGSZ**
  The mean cache page size in the cache structure, rounded up to the next 256-byte boundary. Cache page size is the area page size rounded up to the next 256-byte boundary.

- **MAXCONC**
  The maximum number of tasks concurrently executing applications that use the shared cache.

- **MAXPGSZ**
  The highest value of the cache page size of all areas in the shared cache.
About Shared Cache Examples

The central version has 37 user maxtask and uses two shared caches as follows:

- CACHE1 has size 20000 K and is used by three areas (area1, area2, and area3.)
- CACHE2 has size 10000 K and is used by area4.

The areas contain the following number of pages:

- area1 contains 500000 pages, each 2932 bytes
- area2 contains 200000 pages, each 8192 bytes
- area3 contains 100000 pages, each 4000 bytes
- area4 contains 400000 pages, each 4276 bytes

This is the MEANPGSZ for CACHE1 (assuming each page has the same chance of getting accessed):
\[
\text{RNDUP}(\frac{(5\times3328 + 2\times8192 + 1\times4096)}{(5+2+1)}/256) \times 256 = 4864
\]

This is the amount of storage used for CACHE1:
\[
\text{RNDUP}(\text{RNDUP}(\frac{(20000\times1024)/4864}{818}) \times 16K) = 96K
\]

The MEANPGSZ for CACHE2 is:
\[
\text{RNDUP}(4276/256) \times 256 = 4352
\]

The amount of storage used for CACHE2 is:
\[
\text{RNDUP}(\text{RNDUP}(\frac{(10000\times1024)/4352}{818}) \times 16K) = 48K
\]

The MAXPGSZ is:
\[
\text{MAXPGSZ} = 8192
\]

The amount of XA storage that this CV will use for shared cache is:
\[
(37\times8K) + 96K + 48K = 440K
\]

Deciding to Use Shared Cache

Multiple CVs accessing same database

In general, a CA IDMS environment in which more than one CV requires access to the same database files can benefit from using shared cache.

One CV can benefit

A single CV can also benefit from using shared cache. If the CV is running on a machine with insufficient real memory to define large database buffers, you can use a shared cache to define a large buffer and reduce the number of I/Os to disk.
Shared cache used only by CVs

A shared cache is not accessible by applications running in local mode; it is available only to CVs.

Using more than one shared cache

You can define more than one shared cache in multiple Coupling Facilities to minimize possible contention. You can start with one shared cache and monitor its use. If your monitoring operations reveal contention for use of the cache, consider adding another cache. The following are considerations for defining more than one cache:

- Assign frequently accessed files to a separate cache
- Assign each database to a separate cache

Note: For more information on monitoring a shared cache, see Monitoring Shared Cache.

Dataspace caching

Dataspaces also provide a mechanism for caching database pages for a file. However, dataspaces cannot be shared across multiple CVs. It is recommended that you use only one caching mechanism for a file at a time (either shared cache or dataspace).

Implementing Shared Cache

To use the shared cache feature, you need to perform the following tasks:

- In the Coupling Facility, define the name of each shared cache that you will use in CA IDMS.
- In CA IDMS, do the following:
  - Identify the files that you want to assign to a shared cache.
  - Identify the name of the shared cache using a file override in the DMCL or dynamically using the DCMT VARY FILE SHARED CACHE command.

These tasks are described separately in the next sections.

Defining Shared Cache to the Coupling Facility

Estimating size of each shared cache

You must also define to the Coupling Facility the size of each shared cache that you will implement in CA IDMS. The estimate of the size of a shared cache is based on the following:

- Number of files that you will assign to the cache
- Size of the files you will assign to the cache
Estimating the size of a shared cache is dependent upon the applications that access it. In general, it should be large enough to accommodate all frequently used pages of all files in the cache.

**Note:** For more information on defining the size of a structure, see the IBM manual, *Setting Up a SYSPLEX.*

**If a cache size is too small**

If the size of a shared cache is too small, the Coupling Facility deletes the least recently used (LRU) page(s) when a new one must be added to the cache. This results in extra CPU and disk I/O, which you want to avoid.

**Defining Shared Cache in CA IDMS**

To implement shared cache in CA IDMS, you need to assign the files that will participate in a cache to a shared cache. You can assign a file to a shared cache by specifying the name of a shared cache.

You implement shared cache for a CV through:

- Definition in the DMCL. For more information, see the *Database Administering section.*
- DCMT VARY AREA, DCMT VARY FILE, or DCMT VARY SEGMENT commands. For more information, see the *System Tasks and Operator Reference section.*

**Monitoring Shared Cache**

You can use the following CA IDMS tools to monitor the use of shared cache by an executing CV:

- DCMT DISPLAY commands. For more information, see the *System Tasks and Operator Reference section.*
- CA IDMS Performance Monitor. For more information, see the *Performance Monitor System Administering section.*
- CA IDMS statistic reports. For more information, see the *Reporting section.*

You can also use monitoring tools for the Coupling Facility to monitor the global use of a shared cache. For more information, see the IBM Coupling Facility documentation.

**Tuning a Shared Cache**

Tuning a shared cache involves monitoring its usage and, as necessary, modifying its definition and implementation to use it as efficiently as possible.
What You Can Do

You can make the following changes to tune your use of shared cache:

- Change the shared cache status for a file and either assign it to a new or existing cache or drop it from participating in a shared cache
- Define another shared cache and assign files to it
- Increase the size of an existing cache

Changing the cache status for files

To change the shared cache status for a file, use the DCMT VARY command for an AREA, FILE, or SEGMENT.

⚠️ Note: For more information on DCMT commands, use the DCMT HELP command or see the System Tasks and Operator Reference section.

Defining multiple shared cache

As you monitor the use of a shared cache, you may notice contention for it and may want to define another shared cache. For example, if you find that the FND-IN-CACHE statistic is considerably lower than the NUMBER-READS statistic for frequently accessed files in a cache, you might consider defining another shared cache and assigning these files to it.

⚠️ Note: For more information on monitoring a shared cache, see Monitoring Shared Cache.

Increasing the size of a shared cache

If you determine that the size of a shared cache is insufficient for your processing needs, you can increase the size of it. Before you do this, be sure this is the right solution for your needs.

While you can increase the size of an existing cache, it requires that you perform the following steps:

1. Issue a DCMT VARY SHARED CACHE cache-name OFF command for all CVs
2. Increase the size of the cache structure
3. Issue a DCMT VARY SHARED CACHE cache-name ON command for all CVs
Using Dynamic Database Session Routing

You can use the dynamic database session routing feature to dynamically select the node to which requests for data are sent. The selection is made by determining which CV running in the Sysplex has the CPU cycles available to service the request. You can also use the CV cloning feature to run cloned copies of the same CV. This will make more CVs available to process database sessions.

- About Dynamic Database Session Routing (see page 371)
- Planning to Use Dynamic Database Session Routing (see page 372)
- Implementing Dynamic Database Session Routing (see page 373)
  - Using DBGroups (see page 373)
  - Back-end CV Definitions (see page 373)
  - Front-end CV Definitions (see page 374)
  - How Dynamic Database Session Routing Works (see page 375)
  - Coupling Facility Considerations (see page 376)
  - Application Considerations (see page 377)
- Managing Dynamic Database Session Routing (see page 378)
- Monitoring and Tuning Dynamic Database Session Routing (see page 378)

Using these features, you can dynamically balance your CA IDMS workload to provide increased transaction throughput, better response time, and increased system availability. This section describes how to implement dynamic database session routing and create cloned CVs in CA IDMS systems running in a Sysplex environment.

About Dynamic Database Session Routing

The CA IDMS client/server communications architecture for database sessions is extended to allow a front-end CV to route a database session to a specific back-end CV that has been identified, at runtime, to have the CPU cycles available to service the session.

In a non-Sysplex environment, the routing of database sessions is static; database sessions are routed to an explicit, predetermined CV regardless of the availability of processing cycles.

Benefits

Dynamic database session routing provides the following benefits:

- **Dynamic workload balancing** -- Workload balancing is dynamic and based on actual system load and resource availability; you don't need to predetermine database routing to balance workloads across CVs to get maximum throughput and shorter response time. Database sessions are routed to a CV with available processing cycles.

- **Parallel processing of database sessions** -- Database sessions are processed in parallel to reduce elapsed processing time.
• **Automatic routing of database sessions to an available CV** -- Database sessions are routed to an available CV instead of being routed to the same system whether its available or not.

• **Use with cloned CVs to make multiple copies of systems available** -- You can implement dynamic database session routing with the cloned CV feature to start and stop systems in response to changes in workloads to increase transaction throughput and decrease response time.

**Use with CV cloning**

This feature is designed to work in a Sysplex with the dynamic routing of database sessions feature so that database sessions can be dynamically routed to cloned copies of the same CV.

⚠️ **Note:** For more information on cloning CVs, see CV Cloning. To dynamically route update database sessions, back-end CVs must be members of a data sharing group.

⚠️ **Note:** For more information on data sharing, see Data Sharing.

**Use with shared cache**

You can also use the dynamic database session routing feature with the shared cache feature to share database buffers across multiple CVs to minimize I/O operations to disk and to keep data current across the CVs running in a Sysplex.

**Planning to Use Dynamic Database Session Routing**

Dynamic routing of database sessions is designed to work with CA IDMS environments configured with the following minimum requirements:

- CVs set up to process terminal and application services separate from database services. This is accomplished by defining front-end CVs to process terminal activities and other application-specific services, and back-end CVs to process database requests. Using this configuration, front-end CVs can route database requests to an available back-end CV where requests can then be serviced.

- Back-end CVs set up as data sharing group members that have update access to the database(s) used by the dynamically routed sessions.

- Back-end CVs set up to process retrieval applications separate from update applications. With this set up, multiple CVs can process retrieval requests for a database while one designated CV can process update requests at the same time. In that case, the front-end CVs must have the ability to identify database sessions that perform only retrieval processing against a database. Typically you do this for database run units by assigning a retrieval-only subschema to retrieval run units. Since updates to an area can be performed by only one CV at a time when not using data sharing, dynamic routing is restricted to retrieval database sessions.
If your current CA IDMS environment is configured as described previously, you need to make only a few changes to use dynamic database session routing.

Implementing Dynamic Database Session Routing

To implement dynamic database session routing, you assign back-end CVs that service database sessions to one or more DBGroups. At runtime, database requests are routed to a DBGroup to determine which CV in the DBGroup has the CPU cycles available to process it. Once a CV volunteers to service a request, CA IDMS sends the request to it and it processes the request on the identified node.

Implementing dynamic database session routing involves making changes to the following:

- Back-end CVs
- Front-end CVs
- Coupling Facility

There are also considerations for applications that will participate in dynamic database session routing.

This section describes what you need to do in each of these areas to set up your CA IDMS environment to use dynamic database session routing, beginning with the concept of a DBGroup and how it is used to implement dynamic database session routing.

Using DBGroups

What is a DBGroup

A DBGroup contains one or more CVs. You assign CVs to DBGroups based on the databases they service. At runtime, a request for access to a database is routed to a DBGroup, instead of a node. The DBGroup name is then replaced with the node name for a CV assigned to it that volunteers (i.e. has the CPU cycles) to service the request.

Planning DBGroups

Before you define any DBGroups, you need to plan how you want to group CVs to meet your processing requirements. For example, create one DBGroup for all back-end CVs, if all databases are accessible by all CVs.

Back-end CV Definitions

Back-end CVs must be assigned to a DBGroup

To assign each back-end CV to one or more DBGroups, you update its database name table and add a DBGROUP statement for each DBGroup in which it will participate. This makes the CV a member of the specified DBGroup. You must assign all back-end CVs to all DBGroups in which they will participate. When each back-end CV is started, it will be eligible to process dynamic database sessions for the databases defined in its database name table.

Using cloned back-end CVs
To exploit the parallel processing and dynamic workload balancing features of dynamic database session routing, you must have enough back-end CVs available to process any run unit or SQL transaction that may be dynamically routed to it. You can use the CV cloning feature, as discussed in section CV Cloning, to make multiple copies of CVs available.

**Assigning back-end CVs to DBGroups**

You use the CREATE DBGROUP statement to assign a back-end CV to a DBGroup. To assign a back-end CV to multiple DBGroups, include multiple CREATE DBGROUP statements. For more information on these statements, see the *Database Administering section*.

**Front-end CV Definitions**

In dynamic database session processing, front-end CVs route requests for database services to DBGroups. To do this, it must know about the DBGroups to which it can route requests. DBGroups are identified as nodes in a front-end CV’s resource name table using the system generation NODE statement.

**Adding DBGroups to a resource name table**

The resource name table for each front-end CV must contain the following:

- The name of each DBGroup to which it can route requests. The DBGroup name is added as a node name on the system generation NODE statement with a type of GROUP.

- Optionally, for each back-end CV that can service requests for a DBGroup, a NODE entry specifying the communication method to use to access it. Use the system generation NODE statement to define the node by which a back-end CV is accessed and the communication method used, if it is not already defined. If no NODE entry is defined, the access method will be chosen as follows:
  - If the two systems are executing on the same z/OS image and their system definitions specify the same SVC number, then the SVC access method will be used.
  - If the back-end system has a VTAM line driver, then the VTAM access method will be used.
  - If the back-end system has a CCI line driver, then the CCI access method will be used.

**Note:** Applications executing from CICS, batch, Windows, and any other non-CA IDMS/DC client must first be routed to an explicit CV (static routing), which can then dynamically route it to a back-end CV for servicing.

**Making enough front-end CVs available**

You must have a sufficient number of front-end CVs available to process the applications submitted by your users.

**Using the NODE statement to identify DBGroups**
You use the system generation NODE statement to identify DBGroups in the resource name table for front-end CVs. For more information on the NODE statement, see the *CA IDMS Administrating section*.

**How Dynamic Database Session Routing Works**

This section describes how a sample DBGroup is defined and used at runtime.

In the following example, the EMPGROUP contains CVs IDMS060, IDMS070, and IDMS080, which have update access to the EMPDB database. They are members of data sharing group DSGROUP1. At runtime, front-end CV IDMS050 can route requests for the EMPGROUP to either IDMS060, IDMS070, or IDMS080, whichever one volunteers. If neither IDMS060, IDMS070, nor IDMS080 are enabled members of the EMPGROUP, then requests fail. :p

**Modifying database name table**

The following example illustrates how the EMPGROUP DBGroup is defined to the existing system definitions for the IDMS060, IDMS070, and IDMS080 back-end CVs and the IDMS050 front-end CV.

*DBGroup definition in database name table for IDMS060*

```sql
CREATE DBGROUP N60TABLE.EMPGROUP
ENABLED;
GENERATE DBTABLE N60TABLE;
```

*DBGroup definition in database name table for IDMS070*

```sql
CREATE DBGROUP N70TABLE.EMPGROUP
ENABLED;
GENERATE DBTABLE N70TABLE;
```
**DBGroup definition in database name table for IDMS080**

```sql
CREATE DBGROUP N80TABLE.EMPGROUP
ENABLED;
GENERATE DBTABLE N80TABLE;
```

**Modifying resource name table**

```sql
MODIFY SYSTEM IDMS050.
ADD NODE EMPGROUP
GROUP DEFAULT NODE IDMS080.
GENERATE.
```

**How EMPGROUP is used at runtime**

At runtime, using either the resource name table or user exit 23, requests for database services are identified to be serviced by node EMPGROUP using an access type of GROUP.

**Note:** Exit 23 may be used to override the specified node name on the bind.

The access type of GROUP directs CA IDMS to solicit a back-end CV in the EMPGROUP to service the database request. An available CV in the EMPGROUP DBGroup volunteers to service the request. From this point on, processing takes place as it normally does in CA IDMS using the node name of the CV that volunteered to service the database request. A list structure is used in the Coupling Facility to determine which back-end CV will be the volunteer.

**Coupling Facility Considerations**

You need to define one Coupling Facility list structure for each DBGroup defined to CA IDMS with the name "CAIDMS" appended with the DBGroup name, for example, CAIDMSEMPGROUP.

**Sizing the list structure**

Use the following formula for estimating the size of the list structure:

\[ \text{Size} = \text{ROUND1M} (\text{TotTask} \times 365 + \text{MinimumSize}) \]

Where:

- **TotTask** -- is the sum of (MAXIMUM TASKS + MAXIMUM ERUS) in the SYSTEM statement for all front-end systems.
- **MinimumSize** -- is the minimum size of the structure. The minimum size depends on the CFLEVEL of the Coupling Facility. For CFLEVEL=12, use a MinimumSize of 2 megabytes.

For example, suppose the EMPGROUP group is defined to three front-end CVs, and each CV has its MAX TASKS parameter set to 40 and MAX ERUS 10. The size of the list structure can be estimated as 3 megabytes (150 x 365 + 2M, rounded up to a 1 megabyte multiple).
Application Considerations

A database session is routed to a node using the DBNAME and NODENAME passed on the BIND statement for run units and the dictionary name passed on the CONNECT statement for SQL transactions. Dynamic database session routing is used if the DBNAME or dictionary name maps to a node name in the resource name table with a type of GROUP. You can tailor the node name at runtime using the user-written, pre-bind exit Exit 23 or using the VIA parameter on both the DBNAME and the DESTINATION clauses of the RESOURCE TABLE system generation statement.

Specifying DBGroup name on RESOURCE TABLE statement

You can map a DBNAME or DESTINATION-NODE to a DBGroup name using the VIA parameter on the RESOURCE TABLE syntax. Specify a DBGroup name as the nodename on the VIA parameter to route the application to a DBGroup. In the following example, the EMPDB DBNAME is routed to DBGroup EMPGROUP by specifying EMPGROUP as the nodename. In this case, database requests from applications binding to EMPDB will be routed to the EMPGROUP to determine which CV assigned to it has the CPU cycles available to process the request. The request is then directed to the node associated with the CV that volunteered to service the request.

MODIFY SYSTEM IDMS071
MODIFY RESOURCE TABLE
DBNAME IS EMPDB VIA EMPGROUP.
GENERATE.

Impact of long term database resources passed across run units

If there are longterm database resources maintained for an application thread running on a back-end CV, and subsequent BINDS or CONNECTS are issued from that application to the same DBGroup, they are automatically routed to the same back-end CV. If longterm resources are frequently held across database sessions within an application thread, the overall effectiveness of the work load balancing will be impacted.

Long term database resources are either database currencies saved by CA ADS applications or long term locks created by any KEEP LONGTERM database commands. All other application resources are saved on the front-end CV and do not impact the selection of the back-end CV for the database session.

Use of the NOSAVE option on DISPLAY, INVOKE, and LINK commands within a CA ADS application eliminate the saving of currency blocks across a pseudo converse or non-extended run unit. If the NOSAVE option is used, each run unit in the application thread can be dynamically routed (provided longterm locks are not used). It does require that the application logic be in place to reestablish database currencies, if necessary, when the next run unit is bound.

Retrieval and update run units in same application

If currencies or longterm locks are passed from one run unit to another, then all such run units must be routed to the same CV once a back-end CV has volunteered for the initial bind. CA IDMS will ensure this, provided that they are routed to the same DBGroup. Also, if a retrieval-only CA ADS
dialog passes currencies to an update dialog, then both must be routed to the same DBGroup if dynamic routing is to be used. In this case, the DBGroup back-end CVs must be part of a data sharing group.

Managing Dynamic Database Session Routing

You can manage a CV's participation dynamically using the DCMT VARY DBGROUP command.

**What you can do**

You issue the DCMT VARY DBGROUP command to activate and inactivate dynamic database session routing and to manage a CV's participation in a DBGroup. The tasks you can perform are summarized in the following table:

<table>
<thead>
<tr>
<th>To do this</th>
<th>Use these DCMT VARY DBGROUP parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable and disable dynamic database session routing on an executing front-end CV</td>
<td>ACTIVE/INACTIVE</td>
</tr>
<tr>
<td>Join a CV to a DBGroup or disable it from a DBGroup</td>
<td>JOIN/LEAVE</td>
</tr>
<tr>
<td>Activate dynamic database session routing and enable the CV to participate in the named DBGroup or inactivate dynamic database session routing and disable the CV from participating in the named DBGroup</td>
<td>ON/OFF&lt;br&gt;ON is the same as using the ACTIVE and JOIN parameters. OFF is the same as using the INACTIVE and LEAVE parameters.</td>
</tr>
</tbody>
</table>

Monitoring and Tuning Dynamic Database Session Routing

Using DCMT commands and the CA IDMS Performance Monitor, you can monitor how database sessions are being serviced by dynamic database session routing, and as appropriate, modify parameters to suit your processing needs.

The tools you can use are summarized in the following table:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCMT DISPLAY DBGROUP</td>
<td>You can display statistics for all DBGroups to which the currently executing CV can direct requests or for a specific DBGroup. Statistics include the number of active back-end CVs assigned to DBGroups and the total number of requests processed by each.</td>
</tr>
<tr>
<td>DCMT DISPLAY DBTABLE</td>
<td>Lists each DBGroup defined in the table and its status.</td>
</tr>
<tr>
<td>DCMT DISPLAY NODE</td>
<td>Displays the name of each node with its associated type.</td>
</tr>
</tbody>
</table>
DCMT VARY DBGROUP
Allows you to vary the status of a DBGroup and turn dynamic database session routing on and off.

LOOK DBTABLE (DC)
Lists each DBGroup defined in the table and its status.

IDMSLOOK DBTABLE (batch)
Lists each DBGroup defined in the table and its status.

Interval Monitor (online and batch)
For each DBGroup, displays detailed statistics (using DBGROUP category) and wait statistics (using SUMMARY HISTORY, SUMMARY DETAIL, and WAIT Screens).

Merging CA IDMS statistics
If you use CA IDMS statistics for job accounting purposes, merge the statistics for all cloned systems prior to analyzing them. This will give you the proper global view of the actual resources used.

Note: For more information on CV cloning, see CV Cloning.

Data Sharing
By exploiting Coupling Facility technology, CA IDMS allows you to do the following:

- Update data concurrently from multiple CA IDMS systems
- Share queues and enqueue common resources across CA IDMS systems
- Broadcast system tasks to multiple CA IDMS systems

Each of these features depends on the implementation of a data sharing group that individual CA IDMS systems join as members. A data sharing group is a construct that enables the implementation of the above features.

The rest of this section describes what a data sharing group is and how it is defined, and gives details on each of the sysplex exploitation features it uses.

Contents
- Data Sharing Groups (see page 380)
- Designing Data Sharing Groups (see page 380)
  - Types of Groups (see page 381)
  - Data Sharing Group Versus DBGroup (see page 384)
Defining Data Sharing Groups (see page 384)
- Selecting a Group Name (see page 384)
- Configuring the Coupling Facility (see page 385)
- Specifying Group Membership (see page 388)
- Sharing Update Access to Data (see page 389)
  - Shared Area Requirements (see page 389)
  - Notify Locking Considerations (see page 390)
  - Enabling Data Sharing (see page 390)
  - Altering the DMCL Definition (see page 391)
- Member Failure (see page 393)
- Coupling Facility Failures (see page 393)
- Group Restart (see page 395)
- Accessing Unrecovered Data (see page 395)
- Sharing Queues and Enqueued Resources (see page 396)
  - Sharing Queues (see page 396)
  - Sharing Enqueued Resources (see page 397)
- Monitoring Data Sharing Groups (see page 398)
  - Monitoring Through DCMT Commands (see page 399)
  - Monitoring Through Performance Monitor (see page 401)
  - Monitoring Through Journal Reports (see page 401)

Data Sharing Groups

A data sharing group is a named collection of CA IDMS systems within a sysplex. Each CA IDMS system that is associated with a data sharing group is referred to as a member of that group. The name of the member becomes both the CA IDMS system name and the node name for that system. A system can be a member of only one data sharing group at a time.

Data sharing groups allow you to do the following:

- Share update access to a database. For more information, see Sharing Update Access to Data.
- Broadcast commands to all members of the group. For more information, see the CA IDMS System Tasks Operator Reference section.
- Share queues and enqueue common resources. For more information, see Sharing Queues and Enqueued Resources.
- Monitor and report on all members of a data sharing group. For more information, see Monitoring Data Sharing Groups.

Designing Data Sharing Groups

The most significant benefit to a data sharing group is the ability for multiple CA IDMS systems to update the same data concurrently. Satisfying your data sharing needs should therefore be the most significant consideration in designing your data sharing groups.
Types of Groups

There are three basic types of groups: a homogeneous group, a heterogeneous group and a hybrid group.

A homogeneous group is one in which all members are essentially the same. They support the same applications, they access the same databases in the same way, and they have the same security definitions. Members of a homogeneous group likely share the same system definition, using the cloned system capability. Since every member has access to the same resources, it doesn't matter on which member a given transaction is executed.

In a heterogeneous group, every member is unique in terms of the applications that it supports. Although some databases may be shared between members, other databases are unique to a given member. Each member may have its own system and security definitions. They are members of the same group in order to share update access to data, but otherwise are distinct systems. A given transaction must be directed to a particular member in order to ensure that it has access to the resources that it needs.

A hybrid group is one in which some members are clones of one another while other members are not.

Homogeneous groups

A homogenous group can be thought of as a multi-part IDMS system. Facilities such as shared update access to data and shared queues enable applications to execute on any member of the group. The major benefits to this type of group are as follows:

- The ability to adjust the number of members in response to changing workloads
- Fault tolerance in the event of a failure

If a group is reaching capacity in terms of transaction volumes, then an additional member can be started to handle some of the workload. Members might be added and removed on a periodic basis (daily, monthly, etc.) Or members may be permanently added as workload increases over time.

Homogeneous groups also provide fault tolerance in the event that a CA IDMS system or a z/OS image fails. Other members of the group (which may be executing on different z/OS images) can continue to process applications, while recovery is taking place.

The degree to which applications update the same records within the database will determine the effectiveness of a homogeneous group. For example, if every transaction must update a one-of-a-kind control record, then there will be a high degree of contention for that record across members of the group. This will significantly increase the CPU overhead needed to process the transactions, since the members must communicate with one another in order to resolve the contention. Furthermore, if a CA IDMS system fails while it holds an exclusive lock on the control record, then that record remains locked until the failing system has been recovered, thus preventing other transactions from accessing the record. If, on the other hand, transactions tend to access and update different records and pages in the database, with only the occasional overlap, then a homogeneous group should perform well and provide increased fault tolerance. In designing a homogeneous group, consider ways to segment the workload to minimize cross-member contention for resources.
The following diagram illustrates a homogeneous data sharing group. It consists of four members (CUST01, CUST02, CUST03, and CUST04), each of which share update access to the same set of databases (Inventory, Customer and Financial).

![Diagram of homogeneous data sharing group]

**Types of Groups**

**Heterogeneous groups**

A heterogeneous group provides the ability to share update access to certain areas of the database from otherwise distinct CA IDMS systems. It eliminates the need for other solutions such as application-level replication of updates or remote database access. It can alleviate the pressures of an increasing workload, by allowing it to be split along application boundaries even though some areas need to be commonly updated.

While contention for individual records and pages within shared areas may not be as likely in a heterogeneous group, the degree to which such contention occurs will affect performance. This should be one of the considerations in determining how to split your workload across members of a heterogeneous group.

Heterogeneous groups provide a degree of fault tolerance in that members that have not failed will continue to process transactions, some of which may update shared areas. Of course there is no fault tolerance for the portion of the workload that was being processed by the failed member.

The following diagram illustrates a heterogeneous group in which every member accesses a distinct set of databases. Member CORP updates the Financial and HR databases. Member CUST updates the Customer, Financial and Inventory databases. Member INV updates the Inventory and Financial databases.
Types of Groups (2)

Hybrid groups

A hybrid group can provide the benefits of both homogeneous groups and heterogeneous groups. It easily allows the addition of members to handle increased workload and provide fault tolerance for certain applications while allowing other members to continue processing different applications that have little contention for shared databases.

The following diagram illustrates a hybrid group that is similar to the heterogeneous group described above except that the CUST member is cloned so that two members CUST01 and CUST02 service the same set of transactions.
Data Sharing Group Versus DBGroup

There is a difference between a data sharing group and a database or DBGroup. A data sharing group provides the ability to share update access to data. A DBGroup provides the ability to dynamically route database requests. At any one time, a CA IDMS system can be a member of several DBGroups but only one data sharing group.

While it is possible that every member of a data sharing group also is a member of the same DBGroups, this is not a requirement and in fact, is unlikely unless the data sharing group is homogeneous. It would be more likely that members that are clones of one another would be members of the same set of DBGroups, since they can support the same types of transactions.

It is also possible for a DBGroup to span data sharing groups. CA IDMS systems that are members of different data sharing groups or that are not members of any data sharing group can belong to the same DBGroup. While such a scenario may be unlikely, it highlights the fact that data sharing groups and DBGroups are technically unrelated.

Defining Data Sharing Groups

There is no explicit definition for a data sharing group; however, before a CA IDMS system can become a member of a group, the following actions must be taken:

- The number of XCF groups that can be created in the coupling facility may need to be increased
- The number of buffers used for XCF messaging may need to be increased
- A list and lock structure must be defined to the coupling facility
- Each CA IDMS system that is to be a member of a group must be associated with that group through its startup JCL
- Print task codes defined through the $UCFUFT macro must have a corresponding entry added to the LCLENQDQ module. This topic is discussed in Sharing Queues and Enqueued Resources later in this section
- If members of the group are to share update access to data, changes must be made to the DMCLs of all group members. For more information, see Sharing Update Access to Data.

Selecting a Group Name

A data sharing group internally corresponds to an XCF group whose name is that of the data sharing group. The name of a data sharing group must be different from the name of any other XCF group within the sysplex.

Group names may be 1-8 characters in length and consist of characters A-Z, 0-9, $, # or @. Names that begin with SYS or UNDESIG are reserved and cannot be used. Names that begin with A-I may be used by the operating system and should be avoided. For more information, see the appropriate IBM documentation.
Configuring the Coupling Facility

Each data sharing group internally uses an XCF group and requires the definition of a list and a lock structure in the coupling facility. In addition, one or more cache structures must be defined if the data sharing group is to share update access to data or if shared buffering is to be used.

XCF group

XCF groups are not explicitly defined to the coupling facility, however the maximum number of XCF groups that can be active is part of CFRM policy. This maximum may need to be increased in order to implement a new data sharing group.

In addition, CA IDMS uses XCF messaging to communicate between group members. It may be necessary to increase the number of buffers available for XCF messaging in order to accommodate the increased traffic. For more information, see the appropriate IBM documentation.

CF structures

The following information must be specified when defining a structure to the coupling facility:

- Structure name
- Structure size

List structure

The name of the list structure must be as follows:

CAIDMS<groupname>LI

where groupname is the name of the data sharing group.

Use the following formula for estimating the size of the list structure:

Size = ROUND1M(SAREALS + SFILELS + OS*QUEUES + QUIESLS + MINSZ)

Where:

- SAREALS is the size of the shared area list. Compute its size as follows:
  
  Compute for each shared area the area-element-size:
  
  ROUND256(184 + number-of-files-in-area * 35) + 97
  
  Sum all area-element-sizes.

- SFILELS is the size of the shared file list. Compute its size as follows:
  
  Compute for each shared file the file-element-size:
  
  ROUND256(164 + number-of-areas-in-file * 6) + 97
  
  Sum all file-element-sizes.
• $QS = 1$ if the queue area is shared; otherwise it is 0.

