

IBM InfoSphere Classic Federation Server for z/OS
Version 11 Release 3

Guide and Reference

IBM

IBM InfoSphere Classic Federation Server for z/OS
Version 11 Release 3

Guide and Reference



Note

Before using this information and the product that it supports, read the information in “Notices” on page 527.

This edition applies to InfoSphere Classic Federation for z/OS (program number 5655-IM4) and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 2003, 2015.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Chapter 1. Overview of product capabilities

With IBM® InfoSphere® Classic Federation Server for z/OS®, you gain many features that help you access and manipulate data across many System z® data sources.

Introduction to Classic federation

Classic federation supports information management initiatives that require accurate, trusted mainframe data integration.

Classic federation delivers the non-relational data on mainframes that you need to stay competitive in a global economy. To keep pace with shifting requirements and accelerating rates of change, an enterprise-scale business must integrate information in diverse proprietary systems on demand. Classic federation delivers up-to-date mainframe data to your data management solutions.

Classic federation provides the latest information to people, processes, or applications. Continue to use your mainframe data as you always have, while sharing it with the rest of your information infrastructure. Whether your business requires mainframe data in e-Business, business intelligence, or data integration solutions, use Classic federation to deliver and manage mainframe information as you need it, when you need it:

- Derive maximum value from information assets across your organization.
- Ensure that mainframe information assets readily participate in all information-dependent initiatives, regardless of format.
- Increase productivity by reducing dependence on the many proprietary APIs that define the mainframe data environment.
- Provide data currency (latency) at the precise level that your business applications require, whether that latency is real-time, yearly, or anything in between.
- Accelerate time to value and time to market with one consistent approach to integrating mainframe data with the processes, methodologies, and tools that you choose.

Classic federation capabilities and features

Classic federation provides direct mainframe data access to meet your information-driven enterprise needs.

Direct mainframe data access

Use Classic federation to query or update mainframe data by mapping it to logical relational tables in a Classic data server. Classic federation uses SQL-based access, which means that you can use standard applications and interfaces, such as ODBC or JDBC, without moving your data from the source database or file system. Federated queries enable you to join data or update transactions across different source databases or file systems, such as IMS™ and CA-Datcom.

Downstream deployments that can access your data include e-commerce Web sites, enterprise self service portals, business intelligence systems, and data warehousing solutions.

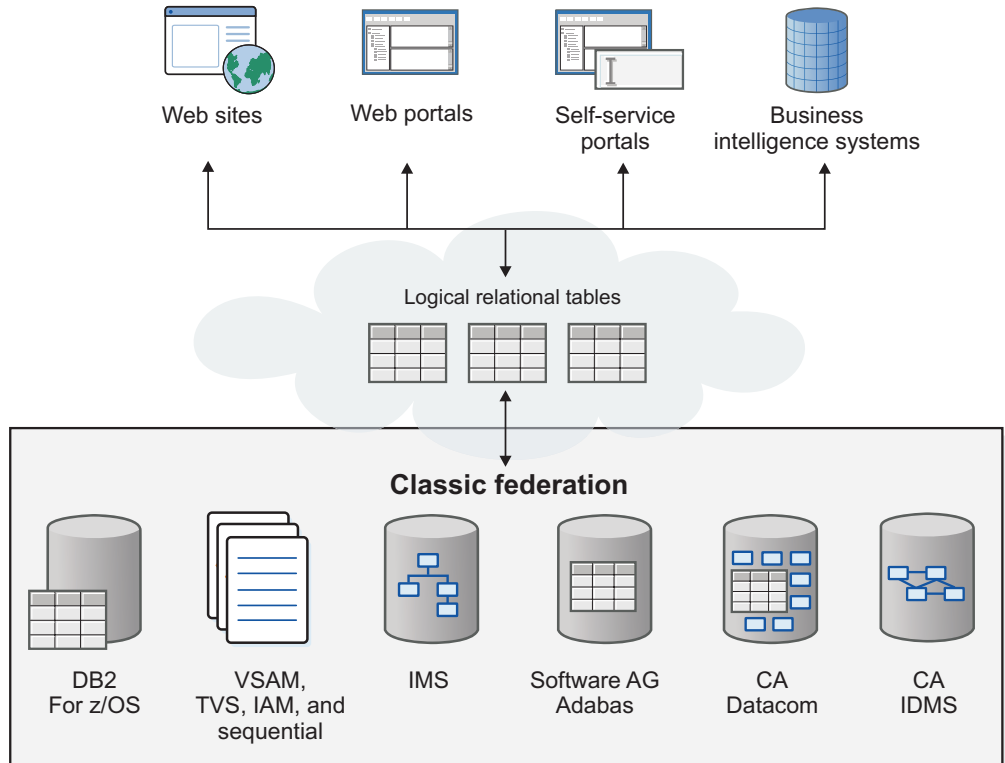


Figure 1. Classic federation with InfoSphere Classic Federation Server for z/OS

Classic federation can query, update, or transfer mainframe data in the following z/OS databases or file systems:

- IMS
- CA-IDMS
- Adabas
- VSAM

Classic federation also supports the following mainframe data sources:

- DB2® for z/OS
- CA-Datacom
- Transactional VSAM (TVS)
- Sequential (SAM)

SQL support

Classic federation provides standards-based relational access to non-relational mainframe data by using applications and tools that use SQL.