• $SQUEUES$ is the size of the shared queue list. Compute its size as follows:

$$\text{Number-of-shared-queues} \times 353$$

• $QUIESCS$ is the size of the quiesce list. It is used when a DCMT QUIESCE command affects one or more shared areas. Its size is dependent on the number of parallel outstanding requests. The size of an outstanding request can be computed as follows:

$$\text{ROUND256}(92 + \text{number-of-target-shared-areas} \times 6) + 97$$

• $MINSZ$ is the minimum size of the structure. The minimum size depends on the CFLEVEL of the Coupling Facility. For CFLEVEL=12, use a MINSZ of 2 megabytes.

⚠️ **Note:** ROUND1M means rounding up to the next 1 megabyte multiple. ROUND256 means rounding up to the next 256-byte multiple. For example: 198 Rounds to 256. ROUND256(198) = 256

ROUND256(256) = 256 ROUND256(258) = 512

The size of a list structure may be altered at any time using an operating system command. The ability to increase the size of the list structure is limited by the available space in the coupling facility. CA IDMS supports system-managed rebuild of the list structure. For more information, see System-Managed Rebuild.

### Lock structure

The name of the lock structure must be:

$CAIDMS\text{\textunderscore groupname}\text{\textunderscore LK}$

where $\text{\textunderscore groupname}$ is the name of the data sharing group.

The minimum size of the lock structure depends on the number of concurrent locks that will be placed on records in shared areas.

A coupling facility lock structure contains two types of information: a lock table and record data entries.

The lock table is used as a hash table for the purposes of detecting contention for a resource. Its size is determined by the LOCK ENTRIES and the MEMBERS parameters specified in the DMCL of data sharing group members.

⚠️ **Note:** For more information on these parameters, see Enabling Data Sharing.
Record data entries are used to hold information on exclusive locks. Each exclusive global lock on a transaction resource has a corresponding record data entry. Applications that update many records before issuing a commit will increase the requirement for record data entries. If the system runs short of record data entries, it will react by releasing proxy locks that are not in use. This will have a negative impact on performance. If, after taking this action, there are not enough record data entries available to satisfy a lock request, the issuing task will fail.

For more information on transaction locking, see the Database Administering section. For more information on monitoring the available record data entries in the lock structure, see Monitoring Data Sharing Groups.

The size of a lock structure must be large enough to hold both the lock table and the maximum number of record data entries that will exist at one time. The first CA IDMS system that starts as a member of a data sharing group determines the size of the lock table. If the lock structure is too small to accommodate the lock table, startup will fail. Once the lock table has been allocated, its size remains the same until all group members have been shut down normally.

Whatever space remains in the lock structure after the lock table has been allocated is used for record data entries. The amount of space in the lock structure can be increased (or decreased), by using the SETXCF START ALTER command. The ALTER command only affects the amount of space available for record data entries. Task abends will result if there is not enough space to create a record data entry when one is needed. The amount of available space in the lock structure can be monitored by using the DCMT DISPLAY DATA SHARING command. For more information, see Monitoring Data Sharing Groups.

The minimum size of a structure depends on the CFLEVEL of the Coupling Facility. For CFLEVEL=12, use a minimum of 2 megabytes. This value may need to be significantly increased if areas are to be shared for update access. Use the following formula for estimating the size of the lock structure:

\[
\text{Size} = \text{LTE} \times \text{LTES} + \text{RD} \times 140 + 35K
\]

where:

- \(\text{LTE}\) is the number of lock table entries
- \(\text{LTES}\) is the lock table entry size as determined from the following table
- \(\text{RD}\) is the maximum number of record data entries needed

<table>
<thead>
<tr>
<th>Maximum Number of Members in Data Sharing Group</th>
<th>Lock table entry size (LTES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8-23</td>
<td>4</td>
</tr>
<tr>
<td>24-55</td>
<td>8</td>
</tr>
<tr>
<td>56-119</td>
<td>16</td>
</tr>
<tr>
<td>120-247</td>
<td>32</td>
</tr>
</tbody>
</table>

CA IDMS supports system-managed rebuild of the lock structure. For more information, see System-Managed Rebuild.

Cache structures
At least one cache structure is needed in order to share update access to data or to enable shared buffering. Both of these capabilities require that each affected CA IDMS file be associated with a cache structure. You may associate any number of files with a single cache or allocate a cache for each file.

The larger the cache structure, the more likely a page will remain in the cache, eliminating a disk access when the page is next needed by a system. Ideally, each file would be assigned to its own cache structure and that structure would be large enough to hold the entire file. However, this may not be practical due to the amount of space and the number of structures that would be needed. The more space that can be assigned, the fewer disk accesses that will be needed and hence the better the performance.

The size of a cache structure may be altered at any time using the SETXCF START ALTER command. The size may be increased, provided there is sufficient space in the coupling facility.

There are no additional requirements for the name of a cache structure beyond those imposed by the operating system.

CA IDMS supports system-managed rebuild of the cache structure. For more information, see System-Managed Rebuild.

Note: The IBM documentation on Coupling Facility code levels states: "When migrating to a higher CFLEVEL, structure sizes might need to be increased." This is particularly important for small structures of only a few megabytes. For such structures, the size computed with the above formulas might need to be increased by one or two megabytes.

Specify ALLOWAUTOALT=YES in the CFRM definition for the shared cache. The ALLOWAUTOALT parameter controls the "automatic alter" function of the Coupling Facility structure. The "automatic alter" dynamically monitors and alters Coupling Facility structure sizes and ratios for optimum performance. At the time of this writing, IBM APAR OW50397 is required when using ALLOWAUTOALT (YES). For more information on ALLOWAUTOALT, see the IBM manual, Setting up a Sysplex. For more information on shared cache, see Using Shared Cache (see page 364).

Specifying Group Membership

Each CA IDMS system that is to be a member of a data sharing group must specify the name of the group and its membername in the SYSIDMS card image file in the startup JCL.

Syntax

```
DSGROUP=group-name
DCNAME=member-name
```

Parameters

- **DSGROUP=group-name**
  Specifies the name of the data sharing group of which this system is a member. All CA IDMS systems that are members of the same group must specify the same group name.
- **group-name**
  Must be a 1-8 character name consisting of characters A-Z, 0-9, $,# or @. Names that begin with SYS or UNDESIG are reserved and cannot be used. Names that begin with A-I may be in use by the operating system and should be avoided.

- **DCNAME=member-name**
  Specifies the member name of the system within a data sharing group. This name also becomes the system (node) name, overriding the value specified in the system definition.

- **member-name**
  Must be a 1-8 character name consisting of characters A-Z, 0-9, $,# or @.

**Usage**

**Specifying a group name**

All CA IDMS systems that specify the same group name are members of the same data sharing group and are therefore capable of sharing update access to data as well as exploiting other features associated with data sharing groups.

**Specifying a member name**

Member names must be unique within a data sharing group. Member names should also be unique across your environment since the member name becomes the system (node) name.

**Changing group and member names**

Once a CA IDMS system has become a member of a data sharing group, it remains a member until the system has been shut down normally. This means that after an abnormal termination, the CA IDMS system must be restarted with the same group and member name as at the time of failure. This also means that a failed group member cannot be restarted without a group and member name.

Similarly, a CA IDMS system that was not a member of a data sharing group can become a member only if the system had previously terminated normally.

**Sharing Update Access to Data**

The ability to share update access to data is referred to as *data sharing*. It allows multiple CA IDMS systems to update specified areas of the database concurrently.

For data sharing to occur, each CA IDMS system that shares update access must be a member of a data sharing group. Only members of one data sharing group can have update access to a shared database area at one time. Other data sharing groups, other CA IDMS systems that are not members of the group and local mode IDMS applications can access the area in retrieval mode only. For more information on data sharing, see Data Sharing Groups.

**Shared Area Requirements**

For an area to be eligible for data sharing, the following attributes of the area and associated files must be identical in all sharing systems within a group:

- Page range, page group and number of records per page
CA IDMS - 19.0

- Segment and area names
- Page size
- File mappings
- IDMS file names
- DSNAME and VOLSER of the associated disk files

Additionally:

- A shared area cannot be native VSAM.
- An area that is part of a dictionary controlled by CA Endevor/DB cannot be shared.
- No two shared areas within a data sharing group can have overlapping page ranges within a page group.
- Within a data sharing group, the combination of DSNAME and VOLSER must be unique for all IDMS files associated with shared areas.

If these conditions are not satisfied, you must alter your DMCL and segment definitions before declaring the area to be shared. Failure to do so will mean that one or more members of the group will be unable to access the area.

These requirements are waived on any CA IDMS system that is accessing the area in a transient retrieval mode regardless of whether or not the area has been designated for data sharing.

**Notify Locking Considerations**

Notify locks are supported in a data sharing environment. If a transaction executing in one member places a notify lock on a record, it will be informed of any changes made to that record by other transactions regardless of where (in which member) the updating transaction executes. However, cross-member notification of retrieval is not supported. If an application relies on notification of retrieval, the database that it accesses should not be shared for update.

**Enabling Data Sharing**

In addition to defining a data sharing group, the following tasks must be performed to enable data sharing:

- The DMCL of every member of the group must be altered to indicate that data sharing is allowed
- Each area that is to be eligible for data sharing must be so designated. This can be done in the DMCL or through DCMT commands.
- Each file associated with a shared area must be associated with a coupling facility cache structure. This can either be done in the DMCL or through DCMT commands.
Altering the DMCL Definition

To enable data sharing, the DMCL definition for every system in the data sharing group must be altered to indicate that data sharing is allowed and to specify certain related information.

The DMCL definition may also be changed to indicate which areas are to be shared and the shared cache structure to be associated with their files. This information should be specified in the DMCL rather than through DCMT commands for areas that are always or typically shared among members of the group. For more information on DMCL and data sharing, see the Database Administering section.

Data sharing attributes that are defined in the DMCL include the following:

- Whether data sharing is to be used
- The number of lock table entries
- The maximum number of group members
- A default shared cache
- What to do if problems with the coupling facility are encountered

The impact of these attributes is discussed below.

**Specifying data sharing attributes**

Each data sharing group has an associated coupling facility lock structure. The first CA IDMS system to become a member of the group, establishes the attributes of the lock structure. These attributes remain in effect until all members of the group have terminated normally. As long as any CA IDMS system is either active or has failed and not yet been restarted, the existing lock structure attributes remain in effect.

**Specifying the number of lock table entries**

The number of lock table entries determines the number of hash entries within the lock structure that will be used for managing locks. The higher the number of lock entries, the less chance that multiple resources will hash to the same lock table entry, a situation that results in increased overhead. However, the higher the number of lock table entries, the larger the size of the lock structure.

As a guideline, specify as the number of lock entries, the highest SYSLOCKS value of any CA IDMS system that will be a member of the data sharing group. The number of lock entries will be rounded up to a power of 2.

**Specifying the maximum number of group members**
The maximum number of group members determines the number of CA IDMS systems that can be members of a data sharing group at any one time. If CA IDMS system terminates normally, it does not count as a group member for the purposes of this limit; however, if a CA IDMS system terminates abnormally, it is still a member of the group until it is restarted and shut down normally. The higher the maximum member count, the more space is required in the lock structure. For more information on sizing the lock structure, see Configuring the Coupling Facility.

The value specified may be overridden by CFRM policy. For more information, see the appropriate IBM documentation. To determine the actual value in effect, use the DCMT DISPLAY DATA SHARING command.

**Specifying a shared cache**

A shared cache is a structure defined within a coupling facility that allows data stored in the cache to be shared by multiple CA IDMS systems. Assigning a file to a shared cache allows CA IDMS systems to use the cache as a shared buffer.

Files may be assigned to a shared cache whether or not their associated areas are designated for data sharing. However, if an area is designated for data sharing, all of its associated files must be assigned to a shared cache.

**Specifying the connectivity loss option**

The ON CONNECTIVITY LOSS parameter enables a site to specify what action a data sharing member should take in the event that connectivity to the coupling facility is lost or a failure in the coupling facility is detected. Specifying ABEND directs the system to abend immediately; specifying NOABEND directs the system to remain active as long as possible. By specifying NOABEND, it is possible for a member to remain active servicing requests for non-shared areas. It will not be possible to shut down the system normally.

**Using DCMT Commands**

DCMT commands provide the ability to do the following:

- Change the default shared cache or connectivity loss setting for a CA IDMS system. For more information, see the DCMT VARY DATA SHARING command in the *CA IDMS System Tasks and Operator Reference section*.

- Enable or disable data sharing for an area or for all areas in a segment. For more information, see the DCMT VARY AREA and DCMT VARY SEGMENT commands in the *CA IDMS System Tasks and Operator Reference section*.

- Change the shared cache for a file, for all files associated with an area, and for all files in a segment. For more information, see the DCMT VARY FILE, DCMT VARY AREA, and DCMT VARY SEGMENT commands in the *CA IDMS System Tasks and Operator Reference section*.

Additional DCMT commands display information on data sharing. For more information on these commands, see Monitoring Data Sharing Groups.
Member Failure

When a member of a data sharing group fails, recovery is typically effected by restarting the system and allowing warmstart to recover in the normal way. The primary data sharing consideration is that the system must be restarted using the same group and member names that were in effect at the time of failure.

Additionally, it is important to restart the system as soon as possible, since other members are prohibited from accessing resources that need to be recovered by the failing member. For more information on how the system deals with attempts to access unrecovered data, see Accessing Unrecovered Data (see page 395).

If warmstart fails, manual recovery must be used to restore the database to a valid state.

⚠️ Note: For more information on the impact of data sharing on manual recovery, see the CA IDMS Database Administering section.

Coupling Facility Failures

The following types of failures relating to the coupling facility can occur:

- Loss of connectivity to a coupling facility structure
- Failure of a coupling facility structure
- An XES processing error

**Loss of connectivity**

Loss of connectivity to a coupling facility structure occurs when a connector (in this case a CA IDMS system) can no longer communicate with the coupling facility in which the structure resides. Loss of connectivity may be the result of operator commands or hardware failures. If connectivity is lost to either the list or lock structure associated with a data sharing group, the CA IDMS system will either abnormally terminate or remain active so that requests that do not require access to the affected structure can be serviced. The action taken will depend on the connectivity loss option that is in effect for the system. If the system remains active, tasks that require access to the list or lock structure will be abnormally terminated.

⚠️ Note: For more information on establishing a connectivity loss setting, see Altering the DMCL Definition.

**Structure failure**
Structure failure indicates that a structure residing in the coupling facility has been damaged. Damage may be due to hardware failures or XES processing errors.

CA IDMS treats a failure of the lock or list structure associated with a data sharing group as a loss of connectivity to that structure. Depending on the connectivity loss setting in effect, the system will either abnormally terminate or remain active so that requests that do not require access to the affected structure can be serviced. If the system remains active, tasks that require access to the damaged structure will be abnormally terminated.

XES processing error

An XES processing error will result in the abnormal termination of any CA IDMS system that detects this condition. Contact IBM for the appropriate response. If there is any doubt as to the integrity of the lock and list structures associated with the data sharing group, follow the steps outlined below, deleting both the list and lock structures if necessary.

Responding to coupling facility failures

An error accessing the list or lock structure associated with a data sharing group will result in the abnormal termination of every member that detects the problem. Termination may occur immediately, if the connectivity loss option indicates ABEND, or it may be deferred until an attempt is made to shut down the system. In either case, the system cannot be shutdown normally, because it cannot successfully disconnect from at least one of either the list or lock structures.

If the failure is due to a loss in connectivity, simply restart the failing CA IDMS systems after taking appropriate action to restore connectivity. If the loss in connectivity is due to operator commands, issue the necessary commands to restore connectivity before restarting the system. If the problem is due to a hardware failure and another z/OS image has connectivity to the coupling facility, the failed CA IDMS systems can be restarted there. If no such z/OS image exists, take one of the following actions:

- Correct the hardware problem and restart the systems.
- If an alternate coupling facility can be accessed, treat the loss in connectivity as a structure failure and respond as outlined below. Recreate the list and lock structures on the alternate coupling facility before restarting the systems.
- Undertake manual recovery to roll out only the transactions active at the time of failure. Initialize the journal files and restart one of the CA IDMS systems as a stand-alone system (that is, without data sharing enabled).

To recover from a failure of the list or lock structure, the following steps should be taken:

- Terminate all remaining systems in the data sharing group. If they won’t shut down, cancel them.
- Delete all connections to the failed structure and delete the failed structure using SETXCF FORCE commands.
- Restart all members that did not terminate normally. This will result in a group restart situation.

Note: For more information on restarting all members of a group, see Group Restart.
Group Restart

Group restart occurs when an inconsistency is detected during startup of a group member. CA IDMS uses an XCF group and coupling facility lock and list structures in its data sharing support. Each of these maintains a status for each member of the group. Group restart is necessary if there are inconsistencies in these statuses or if either the lock or list structure doesn’t exist when it should.

The purpose of group restart is to rebuild the lock and list structures from information in the journal files of previously failed members. The need for group restart is detected by a group member during startup. The first group member detecting the need for group restart becomes the restart coordinator. It is the coordinator’s responsibility to monitor the progress of the restart process and direct the actions of other members.

In order to complete group restart, all failed members must be restarted. Members that had previously shut down normally may also be started during group restart, but this is not necessary. The restart coordinator will display messages on the JES log showing the progress of the restart and indicating which members still need to be restarted. Once all failed members known to the coordinator have been restarted, a message will be sent to the operator to confirm that all failed members have been restarted. It is the operator’s responsibility to ensure that this is true before responding positively. Failure to include an abended member in the restart process can lead to corrupted data, since uncommitted updates by such a member have neither been rolled out nor the affected records locked to prevent access by other members.

During group restart, each previously failed member will update the list structure using information contained in its journal files. This information reflects the status of shared areas being processed by this member at the time of failure. The member then proceeds with its normal warmstart process to rollout incomplete transactions. Any transaction that cannot be rolled out will be restarted and appropriate locks acquired to protect the unrecovered data. Once this process is complete, startup is paused until all failed members have been restarted and reached this point in their processing. When this state is achieved, the restart coordinator, after confirmation from the operator, informs the other members that group restart is complete. Members then complete their startup process.

If group restart is interrupted because of a system failure, it can be restarted simply by restarting the failed systems. The procedures to follow after other types of failures will depend on the nature of the error.

Accessing Unrecovered Data

When a group member fails after it has made changes to a shared area and before those changes have been committed or rolled out, locks prevent access to the unrecovered data until the failing member is restarted. These locks might prevent access to an individual record, an entire page of records or an entire area, depending on the resource type and mode of the lock held by the failing member. It is therefore important to restart the failed member as soon as possible.

If a transaction on another system attempts to access unrecovered data, it can either wait for the failed member to recover or it can abort. The choice of actions is determined by the RECOVERY WAIT setting. This value is initially established in the system definition (SYSTEM statement), can be displayed with DCMT DISPLAY TIME, and can be dynamically changed using DCMT VARY TIME. Exit 37 can override the recovery wait action for each task attempting to wait on an unrecovered resource.
Sharing Queues and Enqueued Resources

Data sharing groups provide the ability to share DC queues and enqueued resources between members of the group. This enables online tasks executing in different group members to communicate with one another just as if they were executing within the same CA IDMS system. Each of these facilities is described below.

Sharing Queues

In order to share queues between group members, the system queue area (DDLDCRUN) must be shared for update between members. Just as for any other area, the queue area is shared for update by designating it as such in the DMCL or through a DCMT command. A queue area may be shared by all or a subset of the group's members. There can be only one shared queue area per data sharing group.

⚠️ Note: For more information on enabling data sharing for an area, see Sharing Update Access to Data.

If a CA IDMS system is using a shared queue area, then all queues in that queue area are assumed to be shared except those specified as local in an exception table. A shared queue can be accessed by tasks executing on any group member sharing the queue area. There is only one shared queue with a given name in a queue area. Conversely, local queues are queues that can be accessed by only a single group member. There can be as many local queues with a given name as there are members sharing the queue area. Local queues are automatically qualified by their associated member name.

No changes are needed in application programs accessing queues in a shared queue area, regardless of whether they are local or global. The same queue commands (GET QUEUE, PUT QUEUE, etc.) are used.

Queue-initiated tasks are supported for shared queues, just as for non-shared queues. The member that causes the queue threshold to be exceeded is the one on which the queue-initiated task is executed.

Designating queues as local
If a queue area is shared, then by default all queues in that queue area are shared. To specify a queue as local, add an entry to the LCLQUEUE module using the #LCLRES macro and reassemble LCLQUEUE. LCLQUEUE is distributed with one entry for the RHDCSEITTIMETASKS queue. This entry should always be included in any reassembly of the module.

The queue name of the local queue is specified in the RESNAME parameter of the #LCLRES macro. Queues can be referenced generically by specifying an asterisk (*) as the last character of the RESNAME value. A queue name specified in this way indicates that all queues whose name matches that of the specified name (excluding the asterisk) are local.

The following example designates MYQUEUE as a local queue by adding an entry to the LCLQUEUE module:

```
#LCLRES TYPE=INITIAL,RESTYPE=QUEUE
#LCLRES RESNAME='RHDCSEITTIMETASKS'
#LCLRES RESNAME='MYQUEUE'
#LCLRES TYPE=FINAL
END
```

**Switching queue scope**

If there is a need to switch a queue from being shared to local or vice versa, take the following steps:

- Process and delete all existing queue entries
- Use the QUED task to delete the queue
- Create a new LCLQUEUE module adding or removing entries as necessary
- Issue DCMT VARY NUCLEUS on every group member that shares the queue area to bring in a new copy of the LCLQUEUE module

**Impact of shared queues**

Shared queues not only allow applications executing on different group members to communicate with one another, but they also impact the runtime system in the following ways:

- Report queues are shared globally by all members sharing the queue area. This means that reports can be printed on any group member that has a printer defined for the report’s class.
- Checkouts of maps and dialogs are global across all members that share the queue area. This means that if a dialog is checked out on one member of the group, it is protected from check out on another member of the same group. The integrity of the checkout is guaranteed only if dictionaries are referred to by the same DBNAME in all sharing group members.
- Messages that are to be sent to users whenever they sign on (using the SEND ALWAYS command) will be sent regardless of which member the user signs on to, provided those members are sharing the queue area with the member on which the DCUF command was issued.

**Sharing Enqueued Resources**

Within a data sharing group, all enqueued resources are global by default. This means that if a task enqueues a resource exclusively on one member, that resource is unavailable on all other members of the group.
Resources can be designated as local. Enqueues on local resources impact only the system on which the enqueue is issued.

No changes in application programs are necessary in order to enqueue global or local resources.

**Designating resources as local**

To designate a resource as local, add an entry to the LCLENQDQ module using the #LCLRES macro. LCLENQDQ is distributed with three entries needed for internal CA IDMS processing. These entries should always be included in any reassembly of the module.

The resource id of the local resource is specified in the RESNAME parameter of the #LCLRES macro. Generic resource names can be specified by coding an asterisk (*) as the last character of the RESNAME value. A generic resource name indicates that all resources whose id matches that of the specified name (excluding the asterisk) are local.

The following example designates any resource beginning with "MYRES" as a local resource by adding an entry to the LCLENQDQ module:

```
#LCLRES TYPE=INITIAL,RESTYPE=ENQDEQ
*  
* The following resource-ids are used internally by the 
* system, and need a local scope in a data sharing * group environment.
* 
#LCLRES RESNAME='RHDCD09I'
#LCLRES RESNAME='RHDCD09P'
#LCLRES RESNAME='USERJRNL'
*  
* Define other local resources here
*  
#LCLRES RESNAME='MYRES*'
*  
#LCLRES TYPE=FINAL
END
```

**Switching resource scope**

If there is a need to switch an enqueued resource from being global to local or vice versa, take the following steps:

- Create a new LCLENQDQ module adding or removing entries as necessary
- Issue DCMT VARY NUCLEUS on every group member to bring in a new copy of the LCLENQDQ module

**Defining local print tasks**

A local resource should be defined for every print task defined using the #UCFUFT macro. To do this, add an entry to the LCLENQDQ module specifying the PTID value in the #UCFUFT macro as the RESNAME value in the #LCLRES macro.

**Monitoring Data Sharing Groups**

The following facilities can be used to monitor a data sharing group:
Monitoring Through DCMT Commands

The following commands enable monitoring of various aspects of a data sharing group:

- **DCMT DISPLAY AREA**
- **DCMT DISPLAY DATA SHARING**
- **DCMT DISPLAY LOCK STATISTICS**

⚠️ **Note:** For more information on the output of these commands, see the *System Tasks and Operator Reference section.*

**DCMT DISPLAY AREA**

This command shows the sharability state of an area and whether or not there is inter-CV-interest in the area.

Accessing a shared area for which there is inter-CV-interest will result in higher overhead because global locking must be used to control access to the area by members of the data sharing group.

Accessing a shared area for which there is no inter-CV-interest will incur only slightly more overhead than would be needed if the area were not shared.

**DCMT DISPLAY DATA SHARING**

This command displays the following types of information:

- A list of group members and their status
- Statistics associated with accessing the data sharing group’s list and lock structures
- Information regarding the available space in the lock structure
- Statistics on XCF messages used for inter-member communication

**Member status**

Each member of a data sharing group has a member state that is assigned by XCF and a user state that is assigned by CA IDMS.

The following XCF member states can be associated with CA IDMS systems:

- Active -- Indicates that the member is currently executing
The following user states can be associated with CA IDMS systems:

- **Failed** -- Indicates that the member has terminated abnormally
- **Initial** -- Indicates that startup is in progress for the member
- **Recovering** -- Indicates that the member is in the process of recovering from a prior abnormal termination
- **Ready** -- Indicates that the member has completed recovery and is ready to open the database system
- **Active** -- Indicates that startup is complete
- **Quiescing** -- Indicates that the system is in the process of closing the database system
- **Quiesced** -- Indicates that the database system has been closed

**Monitoring available lock structure space**

Record data entries are stored in the coupling facility lock structure to record information on exclusive global transaction locks. The more concurrently held exclusive locks, the more space is needed in the lock structure.

The output of the DCMT DISPLAY DATA SHARING command allows you to monitor the available space in the lock structure by displaying the maximum number of record data entries that can be stored in the lock structure, the current number that are in use, the highest number that were ever used and the number of times the lock structure encountered a short-on-storage condition.

As the lock structure becomes full, CA IDMS will release exclusive locks on proxies, if possible, to relieve the short-on-storage condition; however this is undesirable because it means it increases the overhead associated with global locking. Furthermore, even releasing all unused proxy locks may not free up enough space in the lock structure. If there is not enough space to store a record data entry when one is required, the task requesting the lock will fail.

The amount of space in the lock structure can be increased while group members remain active by using the SETXCF START ALTER command, provided the coupling facility in which the lock structure resides contains sufficient free space. If it doesn’t, the lock structure must be reallocated. To do this, all members in the data sharing group must be shutdown and the CFRM policy changed.

**DCMT DISPLAY LOCK STATISTICS**

The output from this command displays information on local and global locks acquired to control access to transaction resources.

The following are DCMT DISPLAY LOCK STATISTICS command considerations:

- The ratio of global resource lock requests to local lock requests is a measure of contention for resources between members. If there is no contention, then this ratio will be small since the only global resource locks acquired will be for areas and in an active system in which areas are always readied in a shared mode, global area locks will generally be retained once they are acquired.
The ratio of the number of waits to the number of global requests is also a measure of contention. This contention may be due to resource conflicts or other factors, such as channel contention, or false contention caused by synonyms when hashing to the lock table. If this ratio is high, use operating system tools to determine the nature of the contention. False contention can be reduced by increasing the number of entries in the lock table. For more information on specifying the number of lock table entries, see Enabling Data Sharing.

Monitoring Through Performance Monitor

Performance Monitor collects and reports on information associated with a data sharing group. This capability is available through the Interval Monitor. There are online display screens and Performance Monitor reports available. For more information, see the Performance Monitor System Administering section and the CA IDMS Performance Monitor Using section.

Monitoring Through Journal Reports

Journal reports show the nodename of the system that created the journal image. In a data sharing environment, the nodename is the same as the system’s membername.

Journal report 8 shows the time when the journal image was created. This time is based on GMT (Greenwich Mean Time). This time, in conjunction with a journal sequence number, is used to sort and merge journal images, both for images created by a single system and across members in a data sharing environment. For more information on journal reports, see the Reporting section.

CV Cloning

CV cloning allows you to start multiple CVs that are copies (clones) of an existing system definition for a CA IDMS system. CV cloning is designed to be used in a Sysplex environment, especially for cloning back-end CVs for a DBGroup or members of a data sharing group.

- For a clone that is not part of a data sharing group, all database areas are forced to a status of RETRIEVAL, with the exception of the system log and queue areas.
- For a clone that is a member of a data sharing group, see Using CV Clones With Data Sharing.

- Planning CV Cloning (see page 402)
- Implementing CV Cloning (see page 402)
  - System Definition Requirements (see page 402)
  - Special File Requirements for CV Clones (see page 403)
  - System Startup JCL Requirements (see page 403)
  - Example (see page 404)
  - Using CV Clones with Dynamic Database Session Routing (see page 404)
  - Using CV Clones With Data Sharing (see page 404)
Planning CV Cloning

Planning the number of clones

You must decide in advance which CVs you want to clone and how many clones of each you want to allow. You specify this information in the system startup JCL.

You are limited to a maximum of 255 CVs within a Sysplex, regardless of the number of SVCs you have installed or how many z/OS images you are running.

Defining cloned CVs

The system definition of the CVs you wish to clone must conform to specific naming conventions and include special definitions, and the system start up JCL must include parameters for cloning.

Implementing CV Cloning

To use CV cloning, you have to perform the tasks discussed in the following sections.

System Definition Requirements

System naming conventions

The system definition for a CV you wish to clone must conform to specific naming conventions. By adhering to these naming conventions, CA IDMS can implement clones without generating multiple physical copies of a system definition.

System definitions for systems that you will clone, must include the following:

- The DC system number must match the CV number and must be in the range from 0 through 255. At CV startup, this number is incremented as necessary to locate an available number to assign to each cloned CV.
  The DC system number is the value specified on the first parameter of the system definition SYSTEM statement, SYSTEM dc-ucf-version-number. CV number is the value specified on the CVNUMBER parameter of the SYSTEM statement.

- All VTAM ACB names must follow the convention, xxxxxxxn, where xxx can be any five characters you assign to make the name unique within your environment. Nnn is the sysgenned CV number and is overlaid at runtime with the number determined during start up of the cloned CV; for example, to allow multiple lines within the same CA IDMS system.

- CA IDMS system node names must follow the pattern, yyyyynnn, where the first five characters can be any characters you need to make the name unique within your environment and the nnn is the number specified for the CV at system generation. This number is overlaid at runtime with the CV number determined to be available at system startup.
Note: If you are modifying existing CA IDMS system definitions, be sure to change this node name on all statements in which it is referenced. For example, the RESOURCE NAME and NODE system definition statements.

Define simulator line

You must also define a simulator line (S3270Q), which allows the clones to be configured without operator intervention. To do this, create commands to activate lines as described next.

Create commands to activate lines

Create an input data set for the simulator line that contains the appropriate DCMT commands to activate the communication lines. The following example shows sample commands you might include in this data set:

dcmt vary line ddsvtam on.
dcmt vary line ccil ine on.

Special File Requirements for CV Clones

Each cloned CV requires its own copy of all journal files. Depending on system configuration, each CV may require its own copy of the following files:

- DDLDCLOG -- If the SYSTEM statement in the base system SYSGEN specifies LOG DATABASE, each system requires its own log file.
- DDLDCRUN -- Unless the systems are sharing the queue area, each system requires its own queue file. For more information on sharing the queue area, see Sharing Queues.
- DDLDCSCR -- If the SYSTEM statement in the base system SYSGEN specifies SCRatch in XA STOrage is NO, each system requires its own scratch file.
- SYSC TL -- The default system action is that no SYSC TL file will be opened when a clone is started. All ERUS tasks will then be routed through the first CV that is started. If optional bit 199 is set in RHDCOPTF, each clone attempts to open and update the SYSC TL file. In this case, each system must have its own file. For more information on RHDCOPTF, see Applying Optional Functionality.

System Startup JCL Requirements

The startup JCL for each CV you want to clone must specify that cloning is to be in effect and the maximum number of clones that are to be used. This information is provided as a startup parameter using positional or keyword parameters. For more information on supplying runtime options, see Specifying Runtime Options.

If positional parameters are used, you must specify a C in column 25, followed immediately by a one-to-three digit number indicating the maximum number of clones.

If keyword parameters are used, you must specify CLONING=Y and CLONES=nnn, where nnn specifies the maximum number of clones.
Example

Assume a CV is to run a clone of IDMS070. Specifying CLONING=Y,CLONES=9 on the EXEC PARM indicates cloning is allowed and that a maximum of 9 clones can be created. The search for an available system number to assign to a clone begins with the system number (070 specified in RHDCPARM or the value in the S=nnn of the PARM parameter). The first available system ID for a clone of IDMS070 is IDMS071. If IDMS071 is running, it tries system ID IDMS072 and so forth, until it finds an available system ID or reaches IDMS079.

Using CV Clones with Dynamic Database Session Routing

CV cloning can be used with dynamic database session routing to allow multiple copies of a back-end CV to be available to route requests to.

The following diagram shows the use of cloning with the back-end CVs assigned to the EMPGROUP, which was defined in Using Dynamic Database Session Routing.

Using CV Clones With Data Sharing

If a clone is a member of a data sharing group, the following occurs:

- The area status of areas that have attribute DATA SHARING YES in the DMCL is kept, that is, not forced to retrieval.
- The node name is set to be the system's membername.
System-Managed Rebuild

CA IDMS supports system-managed rebuild of any of the coupling facility structures it exploits. System-managed rebuild is intended for use in planned reconfiguration scenarios.

To enable system-managed rebuild, the following conditions must exist:

- You must have at least two coupling facilities of CFLEVEL=8 or higher in the CFRM preference list for the structure.
- You must have an active CFRM couple data set that is formatted with the ITEM NAME(SMREBLD) NUMBER(1) statement.

The system-managed rebuild process for a structure with the name `structure-name` can be started with the following operator command:

`SETXCF START,REBUILD,STRNAME=structure-name`

Note: For more information, see the following IBM documentation:

- Setting Up a Sysplex
- System Commands

System-Managed Duplexing Rebuild

CA IDMS supports system-managed duplexing rebuild of any of the Coupling Facility (CF) structures it exploits. System-managed duplexing rebuild provides a failure recovery capability. It allows you to maintain the data in duplexed structures on an ongoing basis. If a failure occurs, processing continues with the duplexed structure, thereby eliminating a "single point of failure".

To enable system-managed duplexing rebuild, the following conditions must exist:

- You must have at least two coupling facilities of CFLEVEL=10 or higher in the CFRM preference list for the structure.
- You must have an active CFRM couple data set that is formatted to support system-managed duplexing rebuild.
- You must define the structures with DUPLEX(ENABLED) or DUPLEX(ALLOWED) to the CFRM policy.

Note: For more information, see the following IBM documentation:
Two-Phase Commit Considerations

This section discusses the two-phase commit support provided in CA IDMS, specifically resynchronization. For a description of two-phase commit support, the resynchronization process, and the impact of two-phase commit on recovery, see the Database Administering section.

- General Two-Phase Commit Considerations (see page 406)
- Two-phase Commit Support - RRS (see page 408)
- Two-Phase Commit Support with CICS (see page 420)

General Two-Phase Commit Considerations

Contents

- Restarting a Failed System (see page 406)
- Resource Name Table Requirements (see page 406)
- System Name During Warmstart (see page 407)
- Incomplete Distributed Transactions at Shutdown (see page 407)

This section discusses the general considerations in using Two-Phase Commit support.

Restarting a Failed System

When restarting a failed central version, it is advisable to restart it on the same logical operating system image as the one on which it abnormally terminated. For more information on the resynchronization process, see the Database Administering section.

Resource Name Table Requirements

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

1. Define the partner system as a NODE in the coordinator's system definition. This option is appropriate if there is a direct communications path between the two systems and dynamic routing through DBGROUPs is not used or if forcing a specific access method with dynamic routing.
2. Define the partner system's node name as a destination in the coordinator's resource name table and identify the DBGROUP to which new connections should be routed by specifying a VIA parameter. This option is appropriate if dynamic routing is in use and the default access method is acceptable. Wildcarding the destination name can eliminate the need for defining a resource for every DBGROUP member and thus allows additional members to be added to the group without defining new resources.

3. Define the partner system's node name as a destination in the coordinator's resource name table and identify the intermediate node through which communication should be routed by specifying a VIA parameter. This option should be used only if there is no direct communications path between the two systems.

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

### System Name During Warmstart

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

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

- If a DCNAME parameter is specified in the SYSIDMS file, it must match the value stored on the journal files, otherwise warmstart fails

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

### Incomplete Distributed Transactions at Shutdown

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

IDMS DC200241 V74 T1 Active transactions exist. Abending.

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

More Information

- For more information on resynchronization, see the CA IDMS Database Administering section.
- For more information on the DCMT DISPLAY DISTRIBUTED TRANSACTIONS and the DCMT VARY DISTRIBUTED RESOURCE MANAGER RESYNC commands, see the CA IDMS System Tasks Operator Reference section.

Two-phase Commit Support - RRS

RRS is IBM's resource recovery platform for z/OS. CA IDMS can exploit RRS services in the following ways:

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

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

- Enabling RRS Support Within a CA IDMS System (see page 409)
- Impact on System Startup (see page 409)
- RRS Support for Batch Applications (see page 410)
  - Enabling RRS for Batch Applications (see page 412)
  - Batch RRS Transaction Boundaries and Application Design Considerations (see page 412)
  - Batch RRS Examples (see page 414)
- RRS Support for Online Applications (see page 415)
  - RRS Support Examples (see page 417)
  - RRS Support Parameters (see page 417)
- Optimizations Supported (see page 418)
- Resynchronization Between RRS and CA IDMS (see page 418)
  - When Does It Occur? (see page 418)
  - What Does It Entail? (see page 418)
  - Responding to Resynchronization Failures (see page 419)
Enabling RRS Support Within a CA IDMS System

To exploit RRS functionality through batch or online applications, you must enable RRS support in one or more central versions. To do this, you code specific values in the PARM parameter of the EXEC statement for the startup routine.

You use the PARM field to do the following:

- Enable RRS support
- Optionally specify one plus the number of subtasks that are capable of accessing RRS. The value specified must be between 2 and 99. If no value is specified, the number of subtasks is determined as one plus the number of processors. If multitasking is also enabled, the value specified also represents the number of subtasks that perform CA IDMS work.

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

```
//STARTUP EXEC PGM=dcucfsys,PARM='MT=N,RRS=Y,SUBTASKS=3'
```

For more information on the PARM parameter, see Specifying Runtime Options.

You can use the DCMT DISPLAY SUBTASK command to see if RRS support is enabled on the system. The DCMT VARY SUBTASK command can be used to dynamically enable or disable RRS support. For more information on these commands, see the System Tasks and Operator Reference section.

Impact on System Startup

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

```
IDMS.RM.node-name.CA
```

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

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

```
DC224001 V73 T23 Registered with RRS services as IDMS.RM.node-name.CA
```

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

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

After successful registration with RRS, a resynchronization process is started in order to exchange information and complete recovery following a failure. For more information, see Resynchronization Between RRS and CA IDMS.
The operating system image on which a failed system is restarted can be significant. For more information, see the IBM manual *MVS Programming: Resource Recovery* and the specific topic "Resource Manager Environments."

**RRS Support for Batch Applications**

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

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

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

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

ENABLE_RRS=ON

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

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

ENABLE_RRS=OFF

Notes:

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

Batch RRS Transaction Boundaries and Application Design Considerations

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

- The application verbs that mark a transaction boundary are the RRS verbs: Commit UR or Backout UR.
- Prior to issuing a Commit UR, all database sessions whose transaction is under the control of RRS must be committed and optionally completed. This can be accomplished by:
  - Issuing a COMMIT TASK ALL or FINISH TASK DML command.
  - Explicitly committing all active database sessions by issuing a COMMIT or FINISH DML command for each session.

The following additional considerations apply:

- A COMMIT TASK or FINISH TASK must be issued if a BIND TASK was issued.
- All remote subordinate database sessions initiated by SQL routines or database procedures executing on behalf of a database session started by the application must be completed before the RRS commit is issued. This can be done by terminating the application's database session using a FINISH TASK or COMMIT WORK RELEASE command or by issuing a COMMIT TASK ALL or COMMIT WORK command.
Committing or finishing a database session does not impact its associated transaction when it is under the control of RRS. Only the database session is affected. For example, when a run unit is finished, the database session is closed and currency locks are maintained until the RRS UR is committed or backed out.

It is possible to serially create and finish database sessions within a single RRS UR; however, unless transaction sharing is in effect, a deadlock may occur if a later session attempts to access a record that was updated by a previous session.

- When a ROLLBACK WORK or ROLLBACK TASK CONTINUE DML command is issued before a Backout_UR request, it results in the back out of the entire RRS UR, even if the application subsequently issues a Commit_UR request. At the time the ROLLBACK command is issued, the changes made to the CA IDMS database are backed out and associated locks are released. However, the RRS UR is not backed out until an RRS commit or backout operation is initiated. If necessary, CA IDMS will vote "BACKOUT" during the first phase of commit processing to cause the RRS UR to be backed out.

- If a Backout_UR is issued prior to a ROLLBACK DML request, all active database sessions must be rolled back (using an appropriate ROLLBACK statement) before any further CA IDMS work can be done.

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

Example of a COBOL Batch Program

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

```cobol
*RETRIEVAL
*NO-ACTIVITY-LOG
*DMLIST
IDENTIFICATION DIVISION.
PROGRAM-ID. MBINDSUB.
*******************************************************************
* SUBSCHEMA CONTROL IS PASSED FROM MAINLINE PROGRAM.
*******************************************************************
ENVIRONMENT DIVISION.
IDMS-CONTROL SECTION.
PROTOCOL. MODE IS BATCH DEBUG
IDMS-RECORDS MANUAL.
DATA DIVISION.
SCHEMA SECTION.
DB EMPSS01 WITHIN EMPSCHM VERSION 100.
WORKING-STORAGE SECTION.
01 WK-DATA.
   02 I PIC S9(4) COMP.
01 COPY IDMS SUBSCHEMA-RECORDS.
01 COPY IDMS SUBSCHEMA-RECORDS.
LINKAGE SECTION.
01 DB-PARM.
   02 DBNAME-IN PIC X(8).
   02 FILLER PIC X.
   02 DBNODE-IN PIC X(8).
   02 FILLER PIC X.
   02 ACTION-CD PIC X.
     88 ACT-BIND VALUE 'R'.
     88 ACT-BINDU VALUE 'U'.
     88 ACT-DML1 VALUE '1'.
     88 ACT-DML2 VALUE '2'.
     88 ACT-DML3 VALUE '3'.
```
Batch RRS Examples

Assume a CV is to run a clone of IDMS070. Specifying CLONING=Y,CLONES=9 on the EXEC PARM indicates cloning is allowed and that a maximum of 9 clones can be created. The search for an available system number to assign to a clone begins with the system number (070 specified in RHDCPARM or the value in the S=nnn of the PARM parameter). The first available system ID for a clone of IDMS070 is IDMS071. If IDMS071 is running, it tries system ID IDMS072 and so forth, until it finds an available system ID or reaches IDMS079.
RRS Support for Online Applications

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

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

To initiate a commit operation involving all interested resource managers, the online application issues a CA IDMS commit DML command (such as a FINISH TASK or a COMMIT WORK). The local transaction manager then uses RRS as an agent to coordinate its updates with those of the external resource managers. The following diagram illustrates the flow of control that occurs:
Programming Interface

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

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

Return codes

00 -- An RRS context exists; the field pointed to by RRSCTXA contains the current RRS context.
- 04 -- No RRS context exists; the field pointed to by RRSCTXA is cleared.

- Any other return code -- An internal error occurred. The content of the field pointed to by RRSCTXA is undefined.

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

Return codes

00 -- The RRS context token was successfully saved by the CA IDMS transaction manager.

Any other return code -- An error occurred. Return codes 103-107, 301, 701, 756, F00, and FFF are from context services. Their description can be found in the IBM section MVS Programming: Resource Recovery in the specific topic "Begin_Context."

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

Return codes:

00 -- The RRS context was successfully terminated. The field pointed to by RRSCTXA is set to binary zeros.

Any other return code -- An error occurred. Return codes 103-107, 360-369, 703, 756, and FFF are from context services. Their description can be found in the IBM section MVS Programming: Resource Recovery in the specific topic "End_Context."

Application Design Considerations

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

RRS Support Examples

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

RRS Support Parameters

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

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

- X'01': Get RRS context.

Optimizations Supported

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

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

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

Resynchronization Between RRS and CA IDMS

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

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

When Does It Occur?

Resynchronization between RRS and a CA IDMS system occurs as follows:

- When a central version is started, as part of registering with RRS.
- When resynchronization is manually driven through a DCMT VARY DISTRIBUTED RESOURCE MANAGER command. For more information, see the *System Tasks and Operator Reference section*.

What Does It Entail?

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

The CA IDMS log name has the following format:
IDMS.RM.jrnlstamp.node-name.CA

- **jrnlstamp**
  Specifies the central version's 26-character journal timestamp with dashes ("-") replaced by underscores ("_"). This value is assigned by a central version the first time it opens a set of journal files after they have been formatted.

- **node-name**
  Specifies the central version's node name, padded with underscores ("_") if it is less than eight characters in length.

The following messages are displayed during the resynchronization process:

DC224002 V73 T23 RRS log name ATR.B989786A2A8AA40.IBM
DC224002 V73 T23 Resource Manager log name IDMS.LOG.yyyy_mm_dd_hh.mm.ss.ssssss.node-name.CA
DC224006 V73 T23 Resynchronization with RRS complete

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

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

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

Any of these conditions result in a resynchronization failure.

**Responding to Resynchronization Failures**

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

- If the RRS LOG was formatted, complete the transactions once the central version is up and running. For more information on how to do this, see the Database Administering section.
- If the CA IDMS journal files were formatted, use the RRS ISPF panels to complete the transactions. For more information on RRS panels, see the IBM manual *MVS Programming: Resource Recovery.*
Two-Phase Commit Support with CICS

Contents
- Implementation Requirements (see page 420)
- Programming Interface (see page 421)
- Optimizations Supported (see page 421)
- Requesting the Use of Two-Phase Commit (see page 422)
- CICS System Name Requirements (see page 423)
- Resynchronization between CICS and CA IDMS (see page 424)
  - The Resynchronization Transaction and Program (see page 424)
  - How is Resynchronization Initiated? (see page 425)
  - When Should You Manually Resynchronize? (see page 425)
  - The Resynchronization Process (see page 425)
  - OPTIXIT Considerations (see page 426)
- CICS Resynchronization Task Execution (see page 426)
  - rsvn Syntax (see page 426)
  - rsvn Parameters (see page 426)
  - rsvn Examples (see page 426)
  - Creating the Resynchronization Program (see page 427)
  - Resynchronization Program Link Edit (z/OS) (see page 427)
  - Resynchronization Program Link Edit (z/VSE) (see page 427)
  - Defining a Resynchronization Transaction (see page 428)
  - Defining the Resynchronization Program (see page 428)

A two-phase commit protocol can optionally be used when accessing a CA IDMS database from a CICS application. Using a two-phase commit protocol ensures that updates made to CA IDMS data are coordinated with those made to other recoverable resources that the application accesses within the same CICS UOW (Unit Of Work). Two-phase commit support is provided only when using an r16 or higher CICS interface (IDMSINTC) to access an r16 or higher back-end CV.

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

Implementation Requirements

For successful two-phase commit operations between CICS and CA IDMS, the following steps must be taken:

- Review the IDMSCINT and CICSOPT parameters that apply to two-phase commit support and assemble either a new CICSOPT or IDMSCINT interface module as appropriate.
Ensure that each CICS system is uniquely identified through the TPNAME parameter of the CICSOPT macro or the CICS_NAME SYSIDMS parameter.

Ensure that the CICS system is logging transaction information. This requires the use of a CICS log file. For more information, refer to the appropriate CICS documentation.

Create a CICS RSYN transaction and program for each CICS interface module with which two-phase commit is used.

If using an OPTIXIT or OPTIQXIT to route requests to different back-end central versions, modify the OPTIXIT to recognize and correctly route resynchronization requests.

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

Programming Interface

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

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

- An explicit CICS BACKOUT command is issued.
- A CICS task terminates abnormally.
- A CA IDMS database session, for which the parameter AUTONLY is enabled, is rolled back. For more information on the AUTONLY parameter, see Requesting the Use of Two-Phase Commit.

Optimizations Supported

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

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

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

These optimizations not only reduce communications with participating resource managers, but also reduce log and journal overhead. For more information on the single-update and read-only optimizations, see the appropriate CICS documentation.
Requesting the Use of Two-Phase Commit

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

The following IDMSCINT and CICSOPT parameters control whether a database session is included in a CICS UOW and therefore if a two-phase commit protocol is used:

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

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

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

- **ONBACK**
  This option specifies the effect that a CICS backout operation has on a database session whose work is included in the CICS UOW. The session can optionally be treated as if a ROLLBACK or a ROLLBACK CONTINUE were issued, meaning that it can be terminated or remain active but have its currencies cleared.
All of these options can be specified through both IDMScINT and CICSOPT parameters. The CICSOPT parameters can override their IDMScINT counterparts or be used as defaults.

⚠️ **Note:** For more information on these parameters, see [CICSOPT syntax](#) and IDMScINT.

### Additional Two-Phase Commit Parameters

In addition to the parameters that control whether a two-phase commit protocol is used, the following additional CICSOPT parameters affect two-phase commit processing:

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

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

- **MAXIDMS**
  Specifies the maximum number of CA IDMS systems that an IDMScINTC interface can access during the life of a CICS system. This limit applies only to systems accessed through database sessions for which AUTOCMT is enabled.

- **RSYNTXN**
  Specifies the name of the resynchronization transaction defined to CICS for this interface. A separate CICS transaction must be defined for each interface in use within a CICS system.

⚠️ **Note:** For more information on the CICS resynchronization transaction, see the [Resynchronization between CICS and CA IDMS](#) (see page 424).

⚠️ **Note:** For more information on these parameters, see [CICSOPT syntax](#).

### CICS System Name Requirements

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

The name of the CICS system is established as follows:
The value in the CICS_NAME parameter specified in the SYSIDMS file included in the CICS startup JCL.

Or, if the CICS_NAME parameter is not specified:

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

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

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

The SYSIDMS parameter for specifying a CICS system name is as follows:

CICS_NAME=cics-name

- **cics-name**
  - Specifies a 1-4 character value that identifies the CICS system being started. It must be unique across all CICS systems that access the same central version.

### Resynchronization between CICS and CA IDMS

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

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

### The Resynchronization Transaction and Program

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

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

- Resynchronization takes place automatically when the interface is started. It resynchronizes with all central versions accessed through the interface and known to CICS as participants in incomplete UOWs.

- When the first database session, for which AUTOCMT is enabled, is connected through the interface to a back-end central version after either system is started, resynchronization takes place automatically for the central version being accessed.

- When the CICS resynchronization transaction (RSYN) is invoked manually, resynchronization takes place for the central version identified in the transaction invocation.

When Should You Manually Resynchronize?

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

The Resynchronization Process

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

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

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

CICS Resynchronization Task Execution

The following syntax is used to execute a resynchronization task:

rsyn Syntax

```sh
rsyntransaction node-name
```

rsyn Parameters

- **rsyn-transaction**
  The name of a CICS resynchronization transaction defined to the CICS system.

- **node-name**
  The name of the CA IDMS central version for which resynchronization is to be performed. The identified system must be accessible through the CICS interface for which the resynchronization transaction was defined.

rsyn Examples

**Successful manual resynchronization example**

The following example shows how manual resynchronization is initiated with central version SYSTEM74 using a resynchronization task called RSYN whose interface module is named IDMSINTC. The resulting messages identify the target node name, the name of the interface module being used, the number of incomplete units of work that need to be recovered and the final outcome of the resynchronization process.

```
RSYN SYSTEM74
CA IDMS Manual 2-PC Resync for IDMSINTC for CV node SYSTEM74 date mm/dd/yyyy
  1 CA IDMS in doubt units of work need recovery for CV node SYSTEM74
  1 CA IDMS in doubt units of work recovery started for CV node SYSTEM74
CA IDMS Two Phase Commit Resync startup completed for CV node SYSTEM74
```

**Unsuccessful manual resynchronization example 1**

This example shows an error condition that occurred during a manual resynchronization because the central version node name was not specified.

```
RSYN
IDMCSYSN error - CV node not specified
```
Unsuccessful manual resynchronization example 2

This example depicts an error condition that occurred during a manual resynchronization because the central version node name that was specified was not available through the CICS interface for which the resynchronization was defined.

RSYN SYSTEM81

CA IDMS Manual 2-PC Resync for IDMSINTC for CV node SYSTEM81 date mm/dd/yyyy
IDMSSCSYN error - Requested CV node SYSTEM81 - Connected CV node SYSTEM74
CA IDMS Two Phase Commit Resync aborted

Successful automatic Resynchronization example

The following example shows the output from an automatic resynchronization initiated when the first request is made to a back-end central version through a CICS interface module or when the interface is started in a CICS Transaction Server for z/OS V2.2 (or later).

CA IDMS Auto 2-PC Resync for IDMSINTC for CV node SYSTEM74 date mm/dd/yyyy
1 CA IDMS in doubt units of work need recovery for CV node SYSTEM74
1 CA IDMS in doubt units of work recovery started for CV node SYSTEM74
CA IDMS Two Phase Commit Resync startup completed for CV node SYSTEM74

Creating the Resynchronization Program

Linking the IDMSCSYN module with an IDMSCINT module creates the resynchronization program. A separate resynchronization program must be created for each version of the CA IDMS interface module (IDMSINTC) that is used within a given CICS system.

In order to simplify future maintenance, the IDMSCSYN module contains minimal executable code. It simply passes control to module IDMSSCCSY which contains most of the executable code to perform resynchronization between CICS and IDMS. If maintenance is applied to IDMSSCCSY, it is not necessary to re-link your resynchronization program.

Resynchronization Program Link Edit (z/OS)

To link a resynchronization program for a CICS environment in z/OS, execute the z/OS Link-Edit JCL (see page 465) inserting the following binder statements.

ORDER DFHEAI,IDMSCSYN
INCLUDE CAGJLOAD(IDMSCSYN)
INCLUDE CUSTLIB(idmscint)
ENTRY CSYNEP1
SETOPT PARM(AMODE=31,REUS(REFR),RMODE=24)
NAME idmsrsyn(R)

- idmscint
  Specifies the name of your IDMSCINT interface module.

- idmsrsyn
  Specifies the name of your resynchronization program.

Resynchronization Program Link Edit (z/VSE)

To link a resynchronization program for a CICS environment in z/VSE, execute the z/VSE Link-Edit JCL, inserting the following binder statements.
PHASE  usercsyn,*
INCLUDE  DFHEAI
INCLUDE  IDMSCSYN
INCLUDE  idmscint
INCLUDE  DFHEA10
ENTRY  CSYNEP1
   // EXEC  LNKEDT,SIZE=128K
/*

- idmscint
  Specifies the name of the idmscint object module.

- usercsyn
  Specifies the user-specified name of the RSYN load module.

Defining a Resynchronization Transaction

A resynchronization transaction must be defined for each IDMSINTC interface to be used within a CICS system. Define the resynchronization transaction to CICS as follows:

DEFINE TRANSACTION(rsyn-transaction-name) PROGRAM(usercsyn)
   GROUP(IDMSGRP) PROFILE(IDMSPRF)
   TASKDATAKEY(CICS)

- rsyn-transaction-name
  Specifies the name chosen for the resynchronization transaction.

- usercsyn
  Specifies the name chosen for the resynchronization program.

The installation default transaction name is RSYN, but another name can be chosen. The name specified in the transaction definition must be identical to the value for the RSYNTXN parameter of the associated interface's CICSOPT macro. For more information on the RSYNTXN parameter, see CICSOPT Syntax.

Defining the Resynchronization Program

A resynchronization program must be defined for each IDMSINTC interface to be used within a CICS system. Define the resynchronization program to CICS as follows:

DEFINE PROGRAM(usercsyn) GROUP(IDMSGRP) LANGUAGE(ASSEMBLER) CEDF(NO) EXECKEY(CICS)

- usercsyn
  Specifies the name chosen for the resynchronization program.

International Character Set Considerations

CA IDMS supports only one single-byte character set in the database engine. The character set (also referred to as code page) definitions reside in module RHDCCODE. The RHDCCODE module is loaded at system startup as part of the nucleus.

This section describes how to customize RHDCCODE to implement a character set other than the default for activation of international characters.
Customizing RHDCCODE

Contents
- DEFBYTE Syntax (see page 430)
  - DEFBYTE Parameters (see page 430)
- Examples (see page 431)
- Assemble and Link Edit RHDCCODE (see page 432)

The default RHDCCODE implements code page 1140 (US EBCDIC) with all international characters (for example, a-grave, e-circumflex, u-diaeresis, and c-cedilla) marked as non-alphabetic.

The following sample members are provided with the installation:

- CP1140F -- United States code page 1140 with all international characters enabled
- CP1141F -- Germany code page 1141 with all international characters enabled
- CP1141R -- Germany code page 1141 with only German characters enabled
- CP1142F -- Denmark/Norway code page 1142 with all international characters enabled
- CP1142R -- Denmark/Norway code page 1142 with only Danish/Norwegian characters enabled
- CP1143F -- Finland/Sweden code page 1143 with all international characters enabled
- CP1143R -- Finland/Sweden code page 1143 with only Finnish/Swedish characters enabled
- CP1145F -- Spanish code page 1145 with all international characters enabled
- CP1145R -- Spanish code page 1145 with only Spanish characters enabled
- CP1147F -- France code page 1147 with all international characters enabled
- CP1147R -- France code page 1147 with only French characters enabled
- CP1148F -- Belgium/Switzerland code page 1148 with all international characters enabled
- CP1148R -- Belgium/Switzerland code page 1148 with only Belgian/Swiss characters enabled

The process of changing RHDCCODE can be divided into the following categories:

- If the desired code page is one of the above sample members, copy the sample member to source RHDCCODE.
- If the installed RHDCCODE does not define the correct attributes you require for your environment, edit source RHDCCODE and change the definitions.
- If the desired code page is not delivered, identify the code page you want to implement and modify source RHDCCODE accordingly.
In all cases, assemble and link RHDCCODE. If you are modifying the RHDCCODE during a new or upgrade installation, see the appropriate section in the Installing section for your operating system. If you are updating RHDCCODE after installation, see Assemble and Link Edit RHDCCODE (see page 432).

**Important!** The tables in RHDCCODE determine the validity of characters accepted by applications and stored in the database. Therefore, as part of the database design, answer these questions:

- Which code page should I implement?
- What is the list of valid characters within the selected code page?

Once a code page is selected and the list of valid characters is made, you should stay with the same definitions because of the following reasons:

- Changing over to another code page might imply conversion of all databases built with the original code page.
- Changing the list of valid characters within a code page is usually possible as long as you add valid characters. However, removing valid characters is usually not possible, unless the database does not contain any of the invalidated characters.

Another concern is uniqueness of data representation. An example is u-diaeresis, a German character that is usually replaced by "ue" if it cannot be typed in. By making the u-diaeresis a valid character, spelling of a word is no longer unique. For example, "Muenchen" and "M\&uumlaut.nchen" both are valid German words for the city of Munich. This problem has to be handled at the application level by, for example, disallowing the "ue" representation of u-diaeresis.

**DEFBYTE Syntax**

Source RHDCCODE must contain 256 #DEFBYTE macros, each defining the attributes of a single byte from x'00' to x'FF'.

```
#DEFBYTE  ebc dic-value,ASCII=ascii-value,TYPE=type
           ,UPPER=uppercase-value   ,PRINT= Y         N
```

**DEFBYTE Parameters**

- **ebcdic-value**
  The EBCDIC value of the byte in hexadecimal notation.

- **ASCII=ascii-value**
  Defines ascii-value as the equivalent ASCII value of ebc dic-value and consequently ebc dic-value as the equivalent EBCDIC value of ascii-value. Specify ascii-value in hexadecimal notation. The ASCII to EBCDIC and EBCDIC to ASCII translate tables are used by IDMSIN01 function STRCONV.
- **TYPE=type**
  Defines the attributes of *ebcdic-value*. Valid values are the following:
  - **ALPHA** -- *ebcdic-value* is an alphabetic character. For example, X'81' (lowercase "a").
  - **HEX** -- *ebcdic-value* is a non-displayable, non-alphanumeric byte. For example, X'0B' (vertical tab).
  - **NUM** -- *ebcdic-value* is a numeric value For example, X'F0' (zero).
  - **VALID** -- *ebcdic-value* is a valid, displayable non-alphanumeric byte. For example, X'50' (ampersand).

- **PRINT=Y|N**
  PRINT is an optional parameter which allows marking *ebcdic-value* as a non-printable byte. If PRINT is omitted, a byte of type HEX is considered non-printable, while all other types are printable.
  - **Y** -- explicitly states that the byte is printable
  - **N** -- explicitly states that the byte is non-printable

- **UPPER=uppercase-value**
  UPPER is an optional parameter that defines *uppercase-value* as the uppercase translation of *ebcdic-value* and consequently *ebcdic-value* as the lowercase translation of *uppercase-value*. If UPPER is omitted, no translation takes place, that is, *uppercase-value* = *ebcdic-value*. The uppercase and lowercase translate tables are used to translate input to uppercase, but also by, for example, built-in functions like TOUPPER, TOLOWER, and WORDCAP.