By using SQL you can read from and write to mainframe data sources. You can return SQL result sets to federated queries or update the data at the source. Classic federation provides secure and reliable SQL query and update support for reading and writing your data. In addition, Classic federation takes advantage of multi-threading with native drivers for scalable performance.

Query and update data from multiple source systems as if they were one system. Examples include transactional updates across multiple systems, such as

Transactional VSAM and CA-Datcom. The level of update support depends on the capabilities of the source database management system (DBMS).

Metadata-driven SQL access

Classic federation makes use of metadata mappings in logical tables to make mainframe data appear as an ODBC- or JDBC-compliant relational data source. You set up Classic federation by mapping relational table structures to your existing physical mainframe databases and files.

You can use the graphical user interface (GUI) in the Classic Data Architect tool to prototype your queries. By mirroring the queries that your applications use, you can verify that the queries retrieve the correct data. You can then develop your applications concurrently and refine your definitions to optimize query performance. Your relational tools then have standards-based access with ODBC, JDBC, or call-level interface (CLI) interfaces.

Classic federation and IBM information management products

Classic federation provides IBM information management products with seamless access to your z/OS data sources.

InfoSphere Information Server tools dynamically import the logical table, column, and view definitions on your Classic data server into their own metadata stores or repositories. After the metadata is available, the tools access your z/OS data by using their native abilities to access relational databases. This automated metadata and operational integration ensures reusability while maintaining the independence and integrity of each solution.

InfoSphere DataStage[®] uses Classic federation to extract, transform, and repackage z/OS data as you need it before delivering it to one or more target databases. In addition to the flexible model that supports SQL-based access, Classic federation provides a unique interface for InfoSphere DataStage that optimizes data transformation flow while minimizing the use of z/OS resources. This special interface shifts the reformatting of the z/OS data from the Classic data server that runs on z/OS to the InfoSphere Information Server engine on Linux, UNIX, or Microsoft Windows.

Scenario for mainframe data access

The following business scenario describes a real-world solution that use Classic federation, based on stories about a fictitious company named JK Life & Wealth.

Existing infrastructure

The insurance division of JK Life & Wealth wants to deploy an interactive voice response (IVR) system and self service Web sites.

JK Life & Wealth wants to reduce call volume at the call center, where representatives take calls from agents who deal with clients directly. The call center processes policy management data in an IMS system that consists of 1000 IMS transactions. In addition to the IMS data, the call center works with claims and billing data in VSAM and IBM DB2 systems.

The existing mainframe environment relies on a complex application deployment that represents decades of investment and evolution. JK Life & Wealth wants to leverage that investment and continue to take advantage of the transactional speed,

security, and reliability of the mainframe systems.

Requirements

After the company deploys the interactive voice response system and the self service environment, the mainframe data sources must continue to support the existing call center with minimal disruption. The company wants a staged solution that shows value every 3 - 6 months, and the solution must deliver performance and accuracy that maintains credibility with customers.

The self service Web sites run on IBM WebSphere® Application Server. The agents, providers, and customers who visit the sites need an easy-to-use interface. These users do not have the more specialized skills of the call center representatives, who understand the character-based interfaces that are specific to each mainframe data source.

Traditional solutions

Typical solutions are too complex and costly. Some approaches rely on point-to-point architectures, where each data source has its own client interface. A change to a business rule might require modifications to multiple applications, leading to duplication of effort, inconsistencies, and substantial delays before enforcement.

The figure shows a different approach that relies on mainframe programs to extract the data to a relational database and Web-based tools to call the mainframe transactions directly. Some information management professionals call this approach *tightly coupled integration*:

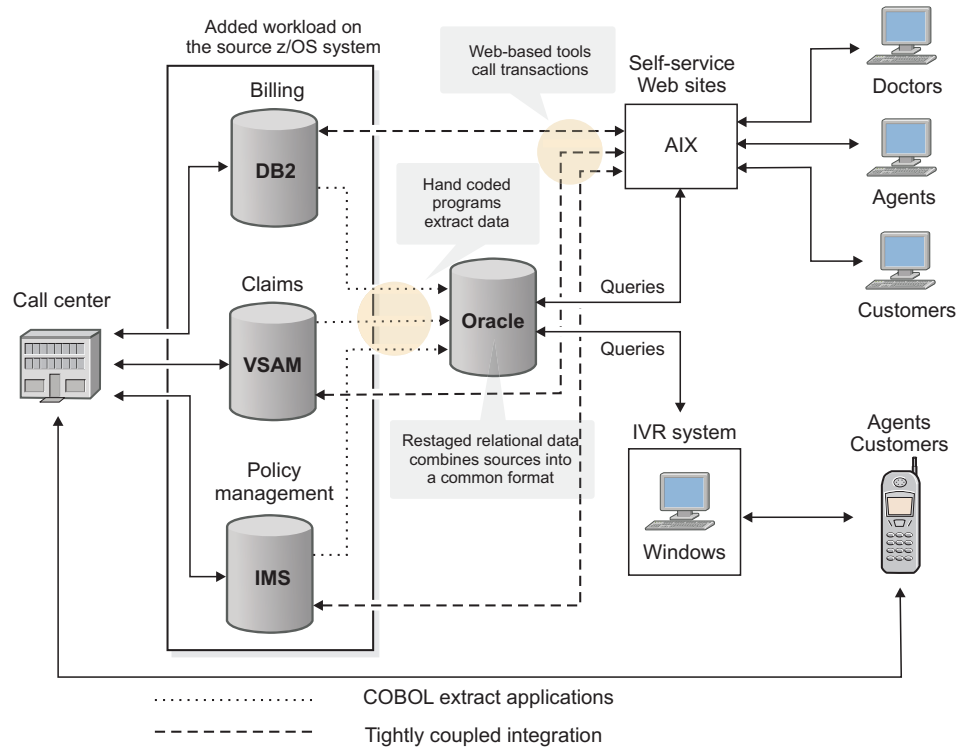


Figure 2. Traditional solution that integrates the data by using mainframe programs and transaction management tools.

This solution transfers the mainframe data to an Oracle database that the interactive voice response system can then query by using standard tools. This approach requires an extract-transform-load (ETL) solution that relies on COBOL extract applications, and has the following disadvantages:

- Because of the large volume of data, you can refresh the Oracle database only once every 36 hours.
- The stale data leads to errors and customer dissatisfaction.
- Workload increases on the source mainframe systems.
- The hardware and software for the ETL solution costs millions of dollars.

For the self service project, the company considered calling mainframe transactions directly by using Web-based transaction management tools. This approach requires an enterprise application integration (EAI) project and also has significant limitations:

- Enterprise application integration projects are costly because they rely on expert mainframe skills.
- Tightly coupled integration increases the workload on the source system.
- Repurposing the native transactions requires too much maintenance overhead.

For example, the company might devote 10 person hours per transaction to repurpose the IMS transactions for Web-based integration. With 1000 IMS transactions, the company might invest in excess of ten thousand person hours in a solution that produces 1000 points of integration to maintain. This approach does not reuse the data effectively.

Solution

After evaluating the options, the company chose Classic federation to provide direct access to the mainframe data.

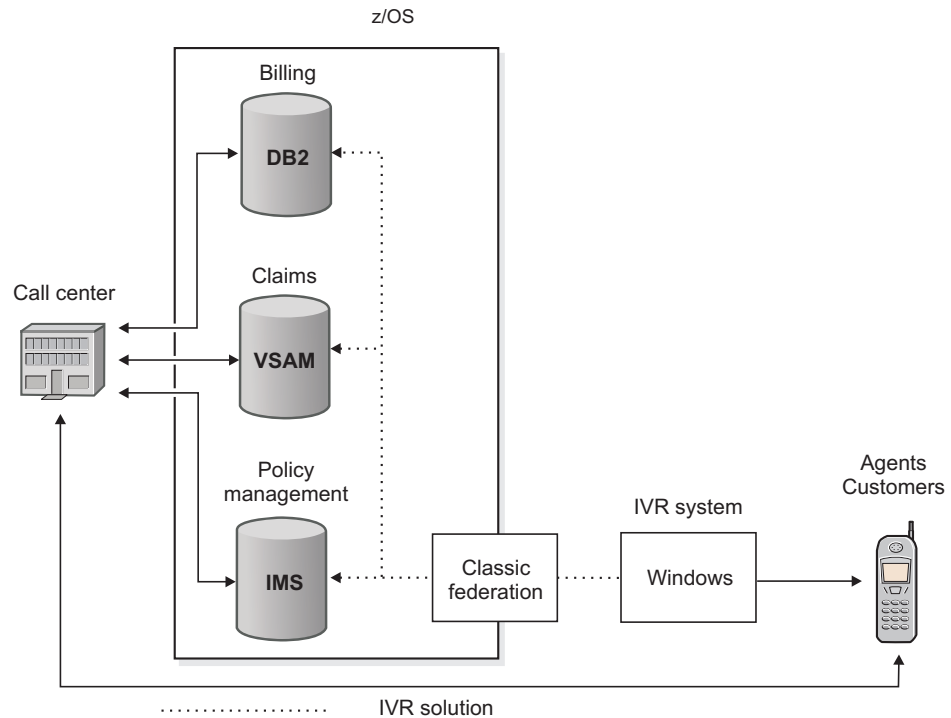


Figure 3. Architecture of the interactive voice response solution.

In the first stage of implementation, Classic federation connects the interactive voice response system directly to the account and claim information in the IBM DB2 and VSAM systems and the policy management data in IMS. Classic federation leverages the inherent SQL capabilities of the tools in the interactive voice response system to make the mainframe data appear to be an ODBC-compliant data source. Customers and agents can now retrieve information about their accounts and the status of their claims from the interactive voice response system, thereby decreasing the volume of more costly inquiries at the call center.

The interactive voice response solution demonstrates value in the first 3 - 6 months. The insurance division decides to move ahead with the second stage and integrate the same data with the self service Web sites.

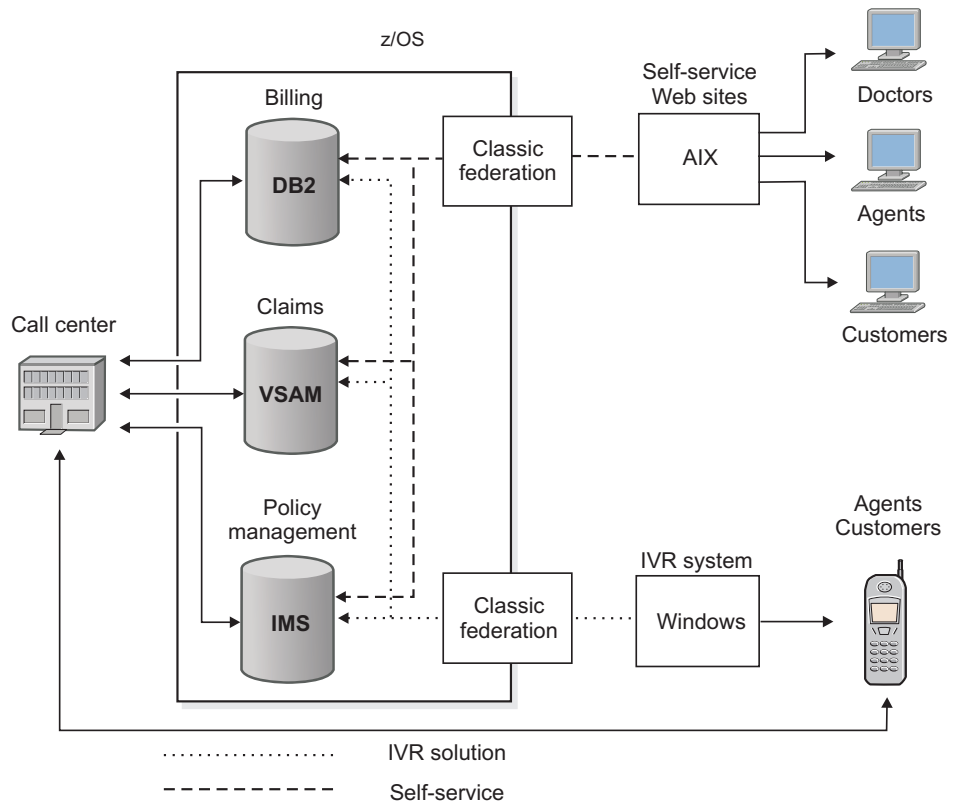


Figure 4. Architecture of the self service solution.

Classic federation now connects IBM WebSphere Application Server directly with IBM DB2, VSAM, and IMS data, with minimal processing overhead on the mainframe. Self service customers can now process billing, policies, and claims accurately, accessing up-to-the-minute data in a single, friendly Web interface. Users access the data transparently, regardless of its location on the mainframe.

In this example, the integration was complete in 200 person hours, compared to the ten thousand person hours that the traditional solutions required. The investment of skilled time and resources was minimal. Classic support for industry-standard interfaces made it simpler to migrate later to a J2EE development environment by switching to the Classic JDBC client.

Release notes for IBM InfoSphere Classic Federation Server for z/OS, version 11.3

The Release Notes® include information about new functionality included in Version 11.3 and changes in existing functionality.

Contents

- "What's New"
- "Migration considerations" on page 8

What's New

The following items are key new features in Version 11.3.

Metadata catalog improvements

- Performance improvements reduce the time needed to access objects in the metadata catalog. The time to needed to run the catalog maintenance utility processes and the run time access to catalog objects is reduced.
- Improved metadata catalog access time eliminates the need to use a linear metadata catalog. The new catalog is faster than linear catalog access and provides full update capability to the catalog.
- Using the zFS file system to store metadata catalog files increases catalog capacity.
 - The zFS file system permits very large numbers of objects to be described. For example, a 4GB metadata catalog can accommodate approximately five million column definitions spread across three thousand table definitions.
 - The maximum supported size of the zFS resident data component of the metadata catalog is approximately 254GB, approximately 127 times larger than the largest supported sequential file system (QSAM/BSAM) resident catalog . At this size the catalog can describe approximately 75 million tables of 300 columns each.

Migration considerations

The following migration considerations apply to version 11.3 of Classic federation.

Upgrading the metadata catalog

An upgrade is required for the metadata catalog. The installation customization process includes the steps required to upgrade the metadata catalog. For more information, see [Upgrading metadata catalogs](#).

Migrating configuration files

An EXPORT/IMPORT operation is required to migrate configuration files. The installation customization process includes the steps required to export the configuration and to import the configuration into a new configuration file that includes all new configuration parameters and their default values. For more information, see [Installing Classic data servers](#).