### Examples

```
#DEFBYTE 81,ASCII=61,TYPE=ALPHA,UPPER=C1
```
Defines hexadecimal EBCDIC byte X'81' with ASCII equivalent X'61'. TYPE=ALPHA indicates that the byte is alphabetic, which implies non-numeric, displayable and printable. UPPER=C1 indicates that the EBCDIC uppercase value is X'C1'.

```
#DEFBYTE 05,ASCII=09,TYPE=HEX
```
Defines hexadecimal EBCDIC byte X'05' with ASCII equivalent X'09'. TYPE=HEX indicates that the byte is non-alphabetic, non-numeric, non-displayable and non-printable.

```
#DEFBYTE F0,ASCII=30,TYPE=NUM
```
Defines hexadecimal EBCDIC byte X'F0' with ASCII equivalent X'30'. TYPE=NUM indicates that the byte is numeric, which implies non-alphabetic, displayable and printable.

```
#DEFBYTE 6F,ASCII=3F,TYPE=VALID
```
Defines hexadecimal EBCDIC byte X'6F' with ASCII equivalent X'3F'. TYPE=VALID indicates that the byte is non-alphabetic, non-numeric, displayable and printable.
Assemble and Link Edit RHDCCODE

To make the customized RHDCCODE available for use, assemble and link edit the RHDCCODE module for the system as follows:

1. Create a RHDCCODE source module as described previously.
2. Save the source module in your custom source library.
3. Assemble and link the module into your custom load library by executing the z/OS assemble and link-edit JCL. Substitute the name of your RHDCCODE source member and insert the following binder statements:
   
   ```
   INCLUDE CAGJLOAD(RHDCCODT)
   ENTRY CODEEP1
   NAME RHDCCODE(R)
   ```

To create a z/VSE RHDCCODE module:

1. Assemble and catalog the module using the sample JCL in z/VSE Assemble JCL (see page 476).
   Modify the JCL by substituting the following in place of the Assembler input statements:
   
   ```
   PUNCH 'CATALOG rhdccode.OBJ REPLACE=YES'
   END
   ```

2. Link the RHDCCODE program using the sample JCL in z/VSE Link JCL (see page 476).
   Modify the JCL by substituting the following statements in place of the Linkage editor control statements:
   
   ```
   PHASE RHDCCODE,*
   INCLUDE rhdccodeENTRY CODEEP1
   ```

   - `rhdccode`
     Specifies the name of the object module containing the assembly output.

z/VM. RHDCCODE assembly and link edit

GLOBAL MACLIB idmslib
FILEDEF TEXT DISK RHDCCODE TEXT A
ASSEMBLE rhdccode.source (NODECK OBJECT

FILEDEF SYSLST PRINTER
FILEDEF SYSLMOD DISK idmslib LOADLIB a2 (RECFM V LRECL 1024 BLKSIZE 1024
LKED linkctl

Linkage editor control statements (linkctl):

```
INCLUDE RHDCODT
INCLUDE RHDCCODE
ENTRY CODEEP1
NAME RHDCCODE(R)
```

<table>
<thead>
<tr>
<th>idmslib</th>
<th>filename of the CA IDMS MACLIB library</th>
</tr>
</thead>
<tbody>
<tr>
<td>idmslib LOADLIB a2</td>
<td>file identifier of the CA IDMS LOADLIB library</td>
</tr>
<tr>
<td>linkctl</td>
<td>filename of the file containing the linkage editor control statements</td>
</tr>
</tbody>
</table>
Change tracking enables changing the database environment of a Central Version (CV) in a fault-tolerant manner. Specifically, it enables the DBA to perform the following actions:

- Vary the data set name of a journal or database file within a CV without introducing the potential for a warmstart failure
- Vary a new version of a DMCL without introducing the potential for a warmstart failure
- Vary the status of an area or segment permanently on a CV without regard to subsequent page range changes
- Change the journal files in use by a CV and coordinate those changes with the associated archive journal jobs

Change tracking also provides an easy way for a local mode job to use the same database environment (definition and data sets) as a CV, even if that environment has been impacted by dynamic modifications.

**Change Tracking and SYSTRK Files**

To track changes, CV maintains a description of its database environment in a type of file called a SYSTRK file. The presence of such a file in the execution JCL of a CV triggers change tracking by that CV. A local mode job, such as a journal archive job, can share the description of the CV’s database environment by referencing the same SYSTRK file in its execution JCL.

A SYSTRK file holds a description of the database environment most recently in use by the CV. During startup, an image of the current DMCL is written to SYSTRK along with information on database and journal files defined in the JCL. If DCMT commands issued during CV execution cause critical changes to its database environment, SYSTRK is updated to reflect those changes. If the CV fails, the runtime database definition is restored from SYSTRK during restart, ensuring that the files being updated at the time of failure are the ones recovered by warmstart unless explicitly overridden by changes in the JCL used to restart the CV.

Change tracking is optional. If no SYSTRK file is referenced in the execution JCL of the CV, change tracking is not in effect, meaning that the potential for a warmstart failure is introduced when varying in a new copy of a DMCL or dynamically changing the data set name of a file. Additionally, any permanent status established for an area whose page range is changed is lost or may be misapplied to another area whose page range is also changed.

Change tracking can be temporarily inactivated or disabled for a CV to facilitate expansion or replacement of a SYSTRK file. However, doing this impacts the ability to dynamically change the database environment.

- Inactivating change tracking has the effect of disallowing DCMT commands that would otherwise require updating SYSTRK.

- Disabling change tracking allows such commands to be executed, but a warning is issued indicating that manual intervention will be needed to restart CV should it fail before change tracking is re-activated.
Implementing Change Tracking

To implement change tracking for a CV, take the following steps:

1. Create and format two to four SYSTRK files. A minimum of two SYSTRK files are needed because mirroring is used to provide fault tolerance and recoverability in case of file damage.

2. Alter CV execution JCL to reference the SYSTRK files.

3. Alter the JCL for the associated archive journal job to also reference the SYSTRK files and to remove references to the disk journal files.

4. Optionally, change the JCL of other local mode jobs to reference the SYSTRK files and remove explicit references to database files.

More Information

- For more information on sizing and formatting SYSTRK files, see the FORMAT utility statement in the CA IDMS Administrating section.

- For more information on options for altering CV startup JCL to reference SYSTRK files, see Referencing SYSTRK Files in Execution JCL.

Referencing SYSTRK Files in Execution JCL

SYSTRK files are referenced in execution JCL using file assignments whose DDname begins with the value specified by the SYSIDMS parameter: SYSTRK_DDNAME_PREFIX. The default value for this parameter is SYSTRK.

Depending on your operating system, you can reference SYSTRK files in execution JCL by including one of the following:

- A model SYSTRK file assignment

- A file assignment for each SYSTRK file to be used

Referencing SYSTRK files using a model is the recommended approach because it enables the set of active SYSTRK files to be changed without impacting execution JCL.

Using a Model SYSTRK File Assignment

A model SYSTRK file assignment has a DDname that is the SYSTRK_DDNAME_PREFIX. It references a data set whose name is used as the prefix for constructing the names of the real SYSTRK files by appending a numeric suffix ranging from 1 to 9 to the end of the model's data set name.

For example, if the SYSTRK_DDNAME_PREFIX is SYSTRK and a model SYSTRK file assignment references a data set name of DBDC.SYSTEM73.SYSD with a disposition of SHR, CA IDMS attempts to discover through dynamic allocation the data sets shown in the following table:
The presence of a file assignment whose DDname is SYSTRKn overrides the generated data set name and disposition. If an overriding file assignment refers to a dummied file, the overridden file is not used.

If a model SYSTRK file assignment refers to a dummied file, it is equivalent to not including a model file assignment in the execution JCL.

⚠️ **Note:** The data set referenced by a model SYSTRK file assignment is never opened. While the file must exist, its contents are not relevant.

⚠️ **Note:** In z/VSE if a model SYSTRK label is used, the individual SYSTRK files must be defined in system labels, or cataloged in a CA DYNAM catalog. Otherwise it is recommended to use individual SYSTRK file assignments. If the IDMSLBLs JCL procedure is used, you may wish to add a SYSTRK model or individual file assignments here.

### Using Individual SYSTRK File Assignments

The DDNAME for an individual SYSTRK file assignment is the SYSTRK_DDNAME_PREFIX suffixed with a digit from 1 to 9. For example, if the SYSTRK_DDNAME_PREFIX is SYSTRK, the DDNAMEs that can be used for SYSTRK files are SYSTRK1, SYSTRK2, . . . SYSTRK9.

Using this approach to reference SYSTRK files requires that a file assignment be included in the JCL for each SYSTRK file to be used. To change the set of files being used, you must update every set of JCL in which they are referenced.

⚠️ **Note:** For more information on SYSIDMS parameters, see the *Common Facilities Guide*.

---

**Managing Change Tracking**

**Contents**

- Expanding SYSTRK Files (see page 436)
- More Information (see page 437)
The following facilities are provided for managing change tracking:

- You can monitor the status of change tracking in a DC/UCF system and determine the SYSTRK files that are in use by issuing a DCMT DISPLAY CHANGE TRACKING command.

- You can alter the status of change tracking and its associated SYSTRK files by issuing a DCMT VARY CHANGE TRACKING command.

- You can increase the size of the SYSTRK files by using the FORMAT utility statement in conjunction with the DCMT VARY CHANGE TRACKING command.

### Expanding SYSTRK Files

A set of SYSTRK files can be expanded by using the FORMAT utility statement. The procedure for expanding files while CV remains active differs depending on whether the SYSTRK files are referenced through a model SYSTRK file assignment or if they are referenced using individual file assignments.

#### Expanding Files Referenced Through a Model SYSTRK File Assignment

Assuming two SYSTRK files are in use, take the following steps to increase the size of the SYSTRK files while the CV remains active:

1. Allocate larger SYSTRK files using data set names that conform to the standard established by the model DD statement.

2. Format the larger files by executing a FORMAT utility statement as follows:
   ```
   FORMAT SYSTRK DD1, DD2 INITIAL LIKE DD3 EXPAND 20 PERCENT
   ```

   Where DD1 and DD2 are the DDnames of file assignments referencing the new SYSTRK files, and DD3 is the DDname of a file assignment referencing one of the old SYSTRK files. In this example, the files are being expanded 20 percent over their current size.

3. Replace use of the old files with the new files by issuing the following command:
   ```
   DCMT VARY CHANGE TRACKING REFRESH
   ```

   When the old files are no longer in use, they can be deleted.

#### Expanding Files Referenced Through Individual File Assignments

Assuming two SYSTRK files are in use, take the following steps to increase the file size while the CV remains active:

1. Allocate larger SYSTRK files using new data set names.

2. Close and deallocate the current set of SYSTRK files by issuing the following command:
   ```
   DCMT VARY CHANGE TRACKING INACTIVE
   ```

3. Format the larger files by executing a FORMAT utility statement as follows:
   ```
   FORMAT SYSTRK DD1, DD2 INITIAL LIKE DD3 EXPAND 20 PERCENT
   ```
Where DD1 and DD2 are the DDnames of file assignments referencing the new SYSTRK files, and DD3 is the DDname of a file assignment referencing one of the old SYSTRK files. In this example, the files are being expanded 20 percent over their current size.

4. Scratch the old files. Rename the new files to have the same data set names as the old files.

5. Allocate the new files and make change tracking active by issuing the following command:
   ```
   DCMT VARY CHANGE TRACKING ACTIVE
   ```

More Information

- For more information on DCMT commands, see the *System Tasks and Operator Reference section*.
- For more information on the FORMAT utility statement, see the *Administrating section*.

Terminating the Use of Change Tracking

Once change tracking has been initiated for a CV, its use should be continued indefinitely; otherwise, permanent area or journal statuses will be lost. Despite this, if you choose to discontinue the use of change tracking, do so as follows:

- Shutdown CV
- Remove all references to the SYSTRK files in the execution JCL of the CV and other jobs
- After restarting CV, use DCMT commands to re-establish permanent area statuses

⚠️ **Note:** You can temporarily disable change tracking by issuing a DCMT VARY CHANGE TRACKING command or by overriding the SYSTRK file assignments to reference a dummy file.

⚠️ **Note:** For more information on DCMT commands, see the *System Tasks and Operator Reference section*.

IDMSINFO Service Provider

The IDMSINFO service provider is a stand-alone address space that provides services through stacking PC routines. This service provider handles space switch PC requests using cross memory communication.
The IDMSINFO service provider is available only on the z/OS platform.

The IDMSINFO service provider is a long-running address space that:

- Consumes few system resources
- Runs authorized
- Provides IDMS performance and status information to monitoring tools
- Provides IDMS resource manipulation to monitoring tools

The IDMSINFO service provider must be active to be able to use the remote monitoring feature of Performance Monitor.

For IDMSINFO to initiate correctly, it is necessary to have run CAIRIM to install the SVC and related APFLIB modules using GJISINIT. When CAIRIM installs the SVC using GJISINIT and the APFLIB, GJISINIT loads several modules in addition to the SVC. One of the modules is PMRTDATA, which is a module in APFLIB. This module contains the IDMSLPAR DSECT, which IDMSINFO requires. If PMRTDATA has not been loaded into LPA by running the CAIRIM, IDMSINFO returns a DC130051 error.

The IDMSINFO Service Provider as a Started Task

If you run the IDMSINFO service provider as a started task (STC), you can request a reusable ASID to be associated with the address space. A reusable ASID that has been assigned to the IDMSINFO service provider address space is free for reuse when the address space is terminated. Being free for reuse prevents losing the ASID from use during an IPL.

Initiate IDMSINFO as a Started Task

To be able to request a reusable ASID, initiate the IDMSINFO service provider as a started task.

Follow these steps:

1. Ensure that a procedure is created, given the appropriate authorities and stored in a site-specific PROCLIB (see the IDMSINFO member of the CAGJSAMP library for sample JCL).
2. Issue the START command with the reuse ASID parameter. The START command has the following format:

   S procname,REUSASID=YES

   This command gives control to the initial program that executes in the address space.

   To terminate the IDMSINFO service provider, issue the STOP or P command.

   **Note:** The IBM message IEF352I ADDRESS SPACE UNAVAILABLE is issued at termination:

   - If the REUSASID is not specified for a started task
   - or
   - If the IDMSINFO service provider is run as a batch job.

   **Important!** We recommend not performing any action repeatedly that returns the IEF352I message. Doing so eventually results in the depletion of all available address spaces and necessitates a re-IPL of the z/OS system.

# SQL Web Connect Considerations

If you use CA IDMS™/DB, you can take advantage of SQL, ODBC, and JDBC connectivity functions using the SQL Web Connect feature. SQL Web Connect provides access for two concurrent users of dynamic SQL through ODBC and JDBC from the Web, Windows, Java, and other non-mainframe platforms, and through the usual CA IDMS tools (for example, OCF, BCF).

- Requirements (see page 440)
- Limitations (see page 440)
- Registration Monitoring (see page 441)
- CA IDMS Server Product Installation (see page 441)
- Feature Activation (see page 442)

SQL Web Connect is beneficial especially for the following uses:

- Prototyping
- Pilot projects
- Application development
- Low-volume application usage
SQL Web Connect works by tracking product usage across all CVs for a given LPAR. Two concurrent users of both CA IDMS™ SQL and CA IDMS™ Server are permitted. When two simultaneous users are registered, subsequent attempts to use these product options results in a failure as dictated by the current RHDCPINT product bit setting, or an LMP warning if a license is required for these options.

**Note:** If you apply the enhancement PTF RO96807, SQL Web Connect is no longer necessary, because the LMP keys for CA IDMS SQL and CA IDMS Server product options are contained in the base CA IDMS product.

On z/OS, usage is tracked by the IBM Software Product Registration function. Calls to the IBM Software Product Registration facility are handled using the LMP Seat License Registration Service within CA Common Services (CCS).

### Requirements

The SQL Web Connect feature uses TCP/IP to communicate with CA IDMS. This means:

- JDBC users must use a Type-4 JDBC connection.
- ODBC users must configure their Data Source to use the CA IDMS communications protocol.
- Installation of CAICCI is not required on the mainframe or the client.

The use of the mentioned communication interfaces requires the installation of CA IDMS Server, as follows:

- JDBC users must run CA IDMS Server r16.0 or greater.
- ODBC users must run CA IDMS Server r17.0 or greater.

For z/OS only: The CA LMP Seat License Registration Service is a component of CA Common Services. The CA Common Services releases that contain the Seat License Registration Service are:

- CCS r14.1 (or greater)
- CCS r14.0 with PTF RO33298
- CCS r12.0 with PTF RO40748

You must be on one of these releases (with the appropriate PTF applied if necessary) in order to use the SQL Web Connect feature.

### Limitations

Full licenses for CA IDMS SQL and CA IDMS Server are needed for:

---

23-Dec-2017

CA IDMS - 19.0
Registration Monitoring

Monitoring SQL and ODBC/JDBC user registrations differs depending on the operating system in use.

**z/OS Users**

z/OS users can monitor SQL and ODBC/JDBC user registrations in two ways:

- Using the ‘REGPROD’ command within SYSVIEW
- Using the ‘DISPLAY PROD’ z/OS operator command

For SYSVIEW users, it can be helpful to sort the REGPROD screen using the ‘SORT ID’ command. Users of the ‘DISPLAY PROD’ command can tailor the list of registered products that are returned using the ‘ID(xx)’ option of the ‘DISPLAY PROD’ command. The following product IDs are used on z/OS:

- **GT** - For the SQL portion of SQL Web Connect
- **XS** - For the ODBC/JDBC portion of SQL Web Connect

**z/VSE Users**

On z/VSE, the ‘LOCK SHOW,PROD*’ operator command can be used to monitor user registrations. The following product IDs are used on z/VSE:

- **1D** - For the SQL portion of SQL Web Connect
- **05** - For the ODBC/JDBC portion of SQL Web Connect

CA IDMS Server Product Installation

Customers who must install the CA IDMS Server client component can acquire the software in the following ways:

1. Download the CA IDMS Server product directly from the “Download Center” page on CA Support Online, at: [https://support.ca.com/](https://support.ca.com/). Select the product “CA IDMS/DB - MVS” and the latest version/release number from the drop-down lists and click on the “Go” button. On the following screen, click the “Download” function next to the “IDMS SRVR CD ZIP FILE ESD ONLY” item.

2. The install image for CA IDMS Server is included in the current CA IDMS installation PAX file. Once the SMP ‘APPLY’ step has been run, this image is available in the CAGJDATV SMP library under member IDSERV17. You must FTP this member (in BINARY mode) to an appropriate location on the hard drive of your PC. Once downloaded, rename the file to have an extension of .zip.
Once the ZIP file is downloaded, extract the installation files and then invoke the setup.exe file to initiate the installation.

⚠️ **Note:** For more information on the configuration and usage of CA IDMS Server, see Installing IDMS Server (https://docops.ca.com/display/IDMS19/Installing+IDMS+Server).

### Feature Activation

No action is needed in order to activate the SQL Web Connect feature, provided the specified requirements (https://docops.ca.com/display/IDMS19/SQL+Web+Connect+for+VSE#SQLWebConnectforVSE-Requirements) are met. Usage tracking automatically takes place when the following two conditions are fulfilled:

- An ODBC/JDBC connection or an SQL session (or both) is established.
- The system determines that you are not licensed for the product in use.

No LMP warnings occur for connections and sessions that are established under SQL Web Connect.

Users who have not already created an SQL Catalog for their system (for use with CA IDMS Visual DBA for example) need to do so. You may also want to create and populate the SQL Demo database on one or more of your CVs.

⚠️ **Note:** For more information on the steps needed to perform these tasks, see the CA IDMS installation section for your operating system.

### z/OS Considerations

#### Maximum Number of Files

Normally, a z/OS job step can access up to 3,273 files. CA IDMS has extended this limit for a CV, to allow up to 10,000 files to be accessed using dynamic allocation and 3,273 files to be accessed using DD statements. If more than 3,273 files are to be accessed, the excess files must be defined for dynamic allocation.

⚠️ **Note:** Because the maximum number of DD statements that can be associated with a job step is 3273, if the number of database files in a DMCL is close to or exceeds this limit, dynamic allocation should be used for all database files so that the limit will not prevent the use of DD statements to override dynamically allocated files when necessary.
z/OS Performance Considerations

When running a DC/UCF system under z/OS, system performance can be enhanced by using selected z/OS features. The following considerations apply:

- Running non-swappable -- For optimal performance, always run a DC/UCF system non-swappable. The DC/UCF system must be run non-swappable if:
  - Any dial-up lines are defined.
  - You execute COBOL II or PL/I programs.
  - You want to monitor this system from a remote CA IDMS Performance Monitor session.

By default, the DC/UCF system runs non-swappable. To run swappable, specify SWAP=Y in the PARM field on the EXEC statement or specify 'S' in column 24 of the PARM field. For more information on the PARM card, see Specifying Runtime Options (see page 508).

- Page fencing -- Performance can be enhanced by fencing off an area of real storage for exclusive use by the DC/UCF system.

- Performance groups -- z/OS allows the assignment of performance groups. You can use this feature to assign the DC/UCF system a high priority. The DC/UCF system should be assigned a performance group:
  - Just below that of VTAM
  - Above other TP access methods and batch performance groups.
  - Above CICS and TSO.

- Data spaces -- You can use data spaces in an ESA environment to provide faster access to high-activity database files.

LPA eligibility

Most nucleus modules are reentrant and are eligible to be put in the link pack area (LPA). To obtain a complete list of eligible modules, run a load library utility that lists the attributes of library members against your CA IDMS load library. All members that are marked REENTRANT are LPA eligible with the following exceptions:

- IDMSNLT
- RHDCCSA
- RHDCNLT
- RHDCNTRY
- RHDCOS00
Note: RHDCBANR, as distributed, is eligible for the LPA; if you modify the source, it is no longer reentrant.

User mode programs (anything with a #BALI or IDMSBALI linked with it) cannot be in the LPA. However, non-executable load modules can be in the LPA. These include CA ADS dialogs, ADSA application load modules, maps, subschemas, and IDD tables.

Important! Some utilities modify the set membership options for the subschema tables. Therefore, the subschema to be used by Step IDMSDBL2 within the RELOAD and FASTLOAD utilities and by Step IDMSDBL4 within the RELOAD, FASTLOAD, MAINTAIN INDEX, and MAINTAIN ASF utilities must not be linked with the attribute reentrant in the LPA.

Any other module that might be loaded but which doesn’t issue CA IDMS calls is eligible, provided it is truly reentrant.

Note: CA Technologies does not recommend placing eligible modules in the LPA. If modules from one release of CA IDMS are in the LPA, they may inadvertently be accessed when running another release of CA IDMS with unpredictable results.

Note: The size of the LPA has a direct effect on the region size available for the private address space. Consult with the operating system systems programmer to assign the size.

zIIP Exploitation

CA IDMS exploits zIIP processors on the z9 series and above for the z/OS operating system. This feature enables offloading computing cycles to zIIPs, thereby increasing overall CPU throughput at lower operational costs. The zIIP feature is not dependent on any other CA IDMS feature, including multitasking.

Note: z/OS software feature HBB7709 is required to use the zIIP feature.

The following sections discuss topics related to zIIP exploitation:
The default mode of operation is to not use zIIP processors unless specifically requested at runtime. A new ZIIP startup parameter is available to enable or disable the use of these processors by CA IDMS. To facilitate analysis of the potential benefit, the feature can be enabled even if no zIIP processors are available. For more information, see Specifying Runtime Options (see page 508).

If this feature is enabled, CA IDMS uses Workload Manager to create a dependent enclave for each OS task capable of servicing work type IDMS. It then schedules a separate pre-emptable SRB into each such enclave. See DCMT DISPLAY SUBTASK (https://docops.ca.com/pages/viewpage.action?pageId=357342618).

Note: On systems utilizing zIIP processors, CPU time in CA IDMS statistics includes time on the zIIP processor normalized to standard processor speed.

zIIP Eligibility

Most CA IDMS system code is eligible to run on a zIIP processor. However, user exits, database procedures, SQL-invoked routines, and application programs are not eligible to run on a zIIP processor. CA IDMS runtime processing ensures that a non-zIIP processor is selected to run routines not eligible for zIIP.

To ensure that only eligible modules are selected to be run on a zIIP processor, some load modules must be loaded from one of the following secured locations:

- An authorized load library named in the STEPLIB concatenation or in the CDMSLIB concatenation. A library is authorized by adding it to the list of APF-authorized libraries in the appropriate PROGxx or IEAAPFxx member in SYS1.PARMLIB.

- The Link Pack Area, which includes the following modules:
  - Dynamic LPA modules, as specified in PROGxx members in SYS1.PARMLIB
  - Fixed LPA (FLPA) modules, as specified in IEAFIxXX members
  - Modified LPA (MLPA) modules, as specified in IEALPAXxx members
  - Pageable LPA (PLPA) modules, loaded from libraries specified in LPALSTXX or PROGxx members
  - A library in the linklist, as specified in PROGxx and LNKLSTxx members.

For more information on authorized libraries, the LPA, and the linklist, see the IBM documentation.
The specific rules for load module residence for zIIP processing are as follows:

- **RHDCOMVS**, the load module that is executed to start the CA IDMS CV must reside in an authorized library in the STEPLIB concatenation or in a linklist library.

- At CV startup, CA IDMS nucleus modules, including all line drivers and service drivers, must be loaded from an authorized load library in the CDMSLIB concatenation or from the LPA.

- The IBM language environment library (usually CEE.SCEERUN) must be authorized if it is included in the CDMSLIB. Alternatively, language environment modules can reside in the LPA.

- Any client-supplied program that runs in system mode must come from an authorized library. This includes all named and numbered user exits, database procedures, and any SQL procedures or table procedures that run in system mode.

- z/OS Callable Services library (SYS1.CSSLIB) must be in the linklist or it must be authorized and included in the STEPLIB concatenation.

CA Technologies does not recommend using the LPA or linklist for modules that are supplied during the CA IDMS installation. Doing this makes maintaining these modules difficult and can cause CA IDMS to use a module that was created for a different release.

Modules that consist of non-executable code and user mode programs do not have to come from a secured location. This category includes the following:

- Client-written code that runs in user mode, including assembler and high-level language application programs, CA ADS dialogs, table procedures, and SQL routines, if they are defined as user mode routines

- DMCL load modules

- Database name tables

- Control blocks or tables that contain no executable code

It is not necessary to authorize individual nucleus members in a load library, and these members should not be linked with SETCODE AC(1). The startup module (RHDCOMVS or site-linked startup module) must be linked with SETCODE AC(1) only if the AUTHREQ parameter is specified for the CA IDMS SVC. For more information on the AUTHREQ parameter, see [Generating the SVC](#) (see page 77).

⚠️ **Note:** Not every load library in the CA IDMS startup STEPLIB and CDMSLIB must be authorized. Only the libraries from which the nucleus modules are loaded must be authorized. Startup error messages identify modules that prevent zIIP exploitation.

To ensure that all nucleus modules are loaded from an authorized library, take one of the following actions:

- Authorize CAGJLOAD and your custom.loadlib created during the installation of CA IDMS.
Manually copy all modules from those libraries to corresponding authorized libraries use by CV startup. Recopy all modules from CAGJLOAD whenever maintenance is applied.

### zIIP Suspension

CA IDMS automatically suspends zIIP when the system detects the load of certain programs from an unauthorized load library. This applies to the following program types:

- Database procedures
- SQL procedures
- User Exits
- IDMS Nucleus modules

You can determine the state of zIIP processing using the DCPROFIL task. A zIIP designation of 'U' indicates that zIIP processing is suspended due to the load of a program from an unauthorized library.

zIIP suspension allows zIIP to be turned back on. Before you re-enable zIIP, follow these steps:

1. Verify that the module does not include malicious content.
2. If you verify that the module is safe, move the module to an authorized library.

After you verify that the module is safe, and it is moved to an authorized library, you re-enable zIIP using the DMCT VARY zIIP ON command.

⚠️ **Security risk warning:** The Database or Security Administrator needs to thoroughly check the unauthorized module before moving it to an authorized library. If this does not occur and zIIP processing is re-enabled, you run a significant security risk and may allow malicious content to spread throughout your z/OS system environment.

We highly recommend that the use of the DCMT VARY ZIIP command be tightly controlled. CA IDMS/DC permits the assignment of a discrete security class to each of the DCMT commands, thereby allowing the administrator to limit the use of the VARY ZIIP command to authorized persons only. For more information on discrete security for the DCMT task, see Administering Security for IDMS.
Batch Considerations

When CA IDMS is used with a batch program, no modules in the batch region are made eligible for zIIP. However, there are considerations that arise from the use of an authorized load library. The z/OS operating system enforces certain rules for programs that are loaded from a set of authorized load libraries.

Any program that is linked with the RENT attribute cannot be modified at runtime. If this rule is violated, an SOC4 program check occurs. Application programs that are linked to the CA IDMS interface module, IDMS, are modified at runtime by CA IDMS. Therefore, the batch STEPLIB concatenation should contain at least one non-authorized load library, or such user programs should be linked without the RENT attribute.

Programs that are supplied by CA Technologies, such as IDDSDDDL, are linked appropriately in the SMP/E target load library, so no special action is required for these programs.

Evaluating the zIIP Feature Benefits

The best way to estimate the benefit of the zIIP feature is to try it on a representative system with an available ZIIP engine for a period of time. To do so, use ZIIP=Y at startup.

⚠️ **Note:** Running with ZIIP=Y on a system without a ZIIP engine can result in performance overhead and inaccurate estimates.

Several easy steps are used to determine the benefits that can be achieved by using the zIIP feature:

1. Run the system with ZIIP=N using your preferred performance test stream.
2. Record the results of DCMT DISPLAY SUBTASK EFFECTIVENESS. Using a UCFBATCH program is a good method for obtaining this information.
3. Run the system with ZIIP=Y using your preferred performance test stream.
4. Record the results of DCMT DISPLAY SUBTASK EFFECTIVENESS. Using a UCFBATCH program is a good method for obtaining this information.
5. Compare the TCB column from Step 2 with that from Step 4. The difference is proportional to the potential reduction in both the total CPU use and the Total Cost of Ownership (TCO) that can be achieved by using the zIIP feature.

The SRB column from Step 4 is proportional to the number of MIPs of zIIP processing power that will be required to achieve these cost reductions.
Examples

The following displays indicate that each run used between 93 and 98 CPU seconds of total normalized CPU. The second run shows that 91.1222 CPU seconds out of a total of 93.5108 CPU seconds were offloaded to an SRB. Approximately one third of this SRB CPU time can be offloaded to a zIIP processor. For this particular application mix, this means that approximately one third of the total CV CPU could be offloaded to zIIP processors.

Since a zIIP processor was present, the actual offloaded CPU can then be confirmed from the JES LOG Step End messages, IEF374I, which, in this case, indicates a total CPU reduction of 31.68 seconds.

Step 2 Output with ZIIP=N

*** Subtask display ***

<table>
<thead>
<tr>
<th>Subtask</th>
<th>Elapsed time</th>
<th>Total CPU time</th>
<th>% CPU</th>
<th>SRB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TCB SRB</td>
<td>TCB SRB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAINTASK</td>
<td>00:00:14.0505</td>
<td>00:00:00.0000</td>
<td>00:00:02.3699</td>
<td>16 N/A N</td>
</tr>
<tr>
<td>SUBT0001</td>
<td>00:00:00.0122</td>
<td>00:00:00.0000</td>
<td>00:00:00.0104</td>
<td>85 N/A N</td>
</tr>
<tr>
<td>SUBT0002</td>
<td>00:00:00.0175</td>
<td>00:00:00.0000</td>
<td>00:00:00.0131</td>
<td>74 N/A N</td>
</tr>
<tr>
<td>SUBT0003</td>
<td>00:00:00.2348</td>
<td>00:00:00.0000</td>
<td>00:00:00.0398</td>
<td>16 N/A N</td>
</tr>
<tr>
<td>SUBT0004</td>
<td>00:00:00.2175</td>
<td>00:00:00.0000</td>
<td>00:00:00.0024</td>
<td>11 N/A N</td>
</tr>
<tr>
<td>SUBT0005</td>
<td>00:01:42.0081</td>
<td>00:00:00.0000</td>
<td>00:01:35.8538</td>
<td>93 N/A N</td>
</tr>
</tbody>
</table>

Totals    00:01:56.5406 00:00:00.0000 00:01:38.3110 00:00:00.0000 84 N/A

JES LOG Step End Message

IEF374I STEP/DCV /STOP 2008242.0425 CPU 1MIN 39.53SEC SRB
0MIN 11.73SEC VIRT 7840K SYS 552K EXT 56072K SYS 11460K

Step 4 Output with ZIIP=Y

*** Subtask display ***

<table>
<thead>
<tr>
<th>Subtask</th>
<th>Elapsed time</th>
<th>Total CPU time</th>
<th>% CPU</th>
<th>SRB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TCB SRB</td>
<td>TCB SRB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAINTASK</td>
<td>00:00:12.9116</td>
<td>00:00:00.0434</td>
<td>00:00:02.1307</td>
<td>16 120 Y</td>
</tr>
<tr>
<td>SUBT0001</td>
<td>00:00:00.0125</td>
<td>00:00:00.0000</td>
<td>00:00:00.0111</td>
<td>88 N/A Y</td>
</tr>
<tr>
<td>SUBT0002</td>
<td>00:00:00.0118</td>
<td>00:00:00.0002</td>
<td>00:00:00.0103</td>
<td>87 00 Y</td>
</tr>
<tr>
<td>SUBT0003</td>
<td>00:00:00.1643</td>
<td>00:00:00.0014</td>
<td>00:00:00.0624</td>
<td>37 114 Y</td>
</tr>
<tr>
<td>SUBT0004</td>
<td>00:00:00.0276</td>
<td>00:00:00.0004</td>
<td>00:00:00.0239</td>
<td>86 200 Y</td>
</tr>
<tr>
<td>SUBT0005</td>
<td>00:00:00.3849</td>
<td>00:01:17.1037</td>
<td>00:00:00.1422</td>
<td>36 118 Y</td>
</tr>
</tbody>
</table>

Totals    00:00:13.5127 00:01:17.1491 00:00:02.3886 00:01:31.1222 17 118

JES LOG Step End Message

IEF374I STEP/DCV /STOP 2008242.0400 CPU 1MIN 01.60SEC SRB
0MIN 17.98SEC VIRT 7840K SYS 552K EXT 56080K SYS 11500K

Storage Key Considerations for z/OS CSA Subpools

The AllowUserKeyCSA parameter was introduced in z/OS V1.8 to prevent jobs from allocating storage from CSA subpools using user keys 8 through 15. You can specify AllowUserKeyCSA(YES) or AllowUserKeyCSA(NO). The default at z/OS V1.8 or lower is AllowUserKeyCSA(YES). Starting at V1.9, the default is AllowUserKeyCSA(NO).
You can determine the current setting of AllowUserKeyCSA by issuing the D DIAG command from the z/OS console.

If YES is specified, there is no impact on CA IDMS or jobs that communicate with CA IDMS through the External Run Unit System (Batch to CV, CICS, etc).

If NO is specified, the CA IDMS system abends during startup with a system abend code of B78-5C unless the following steps are taken:

1. Place the following modules in a special load library:
   - RHDCOMVS
   - RHDCTCKR
   - RHDCCKUR

   Alternatively, the load library can be a copy of the installed or deployed z/OS CA IDMS load library.

2. Authorize that load library by adding a control statement to the appropriate SYS1.PARMLIB (PROG..) member, for example, APF ADD DSNNAME(my.apflib) VOLUME(vvvvvv).

3. Update the appropriate SYS1.PARMLIB(SCHED..) member to include the following line:
   
PPT PGNAME($STARTUP),KEY(4),NOSWAP
   
   - $STARTUP
     Specifies the CA IDMS/DC startup module name - RHDCOMVS, IDMSDC, or IDMSDCV.

4. In the CA IDMS startup JCL, ensure that the library containing the startup module, RHDCTCKR, and RHDCCKUR is the only library specified in STEPLIB.

5. Specify STEP=Y on the EXEC parm as follows:

   **Example**
   
   //IDMSDC EXEC PGM=RHDCOMVS,REGION=0M,TIME=1440,
   //      PARM='S=100,STEP=Y
   //
   
   You can also specify that CA IDMS is to load RHDCCKUR and RHDCTCKR from STEPLIB using the positional "S" startup parameter as follows:

   **Example**
   
   1 2 3 4
   1234567890 1234567890 1234567890 1234567890
   //IDMSKY EXEC PGM=RHDCOMVS,J,REGION=0K,TIME=1440,
   //      PARM='5=100,STEP=Y
   //

   **Note:** "S" in position 10 following the system number instructs the CA IDMS system to load RHDCCKUR and RHDCTCKR from STEPLIB.
To verify that key 4 is in fact being used, you can run the DCPROFIL task. See the field labeled PRIMARY STORAGE PROTECT KEY.

⚠️ **Note:** If an abend should occur during CA IDMS system initialization, it is possible that a complete system dump will not be captured because a system key is being used instead of a user key. The operating system design is not to dump storage for keys 0 to 7 (system keys).

To ensure complete dumps are always captured, the user associated with the startup of the CA IDMS system needs to be granted READ access to facility IEAABD.DMPAKEY. For more information on the procedures to grant access to a facility, see the appropriate security sub-system documentation.

## Dump Options for Abnormal System Termination

When a CA IDMS CV abends, depending on the conditions, it is possible that a dump is not available from which to research the abort. For example, not enough DASD to define a permanent SYSMDUMP file or not having a SYSMDUMP DD defined in the JCL. Considering these conditions and the existing ability of the CA IDMS CV to request an SVC dump when needed, an SVC dump is requested when the CA IDMS CV abends. This capability provides the following benefits:

- A dump is available approximately 100% of the time in an abend situation.
- If desired, any existing dedicated SYSMDUMP files in the CV job stream can be eliminated, and the DASD given back for other use. Alternatively, both dumps can be taken in the event a problem occurs that prevents either dump from completing successfully.
- If all SYSMDUMP, SYSABEND, and SYSUDUMP statements are eliminated from the CV JCL, the time it takes for the operating system to process a CV abend is reduced. This is because the SVC dump process creates an image of the abending CV in another address space and once complete, the CV's address space is allowed to terminate while the address space of the image is written to disk.
- The SVC dump is taken earlier in the abend process than the normal SYSMDUMP or SYSUDUMP. In some cases, this can provide better diagnostic data.

The following considerations apply:

- An SVC dump is directed to the system dump data sets (SYS1.DUMPxx). Therefore, notification should be given to the system programming staff so they can verify that the system dump data sets are sized large enough for the largest CV address space in the system and to ensure that a mechanism is in place to provide for timely offloading and preservation of the dump. If all the system dump data sets are filled with previous dumps, the CA IDMS CV's SVC dump cannot be written.
- This functionality is controlled by the following SYSIDMS parameter:
ABEND_SVC_DUMP=ON|OFF
Controls the system action when an abnormal termination occurs (ABEND). The default (ON) requests an SVC dump prior to the address space terminating.

Note: It is not recommended that this feature be disabled. If the CV address space abends, a dump will be needed in order to determine the cause of the abend.

If your site has any SLIPs set with ACTION=NODUMP, and a CV abends for that condition, no dump is taken as a result. All active SLIPs should be checked for their applicability with respect to the NODUMP action.
For example, if IEASLP00 contains the following SLIP command, and the CV goes into a loop and is cancelled by the operator, no dump is taken unless the operator issues cancel with the DUMP option:

SLIP SET,C=222,ID=X222,A=NODUMP,END

If the CV job stream contains any of the abend DD statements (SYSMDUMP, SYSUDUMP, or SYSABEND), this code does not disable the dump that is generated for the abend. As a result, you receive both an SVC dump and the dump indicated by the abend DD for the same abend.

If DASD space and cycle time are not critical considerations, it is recommended to allow both a SYSMDUMP and an SVC dump to be taken in order to maximize the probability of obtaining a useful dump for Technical Support.

Input Mode Processing

Contents

- IBM Health Checker Exploitation (see page 453)
  - IDMS_SCRATCH_IN_MEMORY Health Check (see page 453)
  - IDMS_CPU_EFFECTIVENESS Health Check (see page 454)
  - IDMS_CHANGE_TRACKING Health Check (see page 455)
  - IDMS_zIIP_USAGE Health Check (see page 456)
  - Modifying CA IDMS Health Check Actions (see page 457)
- CA OPS/MVS Integration for z/OS (see page 458)
  - State Reporting (see page 458)
  - Heartbeats (see page 459)
  - Message Handling (see page 459)
  - Implementation CA OPS/MVS API rules (see page 460)

When coded in JCL, the LABEL=(,,IN) option is honored for database files using the EXCP access method. The file's open mode is forced to input. Any write attempts to the file will fail. If the file is subsequently deallocated, the Label status will not be preserved. If the file is dynamically reallocated, it can be opened in update mode if a write is issued against the file.
IBM Health Checker Exploitation

This section describes the health checks that report on CA IDMS systems. The product owner for all CA IDMS health checks is CA IDMS.

Each health check issues its output as messages to the IBM Health Checker message buffer, which you can view using SDSF, the HZSPRINT utility, or a log stream that collects a history of check output. The HCHECKER facility in CA SYSVIEW provides a convenient method for displaying and modifying the status and attributes of a health check.

If a check finds a potential problem, it issues a WTO message, also known as a message exception. The check exception messages are issued both as WTOs and to the message buffer. The WTO version contains only the message text. The exception message in the message buffer includes both the text and the explanation of the potential problem, including the severity. The message buffer exception message also includes information on what to do to fix the potential problems.

CA IDMS writes information to the JES log about the status of its related health checks whenever a CA IDMS central version is started. This information is provided even when the IBM Health Checker facility is not active on the z/OS system.

The full name of each health check for CA IDMS is in the following format:

<check-name>@<system-name>

<check-name> is the basic check name as documented XXXXXXX.

<system-name> is the name of the system as defined by the SYSTEM ID clause on the SYSTEM statement entered during system generation.

You can use IBM Health Checker syntax to modify the default attributes of a check for a single CA IDMS system. Do this using one of the following methods:

- Specify the system ID as part of the check name.
- Modify the defaults for some or all systems by using the special wildcard characters * or ?.

IDMS_SCRATCH_IN_MEMORY Health Check

Description

IDMS_SCRATCH_IN_MEMORY checks whether scratch is in memory. This health check issues a medium-level exception if scratch is not in memory, because the CA IDMS system runs more efficiently when scratch is kept in memory. This static check runs once during startup.

CA IDMS uses a temporary working storage area referred to as a scratch area. In environments with significant storage constraints, the scratch area can be supported on DASD through standard file I/O. Typically, the performance of the CA IDMS system is improved by enabling this scratch area to reside in main storage. Permitting dynamic extensions to provide for the self-tuning of storage use also improves performance.

Best Practice

Specify SCRATCH IN STORAGE IS YES on the CA IDMS SYSTEM statement.
Parameters

None.

Debug Support

No.

Verbose Support

No.

Reference

For detailed information on setting the SCRATCH IN STORAGE parameter, see the CA IDMS Administering section.

Messages

See the CA IDMS Messages section.

IDMS_CPU_EFFECTIVENESS Health Check

Description:

The CPU effectiveness of a CA IDMS system is the ratio of CPU received to the elapsed time that CPU was required. This ratio shows the amount of time that CA IDMS was forced to wait for CPU resources. Low values of CPU effectiveness indicate serious performance degradation and are often observed as significant response time increases by system users. This number should exceed 90 percent for production systems. This means CA IDMS is reliably provided with at least 90 percent of the CPU required for performing assigned tasks. A lower ratio may be adequate for development or test systems. The ratio that raises an exception can be adjusted for some or all CA IDMS systems.

The CPU effectiveness check is run every 15 minutes, and the CPU effectiveness calculation is based on this interval. A longer interval is not recommended, because the variation in effectiveness over a longer time period makes the calculation less meaningful.

The main cause of low CPU effectiveness is insufficient priority assigned to the CA IDMS job or started task. Another possible cause is high paging rates, indicating that the machine does not have enough real storage to support its current workload. A final cause is a startup parameter on the EXEC statement that specifies SWAP=Y. Swapping should never be allowed for a CA IDMS system.

Best Practice

Ensure that the system is running non-swappable by removing any SWAP=Y parameter from the startup JCL. Assuming that paging rates are acceptable, either increase the IBM Work Load Manager (WLM) velocity goals for CA IDMS or reduce the amount of higher priority work being run on the LPAR. If this does not relieve the condition and the IBM Intelligent Resource Director (IRD) is in use, adjust the IRD configuration to reduce the amount of higher priority work running on the machine. Alternatively, create a separate service class for CA IDMS that is CPU-protected.

Parameters
CPURATIO(cpu-ratio)
cpu-ratio specifies the minimum CPU effectiveness that is acceptable for the CA IDMS system. This ratio can be modified to allow some systems to run at a lower effectiveness level without raising an exception.

Debug Support
No.

Verbose Support
No.

Reference
For detailed information on setting EXEC parameters, see the Administrating section.

For information on WLM and IRD, see IBM documentation.

Messages
See the CA IDMS Messages section.

IDMS_CHANGE_TRACKING Health Check

Description:
IDMS_CHANGE_TRACKING checks whether the change tracking feature is in use. Change tracking is recommended because it facilitates recovery in certain cases, and it can be used to monitor and adjust CA IDMS system parameters. A medium-level exception is issued if change tracking is not in effect. This static check runs once during start up.

Best Practice
CA Technologies strongly recommends that you enable Change Tracking. Change Tracking permits changing the database environment of a CV in a fault-tolerant manner. If the CV fails, the runtime database definition is restored from SYSTRK files during restart, ensuring that the files being updated at the time of failure are the ones recovered by warmstart.

Parameters
None.

Debug Support
No.

Verbose Support
No.

Reference
For detailed information on Change Tracking, see the Administering section.

For information on related administrative procedures, see the CA IDMS Database Administering section.

Messages

See the CA IDMS Messages section.

IDMS_zIIP_USAGE Health Check

Description:

IDMS_zIIP_USAGE checks whether zIIP processors are present and in use. If zIIP processors are available and they are not being used by CA IDMS, it issues a medium-level exception. This check is run once every 24 hours.

CA IDMS optionally exploits zIIP processors on the z9 series and above for the z/OS operating system. This feature permits offloading computing cycles to the zIIP processors, which increases overall CPU throughput while decreasing operational costs.

Most CA IDMS system code is eligible to run on a zIIP processor. CA IDMS runtime processing ensures that a non-zIIP processor is selected to run non-eligible routines, such as: user exits, database procedures, SQL-invoked routines, and application programs.

Best Practice

Enable zIIP usage for CA IDMS systems where one or more zIIP processors are available.

Parameters

None.

Debug Support

No.

Verbose Support

No.

Reference

For more information on zIIP processing, zIIP Exploitation (see page 444).

Messages

See the CA IDMS Messages section.
Modifying CA IDMS Health Check Actions

You can modify the default actions for the CA IDMS health checks by changing the policy statements through IBM Health Check facilities. See IBM’s Health Checker for z/OS User’s Guide for detailed information on establishing or modifying policies to control individual Health Checks.

These sample policy statements could be added to a HZSPRMxx member in SYS1.PARMLIB to permanently change the required CPU effectiveness ratio based on the characteristics of particular CA IDMS systems.

Example 1

/* This sample policy matches the default value for all IDMS systems. */
/* This sample policy entirely deactivates CPU Effectiveness checking for test systems. */
/* This sample assumes that the system ID for each test system begins with the letters “TEST”. */
/* This sample policy enables CPU effectiveness reporting for certain test systems, but with a lower CPU ratio than the default. It also reduces the message severity if an exception occurs. */
/* Since this policy is updated after the previous one, it will take effect for TEST systems with system ids that contain the letters QA in columns 7 and 8 of the system id. */

Example 2

This sample console command can be used to temporarily lower the interval at which CPU effectiveness is checked. You might use this on a critical system when you want an alert if CPU effectiveness falls below the desired level even for a short period of time.

F HZSPROC,UPDATE,CHECK(CA_IDMS,IDMS_CPU_EFFECTIVENESS@PROD100), INTERVAL=(00:01)
CA IDMS - 19.0

CA OPS/MVS Integration for z/OS

CA IDMS provides internal communications with the CA OPS/MVS Application Program Interface (API), employing START/UP/STOP/DOWN state notifications, heartbeats, and passes specific messages into the OPSLOG facility. This helps ease the automation of the production copies of CA IDMS.

State Reporting

CA IDMS reports its current state information to the System State Manager (SSM) component of CA OPS/MVS via the OPS/MVS generic event API. This state can be forwarded to other Common Service components such as the MM Status Monitor. CA OPS/MVS generates these messages into the JES log whenever the status of the CA IDMS system changes, as in the following example:

OPSONOTIFY CASTATE API received for <jobname> with STARTING, Version R18.0 and Level BAT nnn a=008E
OP01370H <jobname> X'0000' X'0000' X'0200' NONE 300 OPSLOGSV CASTATE <jobname> appl id:CAIDMS version:R18.0 level:BAT nnn

OPSONOTIFY CASTATE API received for <jobname> with UP, Version R18.0 and Level BAT nnn a=008E
OP01370H <jobname> X'0000' X'0000' X'0200' NONE 300 OPSLOGSV CASTATE <jobname> appl id:CAIDMS version:R18.0 level:BAT nnn

OPSONOTIFY CASTATE API received for <jobname> with STOPPING, Version R18.0 and Level BAT nnn a=008E
OP01370H <jobname> X'0000' X'0000' X'0200' NONE 300 OPSLOGSV CASTATE <jobname> appl id:CAIDMS version:R18.0 level:BAT nnn

OPSONOTIFY CASTATE API received for <jobname> with DOWN, Version R18.0 and Level BAT nnn a=008E
OP01370H <jobname> X'0000' X'0000' X'0200' NONE 300 OPSLOGSV CASTATE <jobname> appl id:CAIDMS version:R18.0 level:BAT nnn

The CA OPS/MVS OPSVIEW facility is able to manage and display System State Manager resources and its states:

SSM Resource Status----------- CA31 -- 0 P S V I E W -------- Row 1 to 5 of 5
Date/Time: 2010/08/03 11:50 Filtered: N View ====> ALL
System: * SSM Mode: ACTIVE Version: 2 Wait ====> 10
States States
Cm Sta Resource Name Current Desired Res Pre Ref Tng Action Message
-- ---- -------------- -------- -------- --- --- --- --- --- ---------------
__ SYSTEM71 DOWN UP A A A N ACTIVE
__ SYSTEM72 UP UP A A A N ACTIVE
__ SYSTEM73 STARTING UP A A A N ACTIVE
__ SYSTEM74 STOPPING UP A A A N ACTIVE
******************************************************************************

23-Dec-2017 458/524
Heartbeats

Heartbeats are used by applications to report that they are alive, their current processing status, and the reason for that status. This status can be forwarded to other Common Service components such as the MM Status Monitor and the Alert Monitor. CA IDMS currently reports a NORMAL status to indicate that the IDMS system is alive every five seconds. CA OPS/MVS generates messages whenever a change in status is detected, as follows:

```
OP01370H <jobname> X'0000' X'0000' X'0200' NONE 300 OPSONOTIFY CAHEARTBT received for <jobname> applid:CAIDMS version:R18.0
OPSONOTIFY CAHEARTBT received for <jobname> applid:CAIDMS version:R18.0 level: BAT nnn status:NORMAL reason:ACTIVE
```

Heartbeat messages only appear in the log when there is a change. For example, when a product is no longer communicating a NORMAL state or no longer communicating at the expected interval. Likewise, when a problem does occur, the error is only reported in the log once. No other messages are received until the heart beats have been resumed.

Message Handling

CA IDMS employs OPS/MVS API calls to pass log, trace, and journal dump requests into OPS/MVS OPSLOG and to the OPS/MVS rules engine. CA IDMS currently pass these messages into OPSLOG:

- DC050001 - Log % full message?
- DC050004 - Log is FULL message?
- DC205003 - Journal message?
- DC050024 - TRC % FULL message?
- DC050027 - TRC is FULL message?

If any of these messages are encountered, an OPS/MVS API event is generated. You can then write your own API rule to take a specific action. See the Appendix Sample OPS/MVS API rule (see page 516) for a working example.

CA IDMS passes the event name to the OPS/MVS rules engine based on following rules:

```
CAIDMSxxxn
```

```
xxx can be
```

LOG for DC050001,DC050004 messages

- JNL for DC205003 messages
- TRC for DC050024,DC050027 messages

```
n can be:
```

A numeric value in range of 0 - 9 which means the LOG/TRC is 0 - 99% full in 10% increments, e.g. CAIDMSLOG8 means an event has been generated that the log is about 80% - 89% full.
A “F” letter which means the log or TRC is full. For example CAIDMSTRCF.

Implementation CA OPS/MVS API rules

CA OPS/MVS Automated Operations Facility (AOF) can take an action in response to various types of system events. CA IDMS exploits API events to enable easy implementation of AOF rules which permit automation of the production copies of CA IDMS.

Notes:

- For information on how to implement CA OPS/MVS rules, refer to the CA OPS/MVS® Event Management and Automation - AOF Rules Using section.
- For information on how to automate the offloading / archiving processes for DC log, DC traces and journals on CA IDMS, see the Appendix Sample OPS/MVS API rule (see page 516).

Sample z/OS JCL

Contents

- Library, Dataset, and File Name References in z/OS JCL (see page 460)
- z/OS Assemble and Link-Edit JCL (see page 465)
- z/OS Link-Edit JCL (see page 466)
- z/OS DC/UCF Startup JCL (see page 466)

Library, Dataset, and File Name References in z/OS JCL

References are made to the following libraries, Datasets, and file names in z/OS JCL or elsewhere in this section.

For more information on libraries and Datasets see the Installing section for z/OS.

- asfData
  Ddname of the ASF Data (IDMSR-AREA2) area
- asfdefn
  Ddname of the ASF Data definition (IDMSR-AREA2) area
- asfdml
  Ddname of the ASF dictionary definition (DDLDML) area
- asflod
  Ddname of the ASF dictionary definition load (DDLDCLOD) area
- dccat
  Ddname of the system dictionary catalog (DDLCAT) area
- **dccatl**: Ddname of the system dictionary catalog load (DDLDIRL) area.
- **dccatx**: Ddname of the system dictionary catalog index (DDLDIRL) area.
- **cdml**: Ddname of the system dictionary definition (DDLDIRL) area.
- **dclod**: Ddname of the system dictionary definition load (DDLDIRL) area.
- **dclog**: Ddname of the system log (DDLDIRL) area.
- **dcmgs**: Ddname of the system message (DDLDIRL) area.
- **dcrun**: Ddname of the system queue (DDLDIRL) area.
- **dcscr**: Ddname of the system scratch (DDLDIRL) area.
- **dcucfsys**: Name assigned at link edit time to the executable module that starts up DC/UCF. This name is normally RHDCOMVS, which is linked in CAGILOAD during the installation of CA IDMS.
- **dictdb**: Ddname of the application dictionary definition area.
- **dirldb**: Ddname of the IDMSDIRL definition (DDLDIRL) area.
- **dirlld**: Ddname of the IDMSDIRL definition load (DDLDIRL) area.
- **dloddb**: Ddname of the application dictionary definition load (DDLDIRL) area.
- **empdemo**: Ddname of the EMP-DEMO-AREA area.
- **empldemo**: Ddname of the EMPLAREA area.
- **idmsloga**: Ddname of the first sequential log file.
- **idms.appldict.ddlclod**: Data set name of the application dictionary definition load (DDLDIRL) area.
idms.appldict.ddldml
Data set name of the application dictionary definition (DDLDML) area

idms.asfdict.asfData
Data set name of the ASF Data (IDMSR-AREA2) area

idms.asfdict.asfdefn
Data set name of the ASF Data definition (IDMSR-AREA) area

idms.asfdict.ddldclod
Data set name of the ASF dictionary definition load (DDLDCLOD) area

idms.asfdict.ddldml
Data set name of the ASF dictionary definition (DDLDML) area

idms.cagjcics
Name of the CA IDMS CICS library used by TPSORT.
This library exists only if you installed the CICS option.

idms.cagjload
Data set name of the load library containing the vanilla CA IDMS programs

idms.cagjmac
Name of the CA IDMS macro library

idms.cagjsrc
Name of the CA IDMS source library

idms.custom.jcllib
Name of the JCL library created during configuration. Some members in this library are tailored to
your site during the configuration phase of the install process. Others are samples that can be
modified for later use during manual configuration.

idms.custom.loadlib
Data set name of the library containing the load modules customized during configuration.

idms.custom.srclib
Data set name of the library containing the source modules customized during configuration.

idms.dba.loadlib
Data set name of the load library containing the DMCL, Database name table, and programs
associated with demo databases.

idms.empdemo.emdemo
Data set name of the EMP-DEMO-AREA area of the Commonweather Database

idms.empdemo.insdemo
Data set name of the INS-DEMO-AREA area of the Commonweather Database

idms.empdemo.orgdemo
Data set name of the ORG-DEMO-AREA area of the Commonweather Database
- **idms.j1jrnl / idms.j4jrnl**  
  Data set name of the first through fourth disk journal file

- **idms.projseg.projdemo**  
  Data set name of the PROJAREA area of the Commonweather (SQL-defined) Database

- **idms.sqldemo.empldemo**  
  Data set name of theEMPLAREA area of the Commonweather (SQL-defined) Database

- **idms.sqldemo.indxdemo**  
  Data set name of theINDXAREA area of the Commonweather (SQL-defined) Database

- **idms.sqldemo.infodemo**  
  Data set name of theINFORAREA area of the Commonweather (SQL-defined) Database

- **idms.startup.loadlib**  
  Data set name of the load library containing the IDMS startup module

- **idms.sysctl**  
  Data set name of the SYSCTL file

- **idms.systrk**  
  Data set name of the SYSTRK file

- **idms.sysdirl.ddldcload**  
  Data set name of the IDMSDIRL definition load (DDLDLCLOAD) area

- **idms.sysdirl.ddldml**  
  Data set name of the IDMSDIRL definition (DDLDML) area

- **idms.sysloc.ddlocscr**  
  Data set name of the local mode system scratch (SYSLOC.DDLOCSCR) area

- **idms.sysmsg.ddldcmsg**  
  Data set name of the system message (DDLCMSMSG) area

- **idms.syssnap**  
  Data set name of the sequential log file

- **idms.syssql.ddlcat**  
  Data set name of the SQL catalog (DDLCCAT) area

- **idms.syssql.ddlcatlod**  
  Data set name of the SQL catalog load (DDLCCATLOD) area

- **idms.syssql.ddlcatx**  
  Data set name of the SQL catalog index (DDLCCATX) area

- **idms.system.ddldccat**  
  Data set name of the system dictionary catalog (DDLCCAT) area

- **idms.system.ddlcatlod**  
  Data set name of the system dictionary catalog load (DDLCCATLOD) area
- **idms.system.ddldcatx**  
  Data set name of the system dictionary catalog index (DDLCATX) area

- **idms.system.ddldclod**  
  Data set name of the system dictionary definition load (DDLDCLOD) area

- **idms.system.ddldclog**  
  Data set name of the system log (DDLDCLOG) area

- **idms.system.ddldcrun**  
  Data set name of the system queue (DDLDCRUN) area

- **idms.system.ddldcscr**  
  Data set name of the system scratch (DDLDCSCR) area

- **idms.system.ddldml**  
  Data set name of the system dictionary definition (DDLDML) area

- **idms.sysuser.ddlsec**  
  Data set name of the system user catalog (SYSUSER.DDLSEC) area

- **indxdemo**  
  Ddname of the INDXAREA area

- **infodemo**  
  Ddname of the INFOAREA area

- **insdemo**  
  Ddname of the INS-DEMO-AREA area

- **j1jrnl / j4jrnl**  
  Ddname of the first through fourth disk journal file

- **language.runtime.lib**  
  Data set name of the runtime support library for COBOL, PL/I, and so on.

- **orgdemo**  
  Ddname of the ORG-DEMO-AREA area

- **projdemo**  
  Ddname of the PROJDEMO area

- **region-size**  
  Size of the region in which the DC/UCF system is built; this amount should include the FREESTG storage reserved by the startup PARM statement or the default.

- **secdd**  
  Ddname of the system user catalog (SYSUSER.DDLSEC) area

- **sqldd**  
  Ddname of the SQL catalog (DDLCAT) area
• **sqlod**
  Ddname of the SQL catalog load (DDLCATLOD) area

• **sqlxdd**
  Ddname of the SQL catalog index (DDLCATX) area

• **sysctl**
  Ddname of the SYSCTL file

• **systrk**
  Ddname used for SYSTRK files (For more information on referencing SYSTRK files, see Change Tracking.)

• **userdb**
  Ddname of the user Database or application dictionary file

• **user.userdb**
  Data set name of the user Database or application dictionary

• **your.application.loadlib**
  Data set name of the load library containing user-written application programs.

• **your.authorized.loadllib**
  Data set name of the authorized DC/UCF/CV Startup load library. This library is needed as the only library in the startup Steplib if any of the following are true:
  - The zIIP exploitation feature is in effect. See zIIP Exploitation for more information.
  - The z/OS parameter AllowUserKeyCSA(NO) is in effect. See Storage Key Considerations for z/OS CSA subpools for information.
  - The CA IDMS SVC is installed with parameter AUTHREQ=YES. See z/OS for more information.

• **your.custom.lib**
  Data set name of the load or object library containing user-written programs, such as exits and built-in functions.
  This may be the same library that contains the load modules customized during configuration.

• **zOS.maclib**
  Data set name of the z/OS macro library

---

**z/OS Assemble and Link-Edit JCL**

Use the following JCL to assemble and link-edit a module in z/OS.

```
//ASMCL   EXEC  HLASMCL
//C.SYSLIB DD DISP=SHR,DSN=zOS.maclib
//C.SYSIN DD DISP=SHR,DSN=idms.cagjmac
//L.SYSLMOD DD DISP=SHR,DSN=idms.custom.srclib(source-member)
//L.CALOAD DD DISP=SHR,DSN=idms.cagjload
//L.CUSTLIB DD DISP=SHR,DSN=your.custom.lib
```
//L.SYSIN DD *
binder-statements
/*

Replace the variables in the JCL:

- **source-member**
  Replace source-member with the name of the member containing the source to be assembled.

- **binder-statements**
  Replace binder-statements with the binder input statements appropriate to the load module being created.