Release notes for InfoSphere Classic Data Architect, Version 11.3

Updated information for Version 11.3 of InfoSphere Classic Data Architect is provided in release notes.

Migrating the workspace from a previous version of CDA

If you are migrating from a previous version of Classic Data Architect you might have to migrate your workspace the first time you start Classic Data Architect. The new views and perspective changes are not displayed until after the migration. The procedure for migrating from an older version of Classic Data Architect varies depending on the version you are migrating from:

Migrating from CDA Version 11.1

If you are using the Data perspective you need to open it the first time you use version 11.3 of CDA by selecting **Window > Open Perspective > Data**.

Migrating from CDA Version 10.1

- The first time you start the new version of Classic Data Architect, a dialog box will appear asking you to confirm the migration of the workspace. Classic Data Architect will restart when the migration completes.

- If you are using the Data perspective you need to open it the first time you use version 11.3 of CDA by selecting **Window > Open Perspective > Data**.

Migrating from CDA Version 9.5

- Reset the Data perspective by selecting **Window > Reset Perspective**.
- Display the new default perspective by selecting **Window > Open Perspective > Replication**.
- If you are using the Data perspective you need to open it the first time you use version 11.3 of CDA by selecting **Window > Open Perspective > Data**.
- Recreate your data source connections in the Data Source Explorer.

When the workspace is migrated, the new views, the Replication perspective, and the Data perspective are available.

Overview of IBM InfoSphere Classic Federation Server for z/OS

InfoSphere Classic Federation Server for z/OS is a complete, high-powered solution that provides SQL access to mainframe databases and files without mainframe programming.

Using the key product features, you can:

- Read from and write to mainframe data sources using SQL.
- Map logical relational table structures to existing physical mainframe databases and files.
- Use the Classic Data Architect graphical user interface (GUI) to issue standard SQL commands to the logical tables.
- Use standards-based access with ODBC, JDBC, or CLI interfaces.
- Take advantage of multi-threading with native drivers for scalable performance.

The architecture of IBM InfoSphere Classic Federation Server for z/OS consists of the following major components:

- Data server
- Data connectors
- Classic Data Architect
- Metadata catalog
- Clients (ODBC, JDBC, and CLI)
- Stored procedure connectors

Classic data servers

A Classic data server runs in its own address space.

Classic data servers perform the following functions:

- Determine the type of data to access
- Create relational result sets from native database records
- Maintain the environment
- Process SQL queries by performing SQL functions that include:
 - Accepting SQL queries from clients
 - Transforming SQL queries into the native file or database access language

- Optimizing queries
- Processing post-query result sets as needed

A Classic data server accepts connection requests from client applications. Client applications can access a Classic data server using the ODBC, JDBC, or CLI client that InfoSphere™ Classic Federation Server for z/OS provides.

The architecture of a Classic data server is service-based. The Classic data server consists of several components, or *services*.

Services and their functions

When a Classic data server is created during the installation customization process, the services required for the Classic data server are pre-configured.

Configuration definitions include the following server-wide and individual service categories:

- Server-wide, or global, definitions that affect all services within the Classic data server
- Federation-specific service definitions that consist of unique configuration information that affects each service individually.

Each service is a member of a service class. Services also have a service name and a task name.

service class

The type of service, such as the query processor service (QP or QPRR).

service name

The unique name that references a specific instance of a service in a Classic data server, such as additional instances of a query processor service to improve performance.

task name

The load module that is associated with services of a service class, such as CACQP for the query processor service class.

A service class contains a specific set of configuration parameters. The values of the configuration parameters define a service instance and the behavior of that service.

Critical services

A *critical service* is a service that is critical to the operation of the Classic data server. The Classic data server cannot continue running when one or more services that are critical to the operation of the server are stopped or stop abnormally.

The following service is a critical service:

Logger service

The logger service receives messages from all services in the Classic data server and coordinates writing the messages to common logs.

You cannot stop a critical service. If you attempt to stop a critical service by issuing a STOP,SERVICE command or a STOP,TASKID command, a warning message is issued.

Detailed information describes each service that runs in the Classic data server, each service configuration parameter, and configuration methods for administrating the configurations for the Classic data server.

Region controller:

The region controller monitors and controls the other services that run within the data server.

The region controller directly or indirectly activates each service based on the configuration parameters that you define in a the service definition for the region controller service. The region controller starts, stops, and monitors the other tasks that run within the data server.

The region controller also includes a z/OS MTO (Master Terminal Operator) interface that you can use to monitor and control a data server address space.

Connection handlers:

The connection handler listens for connection requests from client applications and routes them to the appropriate query processor task.

The connection handler task can load the following communication protocols:

- TCP/IP
- z/OS cross-memory services

A local client application can connect to a data server using any of these methods. Remote client applications use TCP/IP to communicate with a remote server.

Query processors:

The query processor is the subcomponent of the data server that processes SQL requests. The SQL requests can access a single database or file system or reference multiple types of databases or file systems.