**z/OS Link-Edit JCL**

Use the following JCL to link-edit a module in z/OS:

```plaintext
//LNKUXIT EXEC PGM=HEWL,
  // PARM=(XREF,MAP,LET,LIST,NCAL)
//SYSPRINT DD SYSPRINT DD
//SYSUT1 DD DSN=&&SYSUT1,UNIT=VIO,SPACE=(1700,(600,100))
//SYSLMOD DD DISP=SHR,DSN=idms.custom.loadlib
//CAGJLOAD DD DISP=SHR,DSN=idms.cagjload
//CUSTLIB DD DISP=SHR,DSN=your.custom.lib
//SYSLIN DD *
binder-statements
/*

Replace the variables in the JCL:

- **binder-statements**
  Replace binder-statements with the binder input statements appropriate to the load module being created.

**z/OS DC/UCF Startup JCL**

**z/OS DC/UCF startup**

```plaintext
//STEPLIB DD DSN=your.authorized.loadlib,DISP=SHR
//CDMSLIB DD DSN=idms.dba.loadlib,DISP=SHR
//  DD DSN=idms.custom.loadlib,DISP=SHR
//  DD DSN=idms.cagjload,DISP=SHR
//  DD DSN=your.application.loadlib,DISP=SHR
//j1jrnl DD DSN=idms.j1jrnl,DISP=SHR
//j2jrnl DD DSN=idms.j2jrnl,DISP=SHR
//j3jrnl DD DSN=idms.j3jrnl,DISP=SHR
//j4jrnl DD DSN=idms.j4jrnl,DISP=SHR
//dcld DSN=idms.system.dcl,DISP=SHR
//dclod DSN=idms.system.dclod,DISP=SHR
//dclat DSN=idms.system.dclat,DISP=SHR
//secdd DSN=idms.sysuser.ddlsec,DISP=SHR
//dcmg DSN=idms.sysmsg.dclmsg,DISP=SHR
//dcllog DSN=idms.system.dcllog,DISP=SHR
```
Additional database file assignments, as necessary
SYSOUT DD SYSOUT=A,
//SYSDLOGA DD DSN=idsys.sysloga,DISP=SHR << multitasking only
//SYSDUMP DD SYSOUT=A
//SYSJCL DD DSN=idsys.sysjcl,DISP=SHR << optional if CA IDMS SVC is used
//SYSTRK DD DSN=idsys.sysstrk,DISP=SHR << optional if change tracking is used
teleprocessing network assignments, as necessary
//SYSIDMS DD *
SYSIDMS parameters, as required

A set of startup JCL tailored to your site is created in the STARTUP member of your custom JCL library during the configuration phase of the install process.

z/VSE Considerations

Cross-Address Space Communication ([http://wiki-dev.ca.com/display/IDMS/Cross-Address+Space+Communication](http://wiki-dev.ca.com/display/IDMS/Cross-Address+Space+Communication))

Users can use cross-address space communication. For example, this allows you to run CICS, DC/UCF, and batch jobs each in a separate address space, if desired.

Because this feature uses the shared virtual area (SVA) for communications, you may need to expand the amount of storage allocated to the SVA. The following considerations apply:

- The SVA size is dependent on the number of external request units defined for use in the DC/UCF system.
- Each DC/UCF system has its own SVA storage needs. For example, two DC/UCF systems take up twice the amount of space in the SVA.

For more information, see the following topics:
- Storage Considerations For the Batch External Interface (see page 468)
- Overriding z/VSE File Specifications at Runtime (see page 468)
- Implementing z/VSE Job Accounting Support (see page 469)
- SVA-eligible Nucleus Modules (see page 469)
Storage Considerations For the Batch External Interface

The batch external interface causes all storage allocation and program loading to occur in the program partition GETVIS area. To allocate enough storage, you must include a SIZE parameter in the EXEC statement in the batch program execution JCL.

The SIZE parameter must allow for space beyond the size of the partition. The following conditions apply to specifying the SIZE parameter on the EXEC statement:

- The SIZE parameter must be large enough to allow the program and the batch interface to be loaded.
- The SIZE parameter must be small enough to allow all other modules to be loaded in the remaining space.

For example, if PROG1 executes in a 1000K partition, the program execution JCL might contain:

```
// EXEC PROG1, SIZE=300K
```

The SIZE specification in the above statement allows 300K for PROG1 and the batch interface, leaving 700K for all other modules, as shown below:

<table>
<thead>
<tr>
<th>Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address 0 IDMS</td>
</tr>
<tr>
<td>PROG1 (300K)</td>
</tr>
<tr>
<td>IDMSDBMS</td>
</tr>
<tr>
<td>IDMSDBIO</td>
</tr>
<tr>
<td>IDMSSPF GETVIS area</td>
</tr>
<tr>
<td>DMCL module (700K)</td>
</tr>
<tr>
<td>DMCL buffers</td>
</tr>
<tr>
<td>subschema</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>additional modules</td>
</tr>
</tbody>
</table>

Overriding z/VSE File Specifications at Runtime

You can override z/VSE file specifications for batch jobs that run under the central version or in local mode by using the SYSIDMS parameter file. The SYSIDMS parameter file allows, among other things, z/VSE users to specify information on sequential files used by CA tools and utilities. Specifically, you can specify the following information:
• The file’s file type
• A block size
• A block factor
• A device address
• Whether to use labels for a tape file
• Whether to activate a facility that allows tape files to span multiple volumes
• How a tape file should be positioned when it is opened or closed

⚠️ **Note:** For more information on the SYSIDMS parameter file, see the *Common Facilities Guide*

---

## Implementing z/VSE Job Accounting Support

z/VSE job accounting support is required to collect CPU times for use by DC/UCF timer facilities. Job accounting support is enabled at z/VSE IPL time. The job accounting interface is enabled by using JA=YES specification in the IPL SYS command within the Automatic System Initialization procedure (ASI PROC).

---

## SVA-eligible Nucleus Modules

Most nucleus modules are reentrant and are eligible to be put in the SVA. To get a list of nucleus modules, you can issue a DCMT DISPLAY MEMORY NUCLEUS command. The following nucleus modules are not SVA eligible:

- IDMSNLT
- RHDCCSA
- RHDCNLT
- RHDCNTRY
- RHDCODVS
- RHDCSCRN
Note: RHDCBANR, as distributed, is eligible for the SVA; if you modify the source, it is no longer reentrant. RHDCUXIT is also eligible provided the exit(s) you are using are reentrant.

User mode programs (anything with a #BALI or IDMSBALI linked with it) cannot be in the SVA. However, non-executable load modules can be. These include CA ADS dialogs, ADSA application load modules, maps, subschemas, and IDD tables.

Important! Some utilities modify the set membership options for the subschema tables. Therefore, the subschema to be used by Step IDMSDBL2 within the RELOAD and FASTLOAD utilities and by Step IDMSDBL4 within the RELOAD, FASTLOAD, MAINTAIN INDEX, and MAINTAIN ASF utilities must not be linked with the attribute reentrant in the SVA.

Any other module that might be loaded but which does not issue IDMS or DC calls is eligible, provided it is truly reentrant.

Note: The size of the SVA has a direct effect on the region size available for the private address space. Consult with the operating system systems programmer to assign the size.

Sample z/VSE JCL

Contents
- Library, Dataset, and File Name References in z/VSE JCL (see page 470)
- z/VSE Assemble JCL (see page 476)
- z/VSE Link JCL (see page 476)
- z/VSE DC/UCF Startup JCL (see page 476)

Library, Dataset, and File Name References in z/VSE JCL

The process of installing IDMS includes the following steps:

1. An IDMS library is created or updated with a number of sub-libraries. For ease of access, the IDMS library can be added to standard labels.

2. IDMSLBL. PROC is installed into the IDMS library.

What is the IDMSLBL. PROC
IDMSLBL is a procedure provided during a CA IDMS z/VSE installation. It contains file definitions for the following CA IDMS components which are provided during installation:

- Dictionaries
- Sample databases
- Disk journal files
- SYSIDMS file

The IDMSLBL procedure is tailored during the installation to reflect the file names and definitions in use at your site.

The sample z/VSE JCL provided in this document includes the IDMSLBL procedure. Therefore, individual file definitions for CA IDMS dictionaries, sample databases, disk journal files, and SYSIDMS file are not included in the sample JCL.

References are made to the following libraries, datasets, and file names in z/VSE JCL or elsewhere in this section:

⚠️ **Note:** For more information on libraries and datasets see the *Installing section for z/VSE.*

- **asfData**
  Filename of the ASF Data (IDMSR-AREA2) area

- **asfdefn**
  Filename of the ASF Data definition (IDMSR-AREA2) area

- **asfdml**
  Filename of the ASF dictionary definition (DDLDML) area

- **asflod**
  Filename of the ASF dictionary definition load (DDLDCLOD) area

- **dccat**
  Filename of the system dictionary catalog (DDLCAT) area

- **dccatl**
  Filename of the system dictionary catalog load (DDLDCATLOD) area

- **dccatx**
  Filename of the system dictionary catalog index (DDLCATX) area

- **dcdml**
  Filename of the system dictionary definition (DDLDML) area

- **dclod**
  Filename of the system dictionary definition load (DDLDCLOD) area
- **dclog**
  Filename of the system log (DDLDCLOG) area

- **dcmsg**
  Filename of the system message (DDLDCMSG) area

- **dcrun**
  Filename of the system queue (DDLDCRUN) area

- **dcscr**
  Filename of the system scratch (DDLDCSCR) area

- **dictdb**
  Filename of the application dictionary definition area

- **dirldb**
  Filename of the IDMSDIRL definition (DDLDML) area

- **dirllod**
  Filename of the IDMSDIRL definition load (DDLDCLOD) area

- **dloddb**
  Filename of the application dictionary definition load (DDLDCLOD) area

- **empdemo**
  Filename of the EMP-DEMO-AREA area

- **empldemo**
  Filename of the EMPLAREA area

- **idmsloga**
  Filename of the first sequential log file

- **idms.appldict.ddldclod**
  File-ID of the application dictionary definition load (DDLDCLOD) area

- **idms.appldict.ddldml**
  File-ID of the application dictionary definition (DDLDML) area

- **idms.asfdict.asfData**
  File-ID of the ASF Data (IDMSR-AREA2) area

- **idms.asfdict.asfdefn**
  File-ID of the ASF Data definition (IDMSR-AREA) area

- **idms.asfdict.ddldclod**
  File-ID of the ASF dictionary definition load (DDLDCLOD) area

- **idms.asfdict.ddldml**
  File-ID of the ASF dictionary definition (DDLDML) area
- **idmslib.custlib**
  Name of the library.sublib containing CA IDMS programs customized for your site during the installation process or created during post-installation configuration.

- **idmslib.dbalib**
  Name of the library.sublib containing the DMCL, Database name table, and programs associated with demo databases

- **idms.empdemo.emdemo**
  File-ID of the EMP-DEMO-AREA area of the Commonweather Database

- **idms.empdemo.insdemo**
  File-ID of the INS-DEMO-AREA area of the Commonweather Database

- **idms.empdemo.orgdemo**
  File-ID of the ORG-DEMO-AREA area of the Commonweather Database

- **idmslib.idmsbase**
  Name of the library.sublib containing the installed non-customized CA IDMS modules

- **idmslib.idmsupp**
  Name of the library.sublib containing the CA IDMS module used for Upper-Case-Only Terminal support

- **idms.j1jrnl / idms.j4jrnl**
  File-ID of the first through fourth disk journal file

- **idms.projseg.projdemo**
  File-ID of the PROJAREA area of the Commonweather (SQL-defined) Database

- **idms.sqldemo.empldemo**
  File-ID of the EMPLAREA area of the Commonweather (SQL-defined) Database

- **idms.sqldemo.indxdemo**
  File-ID of the INDXAREA area of the Commonweather (SQL-defined) Database

- **idms.sqldemo.infodemo**
  File-ID of the INFOAREA area of the Commonweather (SQL-defined) Database

- **idms.startup.loadlib**
  File-ID of the load library containing the IDMS startup module

- **idms.sysctl**
  File-ID of the SYSCTL file

- **idms.systrk**
  File-ID of the SYSTRK file

- **idms.sysdirl.ddldclod**
  File-ID of the IDMSDIRL definition load (DDLDCLOD) area
idms.sysdirl.ddldml
File-ID of the IDMSDIRL definition (DDLDML) area

idms.sysloc.ddlocscr
File-ID of the local mode system scratch (SYSLOC.DDLOCSCR) area

idms.sysmsg.ddldcmsg
File-ID of the system message (DDLDCMSG) area

idms.syssnap
File-ID of the sequential log file

idms.syssql.ddlcatalog
File-ID of the SQL catalog (DDLCAT) area

idms.syssql.ddlcatalogld
File-ID of the SQL catalog load (DDLCATLOD) area

idms.syssql.ddlcatalogx
File-ID of the SQL catalog index (DDLCATX) area

idms.system.ddldccatalog
File-ID of the system dictionary catalog (DDLCAT) area

idms.system.ddlcatalogld
File-ID of the system dictionary catalog load (DDLCATLOD) area

idms.system.ddlcatalogx
File-ID of the system dictionary catalog index (DDLCATX) area

idms.system.ddldcload
File-ID of the system dictionary definition load (DDLDCLOD) area

idms.system.ddldclog
File-ID of the system log (DDLDCLOG) area

idms.system.ddldcrun
File-ID of the system queue (DDLDCRUN) area

idms.system.ddldcscr
File-ID of the system scratch (DDLDCSCR) area

idms.system.ddldmml
File-ID of the system dictionary definition (DDLDML) area

idms.sysuser.ddlsystem
File-ID of the system user catalog (SYSUSER.DDLSEC) area

indydemo
Filename of the INDXAREA area
- infodemo
  Filename of the INFOAREA area

- insdemo
  Filename of the INS-DEMO-AREA area

- j1jrnl / j4jrnl
  Filename of the first through fourth disk journal file

- language.runtime.lib
  File-ID of the runtime support library for COBOL, PL/I, and so on.

- orgdemo
  Filename of the ORG-DEMO-AREA area

- projdemo
  Filename of the PROJDEMO area

- region-size
  Size of the region in which the DC/UCF system is built; this amount should include the storage specified on the FREESTG startup PARM.

- secdd
  Filename of the system user catalog (SYSUSER.DDLSEC) area

- sqldd
  Filename of the SQL catalog (DDLCAT) area

- sqllod
  Filename of the SQL catalog load (DDLCATLOD) area

- sqlxdd
  Filename of the SQL catalog index (DDLCATX) area

- sysctl
  Filename of the SYSCTL file

- systrk
  Filename used for SYSTRK files

**Note:** For more information on referencing SYSTRK files, see Change Tracking (see page 432).

- userdb
  Filename of the user Database or application dictionary file

- user.userdb
  File-ID of the user Database or application dictionary
z/VSE Assemble JCL

Use the following JCL to assemble a source module and catalog the resulting object deck:

```
// LIBDEF *,SEARCH=(idmslib.custlib, idmslib.idmsupp, idmslib.idmsbase)
// LIBDEF *,CATALOG=idmslib.custlib
// DLBL IJSYSPH,'WRK1WORK'
// EXTENT SYSPCH,wrkvol,1,0,begin-track,number-of=tracksASSIGN SYSPCH,DISK,VOL=wrkvol,SHR
// OPTION DECK,LIST,NORLD,NOLRND
// EXEC ASMA90,SIZE=(ASMA90,64K), PARM='CPAT(SYSL),EX(LBX(EDECKXIT(ORDER=AE))),FOLD,OP(ESAX)
  Assembler input statements
END
/*
CLOSE SYSCH,PUNCH
// DLBL IJSYSIN,'WRK1WORK'
// EXTENT SYSIPT,wrkvol
ASSIGN SYSIPT,DISK,VOL=wrkvol,SHR
// EXEC LBR,PARM='ACCESS SUBLIB=idmslib.custlib'
CLOSE SYSIPT,READER
/*
```

z/VSE Link JCL

Use the following JCL to link an executable phase using previously assembled object modules:

```
// LIBDEF *,SEARCH=(idmslib.custlib, idmslib.idmsupp, idmslib.idmsbase)
// LIBDEF PHASE,CATALOG=(idmslib.custlib)
// OPTION CATAL
// EXEC LNKEDT,SIZE=128K,PARM='MSHP'
/*
```

z/VSE DC/UCF Startup JCL

Use the following JCL to start your DC/UCF system

```
// OPTION LOG
// LIBDEF *,SEARCH=(idmslib.sublib)
// LIBDEF PHASE,CATALOG=(idmslib.sublib)
// EXEC PROC=IDMSLBLS
*Additional database file assignments, as necessary
*Additional teleprocessing network definitions, as necessary
*Additional journal file assignments, as necessary
// EXEC IDMSDC,SIZE=40K
Input SYSIDMS parameters, as necessary
/*
DC/UCF Test Environment

This section contains the following topics:

- DC/UCF Test Environment Overview (see page 477)
- Programs and the Test Environment (see page 478)
- Test Environment at Runtime (see page 481)
- Example of Test Environment Execution (see page 482)

DC/UCF Test Environment Overview

DC/UCF provides facilities for establishing a test environment in which a mix of test and production application programs execute under one DC/UCF system in a controlled fashion. When you use the test environment, multiple copies of the same program can exist in one DC/UCF system as follows:

- The production copy of a program is the copy defined with a version number of 1. By default, DC/UCF executes version 1 programs for an application.
- Each test copy of a program is assigned a version number other than 1 when the program is added to the system.

The term program refers to various types of definitions, including the following:

- Assembler, COBOL, and PL/I programs
- Database procedures
- Dialogs (CA ADS and CA ADS Batch)
- Edit and code tables
- Maps
- Subschemas
- Access modules

Test version number

When you enable the test facility for your terminal session, you specify a test version number. DC/UCF then uses the test version number to determine which copy of a program to execute at runtime. As shown in the diagram below, DC/UCF executes programs with the test version number whenever possible.

When one application contains several programs that require testing, you give the same version number to all test programs that are to be executed during the same test.
For example, assume that modifications to production program A require you also to execute a slightly modified copy of program B. In this case, you assign the same test version number (for example, 5) to the test copies of both programs A and B as shown in the following diagram:

1. You define a test version number for your terminal session:
   DCUF TEST 5
2. You execute an application:

   Version 5 programs are executed whenever possible

<table>
<thead>
<tr>
<th>Program</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
</tr>
</tbody>
</table>

Programs and the Test Environment

Contents

- General Considerations (see page 479)
- Programs in the Load Areas (see page 479)
- Programs in Load (core-image) Libraries (see page 480)

The DC/UCF test environment allows you to define multiple copies of the same program in one DC/UCF system. The way that each copy of a given program is differentiated from other copies of the program depends on whether the programs are stored in:

- A dictionary load (DDLDCLOD or DDLCATLOD) area
- Load (core-image) libraries

Specific discussions about programs in load areas and programs in load (core-image) libraries are provided on the following pages, after the general considerations presented below.
General Considerations

As a safeguard, it is advisable to enable storage protection for test programs to prevent them from updating storage belonging to other programs. At system generation time, storage protection is enabled by using both the SYSTEM and PROGRAM statements.

Storage protection can also be enabled for a program at runtime by means of the DCMT VARY PROGRAM STORAGE PROTECT ON command (provided that PROTECT is specified in the system generation SYSTEM statement).

Subschema load modules can be generated with different version numbers. However, the source copy of the subschema always has a version number of 1.

Programs in the Load Areas

When storing multiple copies of a program in either the DDLCLOD and DDLCATLOD dictionary load areas, you differentiate each copy of the program from other copies by the unique combination of:

- Version number
- Dictionary name
- Node name

In the following example, each of the following copies of PROGRAMA is identified by a unique combination of version number and dictionary name:

<table>
<thead>
<tr>
<th>Program name</th>
<th>Version number</th>
<th>Dictionary name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAMA</td>
<td>1</td>
<td>TESTDICT</td>
</tr>
<tr>
<td>PROGRAMA</td>
<td>5</td>
<td>TESTDICT</td>
</tr>
<tr>
<td>PROGRAMA</td>
<td>5</td>
<td>QADICT</td>
</tr>
</tbody>
</table>

Defining a Program to the System

You specify the version number, dictionary name, and/or node name for a program when you define the program in the load area. A program is defined to the system in one of the following ways:

- Manually by using the system generation PROGRAM statement
- Automatically by using a CA IDMS compiler (for example, the subschema compiler) or a CA IDMS development tool (for example, ADSC or ADSOBCOM)
- Dynamically by using the DCMT VARY DYNAMIC program command at runtime

More Information

- For more information on defining programs, see the Administering section.
For more information on the DCMT VARY DYNAMIC PROGRAM command, see the CA IDMS System and Task Operator Reference section.

Programs in Load (core-image) Libraries

The number of load (core-image) libraries that can be defined at your site depends on your operating system as follows:

- Under z/VSE, you can concatenate multiple load (core-image) libraries in one LIBDEF statement.
- Under z/OS and z/VM, a DC/UCF system can have multiple load libraries.

⚠️ **Note:** Load libraries must not be allowed to create multiple extents. Multiple extents will result in I/O error messages and messages stating a module cannot be found.

When storing multiple copies of a program in load libraries, you differentiate each copy of the program from other copies by assigning each copy to a different load library. For example, you can store one copy in the CDMSLIB load library and another copy in the V0005 load library.

**Library version numbers**

By default, load (core-image) libraries in the test environment have version numbers. When the test environment searches for a program by version number, it searches for the program in the load (core-image) library that has the current version number. Programs in a given load (core-image) library have the same version number as the library:

- Programs in CDMSLIB are version 1 (production) programs.
- Programs in other load libraries (under z/OS and z/VM) receive the version number given by the 4-character numeric suffix of the load library's ddname/linkname. For example, a program in V0005 has a version number of 5. A program in V0010 has a version number of 10.

**Load lists**

The load list in effect when you use the test environment overrides the way that the test environment selects load libraries based on version numbers. More information on load lists is given later in this section.

**Dynamically loading a program**

A program can be added to a library after system startup and then used for runtime operations by requested that DC/UCF load a new copy of the program. You can use the DCMT VARY PROGRAM NEW COPY command to request a new copy of a defined program.

**Load library status**
You can vary load libraries online or offline by means of the DCMT VARY LOADLIB command. A load library that you’ve varied offline will not be searched. To examine load library status, use the DCMT DISPLAY LOADLIB command.

**CA ICMS systems**

At z/OS sites, one or more test load libraries can be set up to facilitate testing and migration under CA ICMS. Due to overhead considerations, the smallest group of CA ICMS users should be assigned to use the test load library. For example, if few production users currently exist, those users can be assigned to the test load library.

**Test Environment at Runtime**

**Contents**

- Load Lists (see page 481)
- SYSLOAD Load List (see page 481)

You enable the test facility for your current terminal session by using the DCUF TEST command. By using this command, you also specify a test version number (for example, 5). When the test facility is enabled and you execute an application, the following occurs:

1. DC/UCF searches first for application programs defined with the current test version number
2. If DC/UCF cannot find a copy of the program defined with the test version number, DC/UCF searches for version 1 of the program

You disable the test facility by using the DCUF TEST OFF command.

**Load Lists**

The load list for your current terminal session determines the path that DC/UCF follows when searching for programs to be loaded. The load list identifies the load (core-image) libraries and data dictionaries to be searched and specifies the order that DC/UCF is to follow while searching.

DC/UCF always uses the system-supplied SYSLOAD load list. Users can optionally use site-defined load lists instead of SYSLOAD. Load lists are defined by the system generation LOADLIST statement.

At runtime, a user enables an existing load list for the current terminal session by using the DCUF SET LOADLIST command. For an example of how load list definitions influence the test facility, see the SYSLOAD discussion below.

**SYSLOAD Load List**

The system-supplied SYSLOAD load list is shown below. The SYSLOAD load list uses special keywords when identifying load (core-image) libraries and data dictionaries. These keywords identify libraries and dictionaries symbolically. The following are the SYSLOAD keywords:
User-Default -- Instructs DC/UCF to search using a user-specified value. As a user, you can specify a dictionary by using DCUF SET DICTIONARY/DICTNAME. You specify a dictionary and load library version number by using DCUF TEST.

System-Default -- Instructs DC/UCF to search using a system default value.

ADD LOADLIST SYSLOAD
  DICTNAME IS USER-DEFAULT VERSION IS USER-DEFAULT
  DICTNAME IS SYSTEM-DEFAULT VERSION IS USER-DEFAULT
  LOADLIB IS USER-DEFAULT
  DICTNAME IS USER-DEFAULT VERSION IS 1
  DICTNAME IS SYSTEM-DEFAULT VERSION IS 1
  LOADLIB IS SYSTEM-DEFAULT

Program search

When you have enabled the test environment and the SYSLOAD load list is in effect, DC/UCF follows the SYSLOAD definition (shown previously) when searching for programs. In this case, DC/UCF does the following:

1. Searches for a program defined with the current test version number:
   - In the current session default (if any) dictionary load area
   - In the system default dictionary load area
   - In the load library (under z/OS and z/VM) whose suffix is the same as the current test version number (for example, V0005 for test version 5)

2. Searches for a version 1 copy of the program if the test version cannot be found. DC/UCF searches in the following areas:
   - The current session default (if any) dictionary load area
   - The system default dictionary load area
   - The CDMSLIB load (core-image) library

If DC/UCF cannot find either a test version or a production version of the program, DC/UCF returns a not-found condition.

Example of Test Environment Execution

The following example illustrates the establishment of a test environment and runtime operations in that environment. It assumes the default loadlist, SYSLOAD, is used. The example shows how the programs in sample task X are executed in the test environment. The constituent procedures of sample task X are shown below:

System generation statements

Task Code X Invokes Program A

Program A
DC/UCF system generation statements include:

TASK X INVOKES PROGRAM A
PROGRAM A.
PROGRAM B.
PROGRAM MP1 MAP.
PROGRAM C.
PROGRAM SST SUBSCHEMA.

PROGRAM A VERSION 7.
PROGRAM C VERSION 7.
PROGRAM SST VERSION 7 SUBSCHEMA.

Startup JCL

DC/UCF startup JCL or z/VM commands define load libraries for use by the test facility. The test versions of programs A and C reside in CDMS.TESTLIB and their production versions reside in CDMS.PRODLIB. The JCL or commands are shown below, by operating system:

- z/OS

  //CDMSLIB DD DSN=CDMS.PRODLIB,DISP=SHR
  //V0007 DD DSN=CDMS.TESTLIB,DISP=SHR

- z/VSE

  // DLBL IDMSLIB,'IDMS.LIBRARY',2099/365
  // EXTENT ,nnnnn,,ssss,1508
  // LIBDEF PHASE,SEARCH=(IDMSLIB.SUBLIB,USER007.TESTLIB)

- z/VM

  FILEDEF CDMSLIB DISK idmslib LOADLIB a2 (RECFM V LRECL 1024 BLKSIZE 1024)
  FILEDEF V0007 DISK user userlib LOADLIB a2 (RECFM V LRECL 1024 BLKSIZE 1024)

At runtime

At runtime, the following programs are used when task X is executed from terminals that have requested DCUF TEST 7:

<table>
<thead>
<tr>
<th>Program</th>
<th>Version</th>
<th>Loaded from</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
<td>CDMS.TESTLIB (V0007)</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>CDMS.PRODLIB (CDMSLIB)</td>
</tr>
<tr>
<td>MP1</td>
<td>1</td>
<td>DDLCDLOD area</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
<td>CDMS.TESTLIB (V0007)</td>
</tr>
<tr>
<td>SST</td>
<td>7</td>
<td>DDLCDLOD area</td>
</tr>
</tbody>
</table>

The following programs are used when task X is executed from terminals that have not enabled the test environment:
Simulating 3270-Type Terminals

You can use the DC/UCF simulation facility to simulate 3270-type terminal operations at non-3270 terminals or in a batch environment.

The simulator outputs facsimiles of 3270-type screens whenever online application activities and flow of control would cause a screen to be output at a 3270-type terminal. You can use the 3270 simulation facility in online or batch mode as follows:

- The **online simulator** allows you to simulate 3270-type terminal operations at a non-3270 terminal, such as a TTY or a 2741-type terminal.

- The **batch simulator** allows you to simulate 3270-type online operations in batch mode. either single or multiple terminal DC/UCF configurations in a single batch simulator job.

**How the simulator processes input**

The following illustration shows how the simulator processes input lines. You can use the simulation facility to execute, without modifications, any user program designed to run in a normal DC/UCF online environment. The simulator is completely transparent to all modes of DC/UCF terminal I/O:

- **Basic mode**
- **Line mode**
- **Mapping mode**
Online Simulator

When you use the online simulator, the simulator displays at your terminal a facsimile 3270 screen, as described above, whenever application flow of control would cause a screen to be displayed to a 3270 online user.

You enter simulator input in a single line beneath the displayed facsimile screen. To submit your input data to the simulator, press your terminal’s input key (for example, Return).
**Batch Simulator**

You enter all input lines for a given batch simulator session in a single batch card-image data set. You code input in columns 1 through 72 on the data set. The batch simulator ignores columns 73 through 80.

**Batch simulator lines and terminals:** A simulated DC/UCF system can have a single line with multiple terminals or multiple lines with multiple terminals. For more information, see System Configuration along with other batch simulator topics later in this section.

You submit the card-image data set to the batch simulator when you execute the simulator by using JCL or z/VM commands. Output returned by the batch simulator consists of the screen images that normally would have been displayed on a terminal screen. Output is written to an output (SYSOUT) data set. From there, you can route the output to a line printer.

⚠️ **Note:** For more information about batch simulator execution and output, see Additional Batch Simulator Topics (https://docops.ca.com/pages/viewpage.action?pageId=309429266).

**Starting and Ending a Simulator Session**

The way you start and end a simulator session differs depending on the simulation mode you are using.

You start and end an online simulator session at your terminal by using the DCUF SIMULATE command.

You start a batch simulator session when you submit a data set of input lines to the batch simulator by using JCL statements.

A batch simulator session automatically ends when the batch simulator finishes all simulation activities required by the last line in the input data set.

⚠️ **Note:** For more information about JCL for the batch simulator, see Additional Batch Simulator Topics. (https://docops.ca.com/pages/viewpage.action?pageId=309429266)

**Facsimile Screens**

The DC/UCF simulator outputs a facsimile 3270-type screen whenever an online program would display the screen to a user at a 3270 terminal. The screen is either displayed on your screen (online simulation) or written to an output data file (batch simulation).
On the facsimile screen, a question mark (?) marks each field into which you can enter data. An underscore (_) shows the cursor position. For example, the following facsimile screen has 4 literal fields and 4 variable fields. The cursor is positioned at the EMPLOYEE NAME variable field:

EMPLOYEE NAME:?
ID NUMBER....:?
SOC SEC NUMBER:?
DEPARTMENT...:?

## Composing Input Lines

Your input lines simulate all operations that an online user would perform in response to a given screen.

- General Instructions (see page 488)
- Simulating Control Keys (see page 489)
- Changing the Simulator Environment (see page 491)

For example, the following input line simulates how an online user would add information about Mary Smith in response to the facsimile employee information screen shown previously:

MARY SMITH%TAB 1045D%TAB 135-21-8546%TAB ACCOUNTING%ENTER

- %TAB
  Simulates a pressed Tab key.