There are two types of query processors:

Single-phase commit query processor (CACQP)

Accesses and joins information from multiple data sources and performs updates to a single data source.

Two-phase commit query processor (CACQPRRS)

Accesses and joins information from multiple data sources and performs updates to multiple data sources. The two-phase commit query processor uses z/OS Resource Recovery Services to coordinate the data source updates. This query processor can participate in distributed transactions by using a JDBC client and a distributed transaction manager (such as WebSphere Application Server).

The two-phase commit query processor supports the CA-Datcom, DB2 for z/OS, IMS, and transactional VSAM data sources.

You cannot mix these two types of query processors within a single data server.

The data server configuration must include a service definition for a query processor.

The query processor handles requests from the Classic JDBC, ODBC, and CLI clients in form of SQL. It invokes one or more connectors to access the target database, file system, or stored procedures that are referenced in a SQL request from the client.

By using native database and file facilities, the query processor maintains the structural integrity and the performance characteristics of the data source.

Initialization services:

Initialization services are special tasks that initialize different types of interfaces to underlying database management systems and z/OS system components.

Initialization services prepare the data server execution environment. For example, an initialization service is provided to activate the DRA interface that the IMS DRA connector uses to access IMS data.

The following table lists the initialization services that provide access to specific data sources.

Table 1. Initialization services for specific data sources.

Data source	Initialization services
CA-Datcom	<p>CA-Datcom Initializes the data server for connections to the CA-Datcom Multi-User Facility (MUF).</p>
DB2 for z/OS	<p>Call Attachment Facility (CAF) Connects to a DB2 for z/OS subsystem to access and update DB2 data using the DB2 Call Attachment Facility.</p>
IMS	<p>IMS DRA Initializes the DRA interface and connects to an IMS DBCTL region to access IMS data using the DRA interface.</p> <p>IMS BMP/DBB Initializes the IMS region controller to access IMS data using the BMP or DBB interface.</p>
VSAM	<p>VSAM Initializes the region controller to access VSAM data.</p>

The following table lists the initialization service that provides access to z/OS system components.

Table 2. Initialization services that access z/OS system components.

z/OS system components	Initialization services
z/OS Workload Manager (WLM)	<p>Workload Manager Initializes the z/OS Workload Manager subsystem using the WLM system exit to enable query processing in WLM goal mode.</p>

The Language Environment® initialization service is deprecated in Version 10.1.

Logger:

The logger service is a task for system monitoring and troubleshooting. The logger reports data server activities and is used for error diagnosis.

Optimizing memory consumption for a Classic data server (guidelines)

To optimize memory consumption, estimate initial memory settings in the job control language (JCL) for your Classic data server and then evaluate them in a test environment.

Procedure

1. Estimate initial values for **REGION** in the JCL for the Classic data server and for the **MESSAGEPOOLSIZE** configuration parameter.

- For smaller environments, try the default values for the Classic data server:
 - **REGION**=96MB
 - **MESSAGEPOOLSIZE**=64MB
- Consider larger values for **REGION** and **MESSAGEPOOLSIZE** for larger deployments that require more resources. For example, you can begin with values that are larger than needed at first. Then you can define your environment and work toward reducing these values by monitoring the environment and running reports such as the output from the **DISPLAY,MEMORY** command.

Consider factors that contribute to resource consumption:

- Fixed overhead per Classic data server outside message pool storage:
 - C runtime library functions (LE)
 - Added threads, such as an additional query processor service
- The number of concurrent client connections
- The number of cached statements
- The size of SQL statements

2. Experiment with different configurations in a test environment to verify that your Classic data server has sufficient resources for the size of your environment.

- a. Specify a region size that is at least 8 MB lower than the site limit, and use the greater of these values:

- 8 MB higher than the message pool
- 20% higher than the message pool

If the 8 MB gap between the region and the message pool is still not sufficient, increase this difference in increments of 8 MB.

- b. Set the **MESSAGEPOOLSIZE** parameter to the greater of these values:

- 20% less than the region size
- 8 MB below the **REGION** value or 8 MB below any site limit imposed by exits.

If you increase the value of the **MESSAGEPOOLSIZE** parameter, set the region size higher to maintain the 8 MB gap.

- 3.

Memory consumption on a Classic data server:

If default settings on the Classic data server are insufficient for the size of your environment, you might have to evaluate different configurations in a test environment.

REGION and MESSAGEPOOLSIZE

You specify a **REGION** size in the JCL for the Classic data server to define the maximum amount of memory that the server can allocate for 24-bit (below-the-line) and 31-bit (above-the-line) addresses. **MESSAGEPOOLSIZE** is a global configuration parameter for the Classic data server that defines the amount of storage that the Classic data server reserves and manages for its application requirements.

Site limits can restrict region size. If your site specifies a limit, the Classic data server cannot allocate more, regardless of the **REGION** setting in the JCL. When a Classic data server initializes, it allocates storage for **MESSAGEPOOLSIZE** memory first. The Classic data server cannot access more message pool storage than the limit that you set for **REGION**, regardless of the value of the **MESSAGEPOOLSIZE** parameter.

System exits

A set of system exits are subcomponents of the data server and the query processor. The system exits are designed to run in a multi-user environment and perform security, accounting, and workload management functions to support large numbers of concurrent users.

The following table lists the system exits.

Table 3. System exits.

System exit	Purpose
SAF security exit	<p>Authenticates the user ID and password to the z/OS system.</p> <p>The SAF exit can also perform the following functions:</p> <ul style="list-style-type: none">• Authenticate the TCP/IP address of client connections at when a connection is established.• Verify authority to access a physical file or PSB referenced in an SQL query.• Verify authority to run a stored procedure program.• Pass the user IDs and passwords to CA-Datcom to verify user authorization to access CA-Datcom table references in an SQL query. <p>You activate the SAF exit with the SAFEXIT configuration parameter.</p>
SMF accounting exit	<p>Generates SMF user records that report the CPU time and elapsed time for an individual user session that is connected to a query processor service.</p> <p>You activate the SMF exit with the SMFEXIT configuration parameter.</p>

Table 3. System exits. (continued)

System exit	Purpose
CPU resource governor exit	<p>Restricts the amount of CPU time that a user can consume for a unit-of-work.</p> <p>You activate the CPU resource governor exit with the CPUGOVERNOR configuration parameter.</p>
Workload Manager (WLM) exit	<p>Places queries under the control of the Workload Manager in WLM goal mode. With WLM goal mode, the WLM controls the amount of resources that are available for the query to use. With WLM compatibility mode, you can use monitoring reports to examine resource usage in comparison to site-defined goals.</p> <p>You activate the Workload Manager exit with the configuration parameters defined for the WLM service.</p>
DB2 thread management exit	<p>Validates the user ID before establishing connections to DB2 for z/OS by using the CAF initialization service. A SAF RACROUTE call validates the user ID and establishes the primary authorization ID prior to connecting to DB2 for z/OS.</p> <p>You activate the DB2 thread management exit by following the procedure described in Activating the DB2 thread management exit.</p>
Record processing exit	<p>Modifies record characteristics to make it easier to process VSAM and sequential data.</p> <p>You activate the record processing exit by defining a table mapping for a VSAM or sequential file with the Classic Data Architect.</p>

Data connectors

The query processor dynamically loads one or more data connectors to access the target database or file system that is referenced in an SQL request.

The data connectors provide access to the following data sources:

Adabas

Provides access to Adabas files.

CA-Datcom

Provides access to CA-Datcom files.

CA-IDMS

Provides access to CA-IDMS files.

DB2 for z/OS

Provides access to DB2 for z/OS tables.

IMS Provides access to IMS data using the IMS DRA interface or the IMS BMP/DBB interface.

Sequential

Provides access to sequential files or members.

VSAM

Provides access to native VSAM files, VSAM files under the control of CICS®, and VSAM files under the control of DFSMSStvs.

Classic Data Architect

To process SQL data access requests, data definitions must be mapped to logical tables. The Classic Data Architect is the administrative tool that you use to perform this mapping.

The purpose of the Classic Data Architect is to administer the logical table definitions, views, and SQL security information that are stored in the metadata catalog. You can also use the Classic Data Architect to administer the configuration of the data server and its services.

The key benefits that the Classic Data Architect tool provides make it easier for you to perform the following tasks:

- Define tables, columns, primary keys, indexes, stored procedures, and views.
- Specify user authorization for all objects.
- Import existing physical definitions from copybooks, CA-IDMS schemas, and IMS database descriptors (DBDs).
- Generate DDL for the objects that you create that can be run directly on a server or saved to a script file.
- Generate DDL script from objects already defined in the catalog and export DDL scripts to a data set on the server for use with the metadata utility.
- Connect directly to a Classic data source and view the objects in the system catalog.
- Connect from the Console Explorer to a data server configuration. You can view, modify, import, and export the configuration.

Metadata catalog

The information that you generate from the Classic Data Architect is stored in metadata catalogs. A *metadata catalog* is a set of relational tables that contain information about how to convert data from non-relational to relational formats. The data server accesses the information stored in these catalogs.

Metadata catalogs emulate relational database catalogs. The metadata defines business-oriented relational mappings.

You can use the catalog initialization and maintenance utility to create or perform operations on a metadata catalog.

In addition, the metadata utility accepts DDL that is generated from the Classic Data Architect. The metadata utility must successfully connect to a data server before running any DDL statements. The utility updates the system catalogs with the contents of the DDL statements.

Clients

InfoSphere Classic Federation Server for z/OS provides the ODBC, JDBC, and CLI clients. The clients enable client applications or tools to submit SQL queries to the data server.

The ODBC and CLI clients are based on the ODBC 3.5 standard. The JDBC client is based on a JDBC 3.0 driver.

The clients use TCP/IP communication protocols to establish a connection with a target data server. When your application connects to a data source, the connection handler activates the appropriate communication protocol based on configuration parameters.

SQL statements can be executed using ODBC, JDBC and CLI APIs to access the databases and files that the data server supports. The components that support the APIs include JDBC drivers, ODBC drivers, and CLI drivers.