- %ENTER
  Simulates a pressed Enter key.

## Contents of input

When you use the simulator in online or batch mode, your input consists of the following:

- Input data that you enter in the same order that an online user would enter the data. For example, suppose an application data-entry screen prompts you to enter an employee ID before you enter the employee's name. In this case, you organize your input line so that the ID number is in front of the employee name.

- Keywords that simulate control keys.

- Commands that change the simulator environment.

## How to compose simulator input lines

The following illustration shows how you compose simulator input lines to simulate online 3270 terminal operations. General instructions for composing input lines are provided below, followed by specific information on simulating control keys and changing the simulator environment.

The following illustration shows how to simulate the simplified online application shown on the left using the simulator input line shown on the right. In this sample application, the menu screen is redisplayed each time an online user presses Enter:
General Instructions

An input line can contain any combination of input data and simulator keywords and commands. It can contain multiple data-fields worth of input data. For example, you can use the 4 methods below to compose a simulated input data line for the screen below:

NAME:...
STATE:...
CITY:...

- **Method 1** -- You can code the simulated input line all on one line:
  
  JOHN DOE%TAB 1134 FOREST-GLEN RD%TAB BOSTON MA%ENTER

- **Method 2** -- You can code the simulated input line on two lines using a continuation character. A hyphen (-) is the default continuation character (when at the end of an input line):
  
  JOHN DOE%TAB 1134 FOREST-GLEN RD%TAB -
  BOSTON MA%ENTER

- **Method 3** -- You can code the simulated input line on two lines, wrapping a word (in this example, BOSTON) to a new line:
  
  JOHN DOE%TAB 1134 FOREST-GLEN RD%TAB BOS-
  TON MA%ENTER
Method 4 -- You can code the simulated input line on three lines:

JOHN DOE%TAB 1134 FOREST--
GLEN RD%TAB -
BOSTON MA%ENTER

Coding rules

Observe the following instructions when composing an input line:

- Start each simulator keyword and command with an escape character. The default escape character is the percent sign (%). For example:

  %ENTER

  The simulator interprets the term after the escape character as a simulator keyword.

- Separate simulator keywords and commands by using a blank. For example:

  field1%NL %FM word2 word3

- Continue input from one line to another (batch simulator only) by using the continuation character. To do this, you end the first line with the continuation character and then continue that line on the next line. The default continuation character is the hyphen (-). For example:

  DEMODICT%TAB %TAB -
  EMPMAP%ENTER

  When you use a continuation character, the character must be the last (nonblank) character of the line. Continuation characters entered before the last nonblank character in the line are treated as data. Blanks that follow the continuation character on a given line are ignored.

- Terminate each input line either by a continuation character or by a simulated 3270 ENTER, CLEAR, PA, PFn, or CNCL control key. The simulator automatically executes a simulated ENTER key at the end of the input line if the line does not end with any of the above. For example, the simulator interprets the following sample input as two separate input lines, each ended with an ENTER:

  MAY SMITH%TAB 1845D%TAB
  135-21-8546%TAB ACCOUNTING%ENTER

  In this example, you indicate that the first line is continued to another line by ending the first line with a continuation characters (the hyphen is the default).

Escape and continuation characters

You can change the simulator escape and continuation characters during a simulator session by using the SET command, described in Changing the Simulator Environment.

Simulating Control Keys

During a simulator session, you simulate 3270-type terminal control keys by using the simulator keywords listed below. When you enter any of these keywords, begin the keyword with the current escape character.
The ENTER, CLEAR, PA\text{n}, PF\text{nn}, and CNCL keywords terminate an input line. The simulator automatically executes the ENTER command for an input line that neither ends with any of these commands nor ends with a continuation character.

### 3270 simulator keywords

These are the 3270-type control keys and events which can be simulated with a keyword. Begin each keyword with the current escape character. In the information below, \% is the escape character, and acceptable abbreviations are shown in parentheses.

- **Control key simulation**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Simulated key</th>
</tr>
</thead>
<tbody>
<tr>
<td>%CLEAR (%C)</td>
<td>Clear</td>
</tr>
<tr>
<td>%CNCL (%CN)</td>
<td>Cancel</td>
</tr>
<tr>
<td>%DUP (%D)</td>
<td>Dup</td>
</tr>
<tr>
<td>%ENTER (%E)</td>
<td>Enter</td>
</tr>
<tr>
<td>%EREOF (%ERE)</td>
<td>EOF</td>
</tr>
<tr>
<td>%ERINP (%ERI)</td>
<td>ERASE INPUT key</td>
</tr>
<tr>
<td>%FM (%F)</td>
<td>FieldMark</td>
</tr>
<tr>
<td>%NL (%N)</td>
<td>New line key</td>
</tr>
<tr>
<td>%PA\text{n}</td>
<td>PA1, PA2, or PA3</td>
</tr>
<tr>
<td>%PF\text{nn}</td>
<td>PF1 through PF24</td>
</tr>
<tr>
<td>%RESET (%R)</td>
<td>RESET</td>
</tr>
</tbody>
</table>

- **Cursor movement simulation**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Simulated key</th>
</tr>
</thead>
<tbody>
<tr>
<td>%BACKTAB (%B)</td>
<td>BackTab</td>
</tr>
<tr>
<td>%MOVECUR (%M) {row, column}</td>
<td>Cursor movement keys; the cursor is set at the specified row and column (enclosed in mandatory parentheses)</td>
</tr>
<tr>
<td>%SKIP</td>
<td>SKIP key; skips the cursor to the next unprotected field</td>
</tr>
<tr>
<td>%TAB (%T) (%)</td>
<td>Tab; identical to skip</td>
</tr>
</tbody>
</table>

- **Light pen simulation**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Simulated event</th>
</tr>
</thead>
<tbody>
<tr>
<td>%SELECT (%SEL) {row, column}</td>
<td>Selects the specified row and column (enclosed in mandatory parentheses)</td>
</tr>
</tbody>
</table>
Changing the Simulator Environment

During a simulator session, you can use the commands listed below to change the simulation environment.

<table>
<thead>
<tr>
<th>Option to change</th>
<th>Command description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuation character</td>
<td>%SET CONTCHAR=continuation-character</td>
</tr>
<tr>
<td>The new continuation character must be a single character. The continuation character indicates that the current input line is continued on the next line. The default is the hyphen (-).</td>
<td></td>
</tr>
<tr>
<td>Escape character</td>
<td>%SET ESCAPE=escape-character</td>
</tr>
<tr>
<td>The new escape character must be a single character. The escape character indicates the start of a simulator keyword. The default is the percent sign (%).</td>
<td></td>
</tr>
<tr>
<td>Message level (online simulator only)</td>
<td>%MSG n</td>
</tr>
<tr>
<td>In this command, n must be an integer in the range 1 through 3. The message level determines how much information the simulator returns after each input operation: 1 -- Output screens and error messages 2 -- Input screens, output screens, and all simulator messages 3 -- Input screens, output screens, all simulator messages, and hexadecimal traces of input and output data streams</td>
<td></td>
</tr>
</tbody>
</table>

Note: For more information about message levels, see the DCUF SIMULATE command in the IDMS System Tasks and Operator Commands Section (https://docops.ca.com/pages/viewpage.action?pageId=309429271).

Additional Batch Simulator Topics

- System Configuration (see page 491)
- Control Commands (see page 493)
- Batch Simulator Output (see page 499)
- Executing the Batch Simulator (see page 502)

System Configuration

You use the DC/UCF system generation compiler to create a DC/UCF system that can execute in batch mode.

Note: For more information about descriptions of system generation procedures, see the CA IDMS Administrating section.
Additionally, z/VSE users must define input and output files for the batch simulator to use. To do this, you must use #DVFILE macros, as discussed in System Startup.

**System Generation Statements**

You use the same system generation statements to generate a batch DC/UCF system as you use to generate an online DC/UCF system. The following special considerations apply to the LINE, PTERM, and LTERM statements:

- The LINE statement defines the batch input/output data sets to be used for simulating 3270-type devices. Specify line type S3270Q for batch 3270 simulation.

- The PTERM statement associates a physical terminal device to be simulated with a batch data set and specifies the characteristics of the device. You can specify either of the following terminal types, allowing simulation of a variety of screen size:
  - Terminal type S3277 (models 1 and 2)
  - Terminal type X3278 (models 1 through 5)

The table below lists the terminal types and modules that you can specify, along with corresponding screen sizes.

- The LTERM statement is used to define a logical terminal and to associate the logical terminal with a batch simulator physical terminal.

### Valid terminal types and modules

<table>
<thead>
<tr>
<th>PTERM statement (TYPE and MODEL)</th>
<th>Screen size (rows x columns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3277 1</td>
<td>12 x 40</td>
</tr>
<tr>
<td>S3277 2</td>
<td>24 x 80</td>
</tr>
<tr>
<td>S3278 1</td>
<td>12 x 80</td>
</tr>
<tr>
<td>S3277 2</td>
<td>24 x 80</td>
</tr>
<tr>
<td>S3277 3</td>
<td>32 x 80</td>
</tr>
<tr>
<td>S3277 4</td>
<td>43 x 80</td>
</tr>
<tr>
<td>S3277 5</td>
<td>27 x 133</td>
</tr>
</tbody>
</table>

### Examples

The following example shows system generation LINE, PTERM, and LTERM statements that allow you to simulate a DC/UCF system with a single line and terminal. The input and output data sets are named SIMIN1 and SIMOUT1, respectively. The simulated screen is 12 rows by 40 columns (that is, type S3277, model 1).

```
LINE line-name TYPE=S3270Q
  INPUT DDNAME=SIMIN1
  OUTPUT DDNAME=SIMOUT1.

PTERM pterm-name TYPE=S3277
```
MODEL=1.

LTERM lterm-name PTERM IS pterm-name.

Configurations for Multiple Lines and Terminals

To simulate several 3270-type devices, use either of the following configurations:

- A single communication line with multiple terminals -- In a single-line/multiterminal configurations, one card-image data set includes simulated input from multiple terminals. When you use the batch simulator, you use the TERMINAL statement to specify the terminal that issues a given input line. More information on the TERMINAL statement is given in Control Commands later in this section.

To create a single-line/multiterminal configuration, code one LINE statement. After the LINE statement, code a PTERM statement and and LTERM statement for each terminal in the configuration. The following example gives a configuration with 1 line and 2 terminals:

```
LINE LINE001 TYPE=S3270Q
  INPUT DDNAME=SIMIN1
  OUTPUT DDNAME=SIMOUT1.
  PTERM TERMX TYPE=S3277 MODEL=1.
  LTERM LTERM X ENABLED PTERM IS TERMX.
  PTERM TERY TYPE=X3278 MODEL=2.
  LTERM LTERMY ENABLED PTERM IS TERY.
```

- Multiple communication lines with multiple terminals -- In a multiline/multiterminal configuration, each simulated communications line is represented by a dedicated card-image data set.

To create a multiline/multiterminal configuration, code 1 LINE statement for each simulated line. After each LINE statement, code one or more PTERM statements, each specifying a unique terminal ID. The following example illustrates a configuration with 2 lines and 2 terminals:

```
LINE LONE TYPE=S3270Q
  INPUT DDNAME=SIMIN1
  OUTPUT DDNAME=SIMOUT1.
  PTERM TONE TYPE=S3277 MODEL=2.
  LTERM LT102 ENABLED PTERM IS TONE.

LINE LTWO TYPE=S3270Q
  INPUT DDNAME=SIMIN2
  OUTPUT DDNAME=SIMOUT2.
  PTERM TTWO TYPE=S3278 MODEL=4.
  LTERM LT104 ENABLED PTERM IS TTWO.
```

Control Commands

To simulate real-time processing, DC/UCF provides the batch simulator control commands described below. Each of these control commands is detailed in alphabetical order after the table.

You include these commands in the input card-image data set that you submit to the batch simulator. With the exception of the MAXTERM function, all of the simulator control commands presented below are optional and are normally followed by a simulated input data line.

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Control command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifying terminal information</td>
<td>MAXTERM -- Use MAXTERM to specify the maximum number of terminals to be simulated in the current batch simulator input data set.</td>
</tr>
</tbody>
</table>
### Purpose

<table>
<thead>
<tr>
<th>Control command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERMINAL --</td>
<td>In a single-line/multiterminal configuration, use TERMINAL to specify the terminal to which the current simulated input data line applies.</td>
</tr>
<tr>
<td>PAUSE --</td>
<td>PAUSE controls the timing of the input relative to the time the input request is issued. PAUSE applies only to the current input data line.</td>
</tr>
<tr>
<td>SET --</td>
<td>SET changes the escape and continuation characters in effect and sets a default pause interval for subsequent terminals on the simulated communications line.</td>
</tr>
<tr>
<td>TIME --</td>
<td>TIME controls the timing of input relative to the beginning of the simulator session. TIME applies only to the current input data line.</td>
</tr>
</tbody>
</table>

### MAXTERM

MAXTERM allows you to specify the number of physical terminals on a simulated communications line. You also can use MAXTERM to suppress the hexadecimal trace printed by the batch simulator.

### Syntax

```
MAXTERM=terminal-count [ NOTRACE ]
```

### Parameter

- **terminal-count**
  
  Specifies the maximum number of terminals on the simulated line. *Terminal-count* must be a positive integer.
  
- **NOTRACE**
  
  Suppresses printing of the hexadecimal trace on batch simulator output.

### Usage

**General coding rules for MAXTERM**

- MAXTERM must be the first statement in the input card-image data set.
- You can code MAXTERM anywhere between columns 1 and 72.
- Embedded blacks are not permitted.

**Number of terminals to specify**

To minimize storage requirements for the batch simulator, MAXTERM should specify the *actual* number of terminals being simulated. For example, if a DC/UCF system is generated with 5 terminals but the simulator run requires only 2 terminals, MAXTERM should specify 2.

### PAUSE

PAUSE allows you to control the timing of input operations for a single input line. You typically use the PAUSE command to override the current default pause interval. Using PAUSE, you can simulate a real-time environment in which timing is critical.
To set a default session or terminal pause interval, you can use the SET command presented later in this section.

Syntax

```
PAUSE= current-escape-character pause-interval
```

Parameter

- **current-escape-character**
  
  Specifies the current one-character escape character.

- **pause-interval**
  
  Specifies the amount of time, in real-time seconds, to elapse between when the input request is issued and the time the application program receives the input data line. *Pause-interval* must be a positive integer.

Example

The data entry screen appears, following:

The batch simulator input lines appear below. Note that the actual data set does not contain intervening blank lines.

```
MAXTERM=3
%SET PAUSE=5 PAUSE2=3
%TERMINAL=1 J. DOE%TAB 123%ENTER
%TERMINAL=2 J. SMITH%TAB 0988%ENTER
%TERMINAL=3 TIME=40 J. BROWN%TAB 4536%ENTER
```

```
Sets the default pause interval to 5 seconds for terminals 1 and 3, and to 3 seconds for terminal 2.
First request from terminal 1 (sent to the program after 5 seconds elapse).
First request from terminal 2 (sent after 3 seconds elapse).
Changes the pause interval to 6 seconds for terminal 1. This request is sent after 6 seconds elapse.
This terminal 1 request is sent after 5 seconds elapse (the pause interval reverts to the terminal's default).
First request from terminal 3. Because of the TIME command, this request is sent to the program 40 seconds after the start of the simulator session.
```

SET

SET allows you to set or change various parameters of the current batch simulation session. For example, you can use SET to change the escape character for the session or to set a default pause interval for a given simulated terminal's input.
Changes established by the SET command apply to all data input on the simulated communication line until another SET command is encountered.

Syntax

```
SET current-escape-character

ESCAPE=escape-character
CONTCHAR=continuation-character
PAUSE=pause-interval
ECHO
NOECHO
```

Parameters

- `current-escape-character`
  Specifies the current one-character escape character.

- `ESCAPE=escape-character`
  Specifies a new escape character.
  The default escape character is the percent sign (%).
  ```
  escape-character
  The new one-character escape character.
  ```

- `CONTCHAR=continuation-character`
  Specifies a new continuation character.
  The default continuation character is the hyphen (-).
  If you include the CONTCHAR parameter in a SET command, use the new continuation character for subsequent lines in the SET command.
  ```
  continuation-character
  The new one-character continuation character.
  ```

- `PAUSE=pause-interval`
  Specifies a new default pause interval. The pause interval is the amount of time, in real-time seconds, that will elapse between the time the input request is issued and the time the application program receives the input data line.
  ```
  pause-interval
  Sets the default pause interval for all terminals on the simulated communication line. The specified pause interval overrides any pause intervals previously established for individual terminals during the simulation session.
  Pause-interval must be a positive integer.
  ```

- `terminal-id=pause-interval`
  Specifies a new default PAUSE interval for the specified terminal.
  Both `terminal-id` and `pause-interval` must be positive integers.

- `ECHO`
  Specifies that input lines are echoed (printed) as they are read. ECHO is the default.
- **NOECHO**
  Specifies that input line are *not* echoed (printed) as they are read. This feature can be used, for example, to prevent the printing of passwords. For example:

```
SIGNON SYSADMIN
ADMIN1
%SET NOECHO
ADPASSX
%SET ECHO
```

**Usage**

**General codes rules for SET**

- The SET command must begin in column 1 of the first card-image record in the input data line.
- SET parameters must be separated from one another by at least 1 blank.
- Embedded blanks are not permitted within a given parameter expression.
- Each parameter expression must be coded on a single card-image line.
- You cannot continue 1 parameter from one line to the next.

**Examples**

The data entry screen appears below:

```
INVOICE NUMBER...:?
PART NUMBER.....:?
PERCENT DISCOUNT:?
```

The batch simulator input lines appear below. Note that the actual data set does not contain intervening blank lines:

```
MAXTERM=3
%SET ESCAPE=@ PAUSE=3 PAUSE2=5

@TERMINAL=3 1234@TAB AB123@TAB 10%@ENTER
@TERMINAL=2 15432@TAB ER321@TAB 12%@ENTER
@SET ESCAPE=%
%TERMINAL=1 98765%TAB DC543%TAB
%20%ENTER
```

Sets the @ sign as the escape character and specifies default pause intervals of 3 seconds for terminals 1 and 3 and 5 seconds for terminal 2.

Specifies input. The percent sign is entered as a data value.

Changes the default escape character to the % sign, but does not change the pause interval.

Specifies input. The percent sign is used as the escape character and also entered as a data value.
TERMINAL

TERMINAL allows you to specify the physical terminal from which the current input data line is to be transmitted. If the TERMINAL command is omitted, the input data line defaults to the terminal last specified or, if no terminal was specified, to the first terminal in the configuration.

Syntax

```
►► current-escape-character TERMINAL=terminal-number ───────────────────────►◄
```

Parameters

- `current-escape-character`
  Specifies the current one-character escape character.

- `terminal-number`
  Specifies the terminal to which the input data line applies.
  *Terminal-number* must be a positive integer in the range 1 through the terminal count specified in the MAXTERM command submitted with the input data set. If *terminal-number* exceeds the value specified in the MAXTERM command, the associated input data line will be ignored.

Usage

**General coding rules for TERMINAL**

- The TERMINAL command must be coded on the first card image of the simulated input data line to which it applies.
- Embedded blanks are not permitted.

Examples

The data entry screen appears below:

```
NAME....:?
CITY....:?
```

The batch simulator input lines appear below. Note that the actual data set does not contain intervening blank lines:

```
MAXTERM=4
%TERMINAL=3 JOHN DOE%TAB BOSTON MA%ENTER
%TERMINAL=2 JAN WHITE%TAB NEWTON MA%ENTER
%J.D. STONE%TAB HUDSON NY%ENTER
%TERMINAL=6 B. JONES%TAB RYE NH%ENTER
```

 Specifies the maximum number of terminals on the simulated line.

 Input data from terminal 3.

 Input data from terminal 2.

 Input data from terminal 2.

 This input line will be ignored because the MAXTERM statement allows only for terminals 1 through 4.
TIME

TIME allows you to delay processing for a specified number of seconds relative to the start of the simulation session. The input data line is not considered received by the application program until the specified number of seconds has elapsed.

Syntax

```
current-escape-character TIME=time-interval current-escape-character
```

- **current-escape-character**
  Specifies the current one-character escape character.

- **time-interval**
  Specifies the amount of time, in real-time seconds, that will elapse between the beginning of the simulator session and the time the input data line is received by the program. Time-interval must be a positive integer.

Usage

**General coding rules for TIME**

- The TIME command must be coded on the first card image of the simulated input data line to which it applies.

- Embedded blanks are not permitted.

Batch Simulator Output

The output generated by the batch simulator consists primarily of screen images.

Representation of 3270 Terminal Attributes

The batch simulator represents 3270-type field characteristics (attributes) by printing special overprinted characters for each attribute. The table below lists the overprinted characters used by the batch simulator. These characters also are listed on the first page of each batch simulator output file.
Information Provided for Screen Images

Each screen image that the batch simulator prints contains the following information:

- Information preceding the screen image:
  - The time, in real-time seconds, relative to the start of the simulator session
  - The terminal to which the simulated data entry operation applies
  - The I/O operation performed by the application program, preceded by three dollar signs; for example:
    $$$ ERASE/WRITE OPERATION
  - The simulated input data line, for read requests only, printed exactly as it appears on the card-image record
  - Simulator messages, preceded by three dollar signs, that indicate the status of the terminal before the I/O request and that list any errors that may have been encountered in the input data set; for example:
    $$$ MODIFIED DATA TAGS HAVE BEEN RESET
  - A hexadecimal trace consisting of the 3270 device-control characters and output data that follow a write request
  - The screen image enclosed in asterisks

- Information following the screen image:
  - Simulator messages, preceded by three dollar signs, that indicate the status of the terminal before and after an I/O request and that list any errors that may have been encountered in the input data set; for example:
    $$$ KEYBOARD IS UNLOCKED
A hexadecimal trace consisting of the 3270 device-control characters and input data submitted as operator input.

Examples

Following is a series of batch simulator output examples:

DC prompts the user for a task code

Sample Screens

Simulated entry of task code QPSG

Sample Screens (2)

DC outputs the screen for the program invoked by the task code
Executing the Batch Simulator

You execute the batch simulator by using JCL or z/VM commands. Your JCL (or z/VM commands) starts up the system on which you will simulate 3270-type activities using the batch simulator.

⚠️ Note: For more information about detailed system startup JCL or commands appropriate to your operating system, see System Startup.
z/OS Systems

In the JCL you use to start up the system for simulation, you must identify the input and output files to be used by the simulator. Sample ddname specifications for the input and output files are shown below:

**Batch simulation files (z/OS)**

```
//simout  DD SYSOUT=A,DCB=(RECRM=FBA,LRECL=121,BLKSIZE=1210)
//simin   DD DSN=siminfle,DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>simin</td>
<td>ddname of the simulator input file</td>
</tr>
<tr>
<td>siminfle</td>
<td>data set name of the simulator input file</td>
</tr>
<tr>
<td>simout</td>
<td>ddname of the simulator output file</td>
</tr>
</tbody>
</table>

You also must specify additional input and output line assignments if you plan to use multiline simulation. You submit batch simulator input statements at the end of the JCL statements for the system.

---

z/VSE Systems

Before you start up the system for simulation, you must define the files to the z/VSE system. To do this, you code #DVFILE macros for the files. For more information about how to do this, see System Startup.

In the JCL you use to start up the system for simulation, you must identify the input and output files to be used by the simulator. Sample JCL statement for these files are shown below:

**Batch simulation files (z/VSE)**

```
// DLBL    simin,'idms.simin',,DA
// EXTENT  sys098,nnnnnn
// ASSGN   sys098,DISK,VOL=nnnnnn,SHR
// DLBL    simout,'idms.simout',,DA
// EXTENT  sys099,nnnnnn
// ASSGN   sys099,DISK,VOL=nnnnnn,SHR
```

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idms.simin</td>
<td>file-id of the simulator input file</td>
</tr>
<tr>
<td>idms.simout</td>
<td>file-id of the simulator output file</td>
</tr>
<tr>
<td>nnnnnnn</td>
<td>volume serial number</td>
</tr>
<tr>
<td>simin</td>
<td>filename of the simulator input file</td>
</tr>
<tr>
<td>simout</td>
<td>filename of the simulator output file</td>
</tr>
<tr>
<td>sys098</td>
<td>logical unit assignment of the simulator input file</td>
</tr>
<tr>
<td>sys099</td>
<td>logical unit assignment of the simulator output file</td>
</tr>
</tbody>
</table>
You also must specify additional input and output line assignments if you plan to use multiline simulation. You submit batch simulator input statements at the end of the JCL statements for the system.

**z/VM Systems**

In the z/VM commands you use to start up the system for simulation, you must identify the input and output files to be used by the simulator. Sample commands for these files are shown below:

**Batch simulation files (z/VM)**

FILEDEF simout PRINTER (RECFM FBA LRECL 121 BLKSIZE 121
FILEDEF simin DISK simin input a (RECFM F LRECL 80 BLKSIZE 80

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>simin</td>
<td>ddname of the simulator input file</td>
</tr>
<tr>
<td>simin input a</td>
<td>file ID of the simulator input file</td>
</tr>
<tr>
<td>simout</td>
<td>ddname of the simulator output file</td>
</tr>
</tbody>
</table>

You also must specify additional input and output line assignments if you plan to use multi-line simulation. You submit batch simulator input statements after you enter all commands to start the system.

**Simulator Messages**

The DC/UCF batch and online simulators return status and error messages to you along with screen images. All messages, with the exception of BEEP, are prefixed by three dollar signs ($$$) when they are output by the simulator.

When you use the online simulator, your current message level determines the messages you receive at your terminal. To change your message level, use the %MSG command discussed in Changing the Simulator Environment.

Simulator messages are presented below in alphabetical order:

*****BEEP***

**Reason:** During an output operation (write or erase/write), the write control character (WCC) in the output data stream was set to sound the 3270 audible alarm.

**ENTER KEY ASSUMED AT indicated-position**

**Reason:** During a simulated data entry operation, the batch simulator did not encounter a keyword, such as ENTER, CLEAR, CNCL, PAn, or PFnn, that terminates the input data line.
The ENTER function is assumed immediately after the last nonblank character on the last card of the input data line. As asterisk is printed directly beneath the card-image record of the input data line to indicate the position at which the batch simulator has assumed the ENTER keyword. Note that this message can be generated if an error in the input data line causes premature termination of the input operation.

**ERASE ALL UNPROTECTED OPERATION**

**Reason:** For an output-type request, the 3270 erase-all-unprotected-fields function has been enabled by the issuing program.

**ERASE/WRITE OPERATION**

**Reason:** For an output-type request, the 3270 erase/write function has been enabled by the issuing program.

**EXPECTED CONTINUATION CARD NOT FOUND**

**Reason:** During a simulated data entry operation, the batch simulator has processed the last card in the input data set having a continuation character as the last nonblank character. The expected continuation card is not present in the data set. Therefore, the input data line is terminated.

**FIELD NOT DETECTABLE**

**Reason:** During a simulated data entry operation, a field specified by the SELECT keyword in the simulated input data line does not have the light-pen detectable attribute.

**FIELDS WITH SPECIAL ATTRIBUTES**

**Reason:** For an output-type request, at least one field on the screen has a nondisplay or light-pen detectable attribute, or has the modified data tag bit set. This message is followed by a list of the fields having special attributes, arranged by field row and column number.

**HEXADECIMAL TRACE**

**Reason:** This message indicates that the printed line to follow is a hexadecimal representation of the input or output data stream. For an output-type request, the hexadecimal trace contains the output data stream, including device-control characters, as constructed by the issuing program and received by the batch simulator for simulated display. For an input-type request, the hexadecimal trace contains the input data stream, including device-control characters, as constructed by the batch simulator and transmitted to the program.

**IGNORING %SELECT AT indicated-position**

**Reason:** This message follows another error message and indicates that a previous error in the input data line has caused the SELECT keyword to be ignored. An asterisk is printed directly below the ignored SELECT keyword.

**ILLEGAL BUFFER ADDRESS-X'hhhh'**
Reason: This message is generated if an I/O-type request references a buffer address that is larger than the size of the simulated screen, as defined in the PTERM statement. For output-type requests, this error occurs following a set-buffer-address order in the output data stream. For input-type requests, this error occurs following an erase-unprotected-to-address operation in the position field of a read-buffer-from-position request.

INPUT INHIBITED AT indicated-position

Reason: During a simulated data entry operation, the simulator is unable to interpret the screen position defined by the MOVECUR or SELECT keyword. The row and column values of the screen position must be specified as (row, column) followed by a blank. The row and column values must be one-, two-, or three-digit nonzero numbers.

KEYBOARD IS LOCKED

Reason: For an output-type request, the write control character (WCC) in the output data stream indicates that the keyboard is to be locked (disabled).

KEYBOARD IS LOCKED, RESET ASSUMED

Reason: The keyboard is currently locked; the previous I/O operation did not reset the keyboard. To perform the data entry operation, the simulator must assume a RESET keyword in the input data line. Normally, the RESET keyword must be specified before input can be accepted.

KEYBOARD IS UNLOCKED

Reason: For an output-type request, the write control character (WCC) in the output data stream indicates that the keyboard is to be reset (unlocked).

KEYWORD IGNORED

Reason: This message follows the INVALID SCREEN POSITION AT message during a simulated data entry operation and indicates that the MOVECUR or SELECT keyword will not be processed.

MAXTERM SET TO terminal-count

Reason: This message indicates the number of terminals to be simulated during the current simulation session. Terminal-count is the numeric value specified in the MAXTERM statement on the first card on the input data set.

MAXTERM WAS NOT SPECIFIED ON THE FIRST CARD. CARD WILL BE IGNORED

Reason: During a simulated data entry operation, the batch simulator could not locate the MAXTERM statement on the first card on the input data set. The number of terminals to be simulated will default to the number of physical terminals defined in the system generation program.

MODIFIED DATA TAGS HAVE BEEN RESET

Reason: For an output-type request, the write control character (WCC) in the output data stream requests an operation to reset the modified data tag (MDT).

NO UNPROTECTED FIELDS ON SCREEN
Reason: This message follows the INPUT INHIBITED AT message and indicates that no unprotected fields appear on the screen. The simulated data entry operation is terminated.

NUMERIC LOCK OCCURRED AT indicated-position

Reason: During a simulated data entry operation, the batch simulator has encountered a non-numeric character in a numeric field. This message is a warning. The character will be inserted at the position of the cursor. On a 3270 device with the numeric lock feature, entry of a non-numeric character causes the keyboard to lock.

OPERATION CHECK

Reason: This message indicates that the I/O operation has terminated due to a logical error caused by an invalid buffer address.

OUTPUT DATA STREAM ENDS IN MIDDLE OF A BUFFER ADDRESS

Reason: For an output-type request, the output data stream has ended prematurely following a set-buffer-address order, repeat-to-address order, or erase-unprotected-to-address order. The current output operation is terminated with an operation check.

READ MODIFIED OPERATION (NO OPERATOR OUTPUT)

Reason: This message is generated as a result of an input operation in which the issuing program has requested an input of all modified fields in the 3270 buffer.

READ OPERATOR INPUT OPERATION

Reason: For an input-type operation, the issuing program has requested an input of all screen entries required by the application.

SCREEN IS UNFORMATTED

Reason: During a simulated data entry operation, the simulator has encountered a SELECT keyword when no light-pen detectable fields have been defined.

SET COMMAND

Reason: During a simulated data entry operation, the batch simulator has encountered a SET command in the input data line. The SET command is printed by the batch simulator on the screen image.

SYNTAX ERROR IN SET COMMAND

Reason: During a simulated data entry operation, the batch simulator has encountered an invalid parameter in the SET command. The SET command is printed on the screen image with an asterisk directly above the invalid parameter.

TABBING TO NEXT UNPROTECTED

Reason: During a simulated data entry operation, the batch simulator has encountered a protected field on the screen and must tab to the next unprotected field in an attempt to insert the input data character.
UNDEFINED CHARACTER CODE IN OUTPUT DATA STREAM-X'hh'

**Reason:** For an output-type request, the output data stream contains an invalid 3270 order code, an invalid DUP character, or an invalid FIELD MARK character. The hexadecimal value of the invalid byte is printed as X'hh' and is stored as a null character. On a 3270-type device, this invalid byte can cause unpredictable results.

UNDEFINED DESIGNATOR CHARACTER

**Reason:** During a simulated data entry operation, the batch simulator has encountered a screen field that was specified by the SELECT keyword and does not have the proper designator character. The first character of a light-pen detectable field should be a question mark, greater-than symbol, blank, or null. The SELECT specification is ignored.

UNDEFINED KEYWORD AT indicated-position

**Reason:** During a simulated data entry operation, the batch simulator has been unable to interpret a keyword in the input data line. The keyword may be misspelled or not followed by a blank, or, in the case of MOVECUR and SELECT, the screen position may be undefined.

WRITE OPERATION

**Reason:** This message is generated by the batch simulator to indicate that an output-type request has been issued by the program.

Specifying Runtime Options

A number of runtime options can be specified in the system startup JCL using the PARM parameter of the EXEC statement in z/OS and z/VSE or the OSRUN command in z/VM.

In z/OS only, you can specify the value of the PARM parameter in one of two formats:

- Freeform in which options are specified as keyword/value pairs
- Positional in which each option is specified in a fixed column within the PARM value

While you can use either format in z/OS, the freeform approach is recommended because it is easier both to code and to understand. Support for the positional format is retained primarily for compatibility with prior releases.

In z/VSE and z/VM, only freeform parameters are supported.

Coding Options as Freeform Parameters

**Contents**

- Coding Syntax (see page 509)
- Coding Parameters (see page 509)
When specifying runtime options as freeform parameters, you code the PARM value as shown in the following z/OS example to start the DC/UCF system version 74 as a multitasking system with 3 subtasks. It also specifies to enable RRS and to use DMCL CVDMCL74 to access the database.

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

**Note:** Because each operating system has a limit on the maximum length of a PARM value, it may be necessary to use abbreviations when specifying runtime options in order not to exceed the limit.

**Coding Parameters**

- **AUTOTASKS|AUTO=Y|N**
  
  Specifies whether to execute startup autotasks. 
  Valid values are the following:
  
  - Y specifies to execute startup autotasks
- N specifies not to execute startup autotasks

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

- **CLONING** | **CLON=Y|N**  
(z/OS systems only) Specifies whether to activate system cloning.  
Valid values are the following:
  - Y specifies to activate cloning
  - N specifies not to activate cloning

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

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

  **Note:** If this parameter is not specified, the amount of storage to be freed defaults to 1600K.

- **MT=Y|N**  
(z/OS systems only) Specifies whether the system runs in multitasking mode.  
Valid values are the following:
  - Y specifies to run the system in multitasking mode
  - N specifies to run the system in unitasking mode

- **MTQDEPTH** | **MTQD=multitasking-queue-depth**  
(z/OS systems only) Specifies the multitasking queue depth.  
The optimum value for the MT queue depth is dependent on factors outside the control of DC, such as other work on the CPUs, operating system dispatcher parameters, paging rate, etc. Therefore, it is advised to experiment with the value and watch the results. The value must be in a range of 0 to 255; however, the advised value is in a range of 0 to 9. The default is 2.

  **Note:** Specifying a low value causes more usage of subtasks. A too-low value causes subtasks to wake up and go back to sleep again without doing any work because the queue was already emptied by another subtask. A too-high value disables multitasking, and most if not all work is processed by only one subtask.
- **PROMPT|PRO=NO|N|PAR|SYS|YES|Y**
  Specifies whether and for what information to prompt the operator during startup. Valid values are the following:
  - NO|N specifies not to prompt the operator for any information during startup.
  - PAR specifies to prompt the operator for system generation related options.
  - SYS specifies to prompt the operator for the version of the DC/UCF system to be started.
  - YES|Y specifies to prompt the operator for both system generation related options and the DC/UCF system version number.

  If no PROMPT option is specified then NO is the default.

  **Note:** For information on how operators respond to the startup prompts, see the CA IDMS System Tasks and Operator Reference section.

- **RMAPSIZE|RMAP=region-map-entry-count**
  Specifies the number of entries to allocate in the DC/UCF region map. region-map-entry-count must be a positive integer. If not specified then 30 is the default.
  The default region map entry count should satisfy most sites; however, if a system uses many optional features, (for example, many line drivers), you may have to increase this value. Issue a DCMT DISPLAY MEMORY MAP command to determine if the map displays all the modules you think it should.

- **RRS=Y|N**
  (z/OS systems only) Specifies whether to enable RRS support. Valid values are the following:
  - Y specifies to enable RRS support
  - N specifies not to enable RRS support

- **STEPLIB|STEP=Y|N**
  (z/OS systems only) Specifies from which library to load RHDCCKUR and RHDCTCKR. Valid values are the following:
  - Y specifies to load from the STEPLIB concatenation
  - N specifies to load from the CDMSLIB concatenation

- **SUBPOOL|SP=operating-system-subpool**
  (z/OS systems only) Specifies the operating system subpool to use for GETMAINN requests. See your operating system documentation for information on operating system subpools. The valid values for operating system subpools are from 1 to 127. The default is 1.
• **SUBTASKS|SUBT=subtask-count**  
  (z/OS systems only) Specifies the number of subtasks (TCBs) to use. **subtask-count** must be a positive integer. This parameter is ignored unless either multitasking or RRS is enabled. If not specified and multitasking or RRS is enabled, the number of TCB's defaults to the number of CPU's available to the operating system.

• **SVC=svc-number**  
  (z/OS systems only) Identifies the CA IDMS SVC number to use during system startup and runtime. The value specified overrides any SVC specified in the SYSGEN SYSTEM statement. **svc-number** must be the number of an active CA IDMS SVC. Specifying an SVC number allows CA IDMS load modules to reside in a PDSE without running CA IDMS as an authorized program.

• **SWAP=Y|N**  
  (z/OS systems only) Specifies whether to run the system as swappable. Valid values are:
  - **Y** specifies to run the system as swappable
  - **N** specifies not to run the system as swappable

  ◣ **Note:** To monitor this system from a remote CA IDMS Performance Monitor session, the DC/UCF system **must** be run non-swappable. Remote monitoring is available only on z/OS.

• **SYSTEM|S=dc/ucf-version-number**  
  Identifies the DC/UCF system to be started. **dc/ucf-version-number** must be the version number of the target system. This parameter is optional; however if not specified, then you must enable the operator to be prompted for the system version number.

• **WTOEXIT|WTO=wto-exit-name**  
  Identifies the write-to-operator (WTO) exit to be used by the DC/UCF system. This value overrides any WTO exit module linked with IDMSUXIT. **wto-exit-name** must be the name of a WTO exit module residing in the DC/UCF load (core image) library. For z/OS and z/VM, if this parameter is not specified, link the WTO exit module with the IDMSUXIT module if you want to exploit the WTO exit. For z/VSE, if this parameter is not specified and you want to exploit this exit, link the WTO exit with the IDMSUXIT module, or as a standalone phase with the name WTOEXIT.

• **WTOREXIT|WTOR=wtor-exit-name**  
  Identifies the write-to-operator-reply (WTOR) exit to be used by the DC/UCF system. This value overrides any WTOR exit module linked with IDMSUSIT. **wtor-exit-name** must be the name of a WTOR exit module residing in the DC/UCF load (core image) library. For z/OS and z/VM, if this parameter is not specified, you must link the WTOR exit module with the IDMSUXIT module if you want to exploit the WTOR exit. For z/VSE, if this parameter is not specified and you want to exploit this exit, link the WTOR exit with the IDMSUXIT module, or as a standalone phase with the name WTOREXIT.
- **ZIIP=Y|N**
  (z/OS systems only) Specifies the type of zIIP support to provide in z/OS.
  Valid values are the following:
  - Y specifies to use zIIP processors if present.
  - N specifies not to use zIIP processors. This is the default.

### Coding Options as Positional Parameters

The use of positional parameters is supported only in z/OS.

When specifying runtime options as positional parameters, you code the PARM value as follows:

```plaintext
Column
0          1          2          3
12345678901234567890123456789012345678
PARM='S=sys#prompt          tnnsclll###submqdz
```

It is important to code parameters in the columns exactly as shown. If the "S=sys#" parameter is omitted, column numbering is relative to the first character of the PARM value. If the "S=sys#" parameter is specified, column numbering is relative to the first character after the sys# value.

**Parameters**

- **S=**
  Identifies the DC/UCF system to be started. This parameter is optional; however if not specified, you must enable the operator to be prompted for the system version number.

- **sys#**
  Specifies the version of the DC/UCF system to be started. `sys#` is a variable length string, in which leading zeroes are ignored.
  **Note:** The remainder of the PARM string requires parameters start in aspecified column position. Column numbering starts (with column one) after the system number, if there is a system number.

- **prompt**
  Columns 1-6 -- Indicates whether to prompt the operator for sysexgen options and the DC/UCF version number.
  Valid values are the following:
  - PROMPT -- Directs startup to prompt the operator.
  - (blank) -- Directs startup to not to prompt the operator.

- **a**
  Column 10 -- Specifies which loadlib to use when loading RHDCCKUR and RHDCTCKR load modules.
  Valid values are the following:
- S -- Loads RHDCCKUR and RHDCCKUR from the STEPLIB concatenation.
- (blank) -- Loads RHDCCKUR and RHDCCKUR from the CDMSLIB concatenation. This is the default.

- b
  Column 11 -- Specifies whether to run startup autotasks.
  Valid values are the following:
  - N -- Startup autotasks are not run.
  - (blank) -- Startup autotasks are run. This is the default.

- tnn
  Columns 21-23 -- Specifies the number of TCB's that CA IDMS uses and controls the use of multitasking and RRS.

- t
  Column 21 -- Controls both multitasking and RRS support.
  Valid values are the following:
  - M -- The system uses multitasking.
  - R -- The system uses unitasking with RRS TCB's.
  - T -- The system uses multitasking with RRS TCB's.
  - U or (blank) -- The system uses unitasking. This is the default.

- nn
  Columns 22-23 -- Specifies the number of TCB's. This is optional; it is only valid if column 21 contained M, R, or T. nn is a left-justified, blank-filled number. If nn is not specified, the number of TCB's defaults to the number of CPU's available to the operating system.

- s
  Column 24 -- Specifies whether to run the DC/UCF system as swappable.
  Valid values are the following:
  - S -- DC/UCF system runs as swappable.
  - (blank) -- DC/UCF system runs as non-swappable. This is the default.

- clll
  Columns 25-28 -- Specifies options for system cloning.

  - c
    Column 25 -- Specifies whether to activate cloning.
    Valid values are the following:
    - C -- Cloning is activated.
    - (blank) -- Cloning is not activated. This is the default.
Columns 26-28 -- Specifies the maximum number of clones. *III* is a left-justified, blank-filled number.

Columns 29-31 -- Specifies a valid CA IDMS SVC number to use during initial system startup. Specifying an SVC number allows CA IDMS load modules to reside in a PDSE without running CA IDMS as an authorized program.

Columns 32-34 -- Specifies the operating system subpool value to use for GETMAIN requests. See your operating system documentation for information on operating system subpools. The valid values for operating system subpools are from 1 to 127. The default is 1.

Columns 35-37 -- Specifies the multitasking queue depth value. The optimum value for the MT queue depth is dependent on factors outside the control of DC, such as other work on the CPUs, operating system dispatcher parameters, paging rate, etc. Therefore, it is advised to experiment with the value and watch the results. The value must be in a range of 0 to 255; however, the advised value is in a range of 0 to 9. The default is 2.

**Note:** Specifying a low value causes more usage of subtasks. A too-low value causes subtasks to wake up and go back to sleep again without doing any work because the queue was already emptied by another subtask. A too-high value disables multitasking, and most if not all work is processed by only one subtask.

Columns 38 -- Specifies the type of zIIP support. Valid values are the following:

- **Y** specifies to use zIIP processors if present.
- **N** specifies not to use zIIP processors. This is the default.

**JESLOG**

In the case of an unexpected abend - Snnn, for example - additional output can appear in the JESLOG:

```
05.04.22 JOB41599 +CCSR010E RHDCOS00 S0C4 at 0000886C LMOD RHDCOMVS CSECT RHDCOESA +0
00FB4 SYSTEM72 TESTDCV DCV
05.04.22 JOB41599 +CCSR020I OWNER = CA IDMS VERSION 18.0.0
05.04.22 JOB41599 +CCSR021I MODULE = RHDCOESA FMID = ~ZAPFMID RMID = ~ZAPFMI
05.04.22 JOB41599 +CCSR061I PSW: 00000000 00000000 078D0F00 8000886C
05.04.22 JOB41599 +CCSR062I ILC: 02 INTERRUPT CODE: 0D REASON CODE: 00000000
05.04.22 JOB41599 +CCSR065I HOME = 0412 PRIMARY = 0412 SECONDARY = 0412
05.04.22 JOB41599 +CCSR070I GR0 - GR1 00000000 00000000 00000000 00000000 02D500 00000000 03D5C50
05.04.22 JOB41599 +CCSR070I GR4 - GR5 00000000 00000000 00000000 00000000 00400000
05.04.22 JOB41599 +CCSR070I GR8 - GR9 00000000 0020D5A0 00000000 0020F7A0
05.04.22 JOB41599 +CCSR070I GR10 - GR11 00000000 000270B0 00000000 000008E2
05.04.22 JOB41599 +CCSR070I GR12 - GR13 00000000 00002ADB0 00000000 3AD7C408
05.04.22 JOB41599 +CCSR070I GR14 - GR15 00000000 00002ADB0 00000000 00000000
```
Sample OPS/MVS API rule

Use the following OPS/MVS API rule to automate the offloading/archiving processes for DC log, DC traces, and journals. This rule eliminates the need to include a WTOEXIT in the CA IDMS system, and also improves system performance when using the CA IDMS zIIP feature.

While there are many possible ways to write an API rule, this is one example of how you can implement a rule to complete the required job submissions for sample DC/UCF systems SYSTEM72 and SYSTEM73 on the same LPAR. This example processes all events starting with the string CAIDMS*. All other parsing of the messages for JOURNAL, LOG, and TRC is done by the rule itself.

```plaintext
)API (CAIDMS*)
)PROC
/**********************************************************************/
/* Proprietary and Confidential Information */
/* Copyright (C) 2010 CA */
/* All Rights Reserved. */
/* Name - APIIDMS */
/* Purpose - Respond to various CA IDMS generated API events. */
/* Related - None */
/* Globals - None */
/* Notes - */
/* This sample OPS/MVS request rule demonstrates */
/* using API rules for CA IDMS sample systems SYSTEM72 */
/* and SYSTEM73. */
/* The CA OPS/MVS generic event Application Program */
/* Interface (API) enables CA software products to */
/* directly generate CA OPS/MVS events. This example */
/* will process on API events initiated by the */
/* CA IDMS product. */
/* This API example will process and perform */
/* the following actions: */
/* Offload_DC_LOG - DC log is nn% full (DC050001) */
/* or DC log is full (DC050004) */
/* messages were received. */
/* The action will be taken to submit */
/* the DC log offload job for that */
/* particular system. */
/* Offload_DC_TRC - TRC is nn% full (DC050024) */
/* or TRC is full (DC050027) */
/* messages were received. */
/* The action will be taken to submit */
/* the traces offload job for that */
/* particular system. */
/* Archive_Journal - Disk Journal is full (DC205003) */
/* message was received. */
/* The action will be taken to submit */
```

---

Sample OPS/MVS API rule

Use the following OPS/MVS API rule to automate the offloading/archiving processes for DC log, DC traces, and journals. This rule eliminates the need to include a WTOEXIT in the CA IDMS system, and also improves system performance when using the CA IDMS zIIP feature.

While there are many possible ways to write an API rule, this is one example of how you can implement a rule to complete the required job submissions for sample DC/UCF systems SYSTEM72 and SYSTEM73 on the same LPAR. This example processes all events starting with the string CAIDMS*. All other parsing of the messages for JOURNAL, LOG, and TRC is done by the rule itself.

```plaintext
)API (CAIDMS*)
)PROC
/**********************************************************************/
/* Proprietary and Confidential Information */
/* Copyright (C) 2010 CA */
/* All Rights Reserved. */
/* Name - APIIDMS */
/* Purpose - Respond to various CA IDMS generated API events. */
/* Related - None */
/* Globals - None */
/* Notes - */
/* This sample OPS/MVS request rule demonstrates */
/* using API rules for CA IDMS sample systems SYSTEM72 */
/* and SYSTEM73. */
/* The CA OPS/MVS generic event Application Program */
/* Interface (API) enables CA software products to */
/* directly generate CA OPS/MVS events. This example */
/* will process on API events initiated by the */
/* CA IDMS product. */
/* This API example will process and perform */
/* the following actions: */
/* Offload_DC_LOG - DC log is nn% full (DC050001) */
/* or DC log is full (DC050004) */
/* messages were received. */
/* The action will be taken to submit */
/* the DC log offload job for that */
/* particular system. */
/* Offload_DC_TRC - TRC is nn% full (DC050024) */
/* or TRC is full (DC050027) */
/* messages were received. */
/* The action will be taken to submit */
/* the traces offload job for that */
/* particular system. */
/* Archive_Journal - Disk Journal is full (DC205003) */
/* message was received. */
/* The action will be taken to submit */
```
/* the archive journal job for that */
/* particular system. */
/* */
/******************************************************************************/

/* Obtain needed event data as well as any need system data that */
/* required for processing. */
/* */
msgtxt = overlay(' ',api.text,5) /* remove * from IDMS message */
localmstcons = OPSINFO('LocMstConsNm') /* Local master console */
msgid = word(msgtxt,2) /* Message ID */
system = word(msgtxt,3) /* system making request */
system = right(system,length(system)-1) /* Strip away leading V */

/******************************************************************************/

/* User threshold values for DC LOG and DC Trace offloading */
LOG_threshold = log-offload-threshold
TRC_threshold = trc-offload-threshold

/******************************************************************************/

/* DC/UCF system version numbers at your IDMS site */
SYSTEM72 = 72 SYSTEM73 = 73 /* Add more systems if needed */

/******************************************************************************/

/* Display debugging information */
MSG1 = "Text Received:" api.text
MSG2 = "MSGID:" msgid" Event:" api.id
MSG3 = "System:" system
/* Print Debugging information - uncomment the next line if needed */
/* Call Debug_rule */

/******************************************************************************/

/* Process this particular class type. */
select
/* Process "DC050001 Log % full" message */
when msgid = 'DC050001' then do
  level = word(msgtxt,7) /* Current % of LOG in use */
  level = left(level,length(level)-1) /* Strip away % sign */
  If level >= LOG_threshold then /* DC Log has exceeded threshold */
    Call Offload_DC_LOG
end

/* Process "DC050004 Log is FULL" message */
when msgid = 'DC050004' then Call Offload_DC_LOG

/* Process "DC205003 Disk Journal FULL" message */
when msgid = 'DC205003' then Call Archive_Journal

/* Process "DC050024 TRC % full" message */
when msgid = 'DC050024' then do
  level = word(msgtxt,7) /* Current % of LOG in use */
  level = left(level,length(level)-1) /* Strip away % sign */
  If level >= TRC_threshold then /* DC Log has exceeded threshold */
    Call Offload_DC_TRC
end

/* Process "DC050027 TRC is FULL" message */
when msgid = 'DC050027' then Call Offload_DC_TRC
otherwise nop
end
return
"*/
/* Offload DC LOG subroutine: */
/* This subroutine will submit offload IDMS DC log job for the */
/* Central Version which issued the event. */
/* */
/* Add statements for all desired DC/UCF systems to be processed */
/*---+----1----+----2----+----3----+----4----+----5----+----6----+----7*/
Offload_DC_LOG:
select
/* Submit the job for sample system SYSTEM72 */
when system = SYSTEM72 then do
  MSGI = "Archive LOG job for the CV" system "has been submitted"
  Call Info rule /* Print informational message into JES log */
  address TSO
  "SUBMIT 'your.custom.jcllib(jcl-member)'
end
/* Submit the job for sample system SYSTEM73 */
when system = SYSTEM73 then do
  MSGI = "Archive LOG job for the CV" system "has been submitted"
  Call Info rule /* Print informational message into JES log */
  address TSO
  "SUBMIT 'your.custom.jcllib(jcl-member)'
end
/* Otherwise do nothing */
otherwise nop
end
return
/*--------------------------------------------------------------------*/
/* Archive Journal subroutine: */
/* This subroutine will submit archive journal job for the Central */
/* Version which issued the event. */
/* */
/* Add statements for all desired DC/UCF systems to be processed */
/*---+----1----+----2----+----3----+----4----+----5----+----6----+----7*/
Archive_Journal:
select
/* Submit the job for sample system SYSTEM72 */
when system = SYSTEM72 then do
  MSGI = "Archive journal job for the CV" system "has been submitted"
  Call Info rule /* Print informational message into JES log */
  address TSO
  "SUBMIT 'your.custom.jcllib(jcl-member)'
end
/* Submit the job for sample system SYSTEM73 */
when system = SYSTEM73 then do
  MSGI = "Archive journal job for the CV" system "has been submitted"
  Call Info rule /* Print informational message into JES log */
  address TSO
  "SUBMIT 'your.custom.jcllib(jcl-member)'
end
/* Otherwise do nothing */
otherwise nop
end
return
/*--------------------------------------------------------------------*/
/* Offload DC TRC subroutine: */
/* This subroutine will submit offload IDMS DC TRC job for the */
/* Central Version which issued the event. */
/* */
/* Add statements for all desired DC/UCF systems to be processed */
/*---+----1----+----2----+----3----+----4----+----5----+----6----+----7*/
Offload_DC_TRC:
select
/* Submit the job for sample system SYSTEM72 */
when system = SYSTEM72 then do
  MSGI = "Archive TRC job for the CV" system "has been submitted"
end
Call Info_rule /* Print informational message into JES log */
address TSO
"SUBMIT 'your.custom.jcllib(jcl-member)"
end

/* Submit the job for sample system SYSTEM73 */
when system = SYSTEM73 then do
  MSGI = "Archive TRC job for the CV system "has been submitted"
  Call Info_rule /* Print informational message into JES log */
  address TSO
  "SUBMIT 'your.custom.jcllib(jcl-member)"
end

/* Otherwise do nothing */
otherwise nop
end
return

/*--------------------------------------------------------------------*/
/* Info subroutine: */
/* This subroutine is used to print various informational strings */
/* into the JES log. */
/*-+----1----+----2----+----3----+----4----+----5----+----6----+----7*/
Info_rule:
MLINXT.1 = 'OPS/MVS IDMS INFORMATIONAL MESSAGE:'
MLINXT.2 = COPIES('**',70)
MLINXT.3 = '***'||CENTER(MSGI,66)||'***'
MLINXT.4 = COPIES('**',70)
address WTO
"Msgid(OPSNOTIFY) Textvar(MLINXT.) Cnname("localmstcons")"
MSGI = ""
return

/*--------------------------------------------------------------------*/
/* Debug subroutine: */
/* This subroutine is used to print various debugging strings into */
/* the JES log. */
/*-+----1----+----2----+----3----+----4----+----5----+----6----+----7*/
Debug_rule:
MLWTXT.1 = 'OPS/MVS IDMS DEBUG MESSAGE:'
MLWTXT.2 = COPIES('**',70)
MLWTXT.3 = '***'||CENTER(MSG1,66)||'***'
MLWTXT.4 = '***'||CENTER(MSG2,66)||'***'
MLWTXT.5 = '***'||CENTER(MSG3,66)||'***'
MLWTXT.6 = COPIES('**',70)
address WTO
"Msgid(OPSNOTIFY) Textvar(MLWTXT.) Cnname("localmstcons")"
MSG1 = ""
MSG2 = ""
MSG3 = ""
return

To complete the API rule, replace these variables:

- **log-offload-threshold**
  Replace log-offload-threshold with the DC log offload threshold value in percent.

- **trc-offload-threshold**
  Replace trc-offload-threshold with the DC traces offload threshold value in percent.

- **your.custom.jcllib**
  The data set name of the card image (FB/80/????) JCL library containing the DC log and DC traces offload jobs and archive journal jobs.
Batch Application Structure

An CA ADS Batch application structure can be defined using the application compiler. You define the application structure in terms of functions, responses, task codes, and global records.

- **Disallowled Functions (see page 520)**
- **Application Flow of Control (see page 521)**
- **Accessing Input Files with Multiple Record Layouts (see page 522)**

Batch application structures differ from online application structures in the following ways:

- Certain functions, such as menu functions, are disallowed.

- Flow of control is slightly different.

- Application structures have a special use in accessing multiple record layouts for input files.

These differences are discussed separately below.

Disallowled Functions

The following types of functions are disallowed in a batch application structure:

- Menu functions

- Menu-related system functions, including POP, POPTOP, HELP, FORWARD, and BACKWARD

- Signon system functions, including SIGNON and SIGNOFF

- The ESCAPE system function

Specifying the Environment

The application compiler allows you to specify the environment in which an application can be executed, as follows:

- **Batch-only** applications can be executed only in the batch environment. The application compiler prevents you from defining disallowed functions.

- **Online-only** applications can be executed only in the online environment.
Application Flow of Control

Flow of control in applications defined using the application compiler is, for the most part, similar for both batch and online applications.

The following special considerations apply to CA ADS Batch applications.

Selection of Responses

Application responses are selected on the basis of batch control events and input record response field values. In the online environment, responses are selected on the basis of a control key pressed or a response field value entered by the terminal operator.

As in the online environment, batch application responses invoke application functions; when a response is selected, so is the function it invokes.

Immediately Executable Functions

Application functions are, by default, immediately executable. Using the application compiler, you can specify, whether a function is immediately executable or deferred. The runtime system uses the specification on mapin operations to determine the next dialog response process or application function to be executed. If, on a mapin operation, both a response process and a function are valid selections, transfer of control depends on the specification for the selected function:

- **Immediately executable**—Control passes to the selected function.

  ![Note: An exception is made when the selected function is the same as the current function. In such a case, the response process is executed.]

- **Deferred**—Control passes to the selected response process. To pass control to the deferred function, the selected response process can issue an EXECUTE NEXT FUNCTION command.

In the batch environment, all functions are, by default, immediately executable. You can override the default for a function by using the new Response Definition screen of the application compiler. The specification is made for the response that invokes the function.

Differences Between Batch and Online

This difference between batch and online flow of control stems from the difference in transaction processing. In the online environment, even if the terminal operator requests transfer to another function, data on the current screen may still require processing by the current dialog before control is passed to the next function. Therefore, the current dialog’s response process takes precedence.

In the batch environment, if the current record's response field value selects a different function, it is assumed that that function is required to process the current input record. Thus, by default, control passes immediately to that function.

Immediately executable functions enable applications to access input files that have multiple record layouts, as described later in this section.
Mapin Operations

Mapin operations are performed by the appropriate functions. For input files that have response fields, the runtime system first examines the response field. If the response field keeps control within the current function, the runtime system maps the record into variable storage and executes a response process. If, instead, the response field selects another function, the runtime system delays the mapin and passes control to the selected function. The next time a mapin operation is performed for that file, the runtime system immediately maps in the record.

Delayed Mapin

The major points regarding delayed mapin are as follows:

- Delaying mapin allows control to be passed to the dialog whose map handles the type of record being read.
- The delayed mapin can be performed only if the application is defined using the application compiler.
- The application functions should be defined as immediately executable, as they are by default.
- A dialog receiving control after a delayed mapin must perform a mapin operation to map the record into variable storage.

Delayed mapin enables applications to access input files that have multiple record layouts, as described below.

For more information on flow of control, see Runtime Flow of Control (https://docops.ca.com/display/IDMS19/RuntimeFlow+of+Control).

Accessing Input Files with Multiple Record Layouts

Application structures have a special use in CA ADS Batch in enabling applications to access input files with multiple record layouts.

For example, consider an application that reads a file containing two types of records stored together: TYPE1 records and TYPE2 records. Each type of record has its own record layout. Additionally, the first two bytes of each record identify the record type: T1 for TYPE1 records and T2 for TYPE2 records.

You define a map for each record layout and associate each map with a dialog (DIALOG1 and DIALOG2). At runtime, the runtime system must use the proper dialog for each input record. Since the sequence of record layouts on input is not predictable, the runtime system must know ahead of time which dialog to execute for each possible record layout. This is done by defining an application structure using the application compiler.

The diagram below shows the application structure and provides sample input data. Response T1 invokes function FUNCTION1, which executes dialog DIALOG1. Response T2 invokes function FUNCTION2, which executes dialog DIALOG2. Both responses are valid from both functions.
Application Execution

Application execution is described below:

1. FUNCTION1 executes DIALOG1. DIALOG1 performs a mapin operation.

2. The runtime system examines the response field of the first input record; it is a T1 record. Since T1 invokes the current function, the runtime system maps the record into variable storage and selects and executes a response process. The response process processes the record, then issues a READ TRANSACTION command, which terminates the current process and performs another mapin operation.
3. The runtime system examines the response field of the second input record; it, too, is a T1 record. The record is mapped in and the response process is executed. The response process issues a READ TRANSACTION command, which terminates the current process and performs another mapin operation.

4. The runtime system examines the third input record; it is a T2 record. Since T2 invokes FUNCTION2, control passes immediately to FUNCTION2, which executes DIALOG2. The record has not yet been mapped into variable storage. DIALOG2 performs a mapin operation.

5. The runtime system immediately maps the third record into variable storage (its response field was already examined) and selects and executes a response process. The response process issues a WRITE TRANSACTION command, which terminates the current process, maps a record to the dialog's output file, then performs a mapin operation.

6. The runtime system examines the response field of the fourth input record; it is a T1 record. The runtime system immediately invokes FUNCTION1, which executes DIALOG1. DIALOG1 performs a mapin operation.

7. The runtime system immediately maps the fourth record into variable storage, then selects and executes a response process. The response process issues a READ TRANSACTION command, which terminates the current process and performs a mapin operation.

8. The runtime system attempts to examine the response field of a fifth input record, but encounters an end-of-file condition.
   Two ways for the application to handle an end-of-file condition are described below:
   - The application can be defined so that the end-of-file condition selects a function whose dialog handles the condition.
   - A response process in DIALOG1 and DIALOG2 can be associated with the end-of-file condition; the response process would be selected and executed when the condition occurred.