The ODBC, JDBC, and CLI drivers process function calls, submit SQL requests to a specific data source, and return results to the application.

- For ODBC, a client application links to the ODBC Driver Manager which loads the Classic ODBC driver and calls the driver APIs.
- For CLI, the client application links directly against the Classic CLI driver library. The CLI and ODBC drivers reside in the same library which supports all APIs for both specifications that you can use under both environments.

Stored procedure connectors

The query processor dynamically loads a stored procedure connector when a client issues a CALL statement.

The stored procedure connector manages the processing of the stored procedure. A stored procedure connector instance manages all stored procedures that a particular query processor task processes. It coordinates asynchronous execution of the stored procedure in particular by forwarding the request to a stored procedure service that runs the stored procedure. By doing so, a query processor can process other queries while the stored procedure is being processed.

Chapter 2. Installing Classic federation

Installing Classic federation consists of installing mainframe components and the Classic Data Architect, preparing your installation environment, and customizing the installation to create a functional runtime environment.

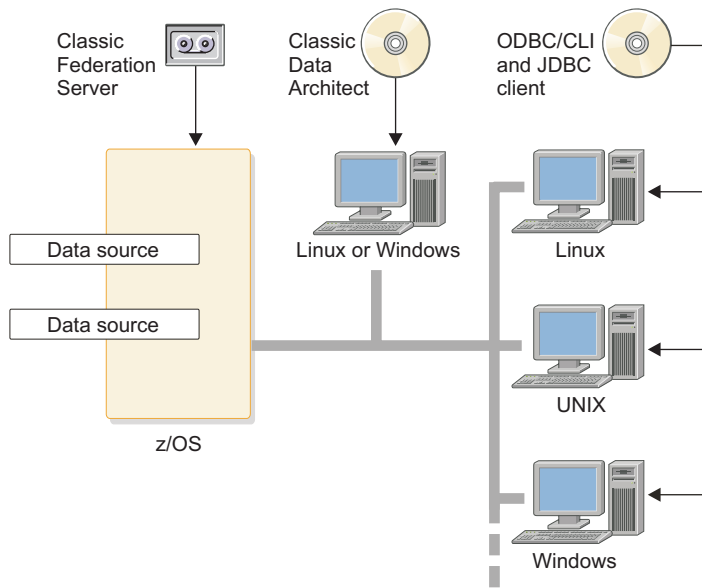
The following table lists the major tasks required for Classic federation installation with a link to where to find information about each task. Perform the tasks in the following recommended order.

Task	Reference
1 Perform the SMP/E tasks to install the components required for Classic federation on the mainframe.	"Installing Classic federation on the mainframe" on page 20
2 Prepare for the installation customization process by completing the tasks in the checklist for setting up the installation environment. Obtaining items such as required authorizations and port numbers will prepare you for the customization procedure.	Setting up the installation environment
3 Customize the server installation environment by completing the tasks in the installation customization process for the type of server that you want to customize.	Customizing the installation environment
4 Install the Classic Data Architect client application to manage server connections and subscriptions, monitor metrics, and perform configuration tasks.	Installing Classic Data Architect
5 Install the ODBC/CLI and JDBC clients	"Installing the ODBC/CLI and JDBC clients" on page 49

System Z Classic federation scenario

IBM InfoSphere Classic Federation Server for z/OS provides SQL access to System Z data sources, such as DB2, IMS, VSAM, Software AG Adabas, CA-Datcom, CA-IDMS, and sequential files.

About this task



Installing Classic federation on the mainframe

The IBM InfoSphere Classic Federation Server for z/OS product is included on tape and the installation instructions are detailed in the product program directory.

About this task

The Program Directory details the system requirements and installation instructions for InfoSphere Classic Federation Server for z/OS.

Setting up the installation environment

After you complete the mainframe SMP/E installation, the next step in the installation process is to set up the installation environment. Setting up the installation environment is a prerequisite to the installation customization process.

The following table provides a checklist of tasks needed to set up the installation environment for Classic data servers.

Table 4. Checklist of installation environment setup for Classic data servers

Task	Reference
Obtain APF library authorizations for the installation load library SCACLOAD	Obtaining library authorizations for the authorized program facility (APF)
Assign port numbers for communication for Classic data servers.	Obtaining ports for communication for Classic data servers
Set up resources profiles and security classes for security for Classic data servers.	Securing a Classic data server

Table 4. Checklist of installation environment setup for Classic data servers (continued)

Task	Reference
Ensure that you have the authorization required to run the Administrative Data Utility (IXCMIAPU). You need this authorization before you run the utility to define the Classic event log and the diagnostic z/OS log streams.	Administrative Data Utility

Customizing the installation environment

The goal of the installation customization process is to simplify the setup of your runtime environment by providing a central place for you to specify the site-specific information that is needed to configure your environment.

The information that you provide is then used as input for generating all JCL and configuration data needed to build the runtime environment.

Installation customization process

Installation customization is a process that allows you to provide setup and configuration information to create a customized installation environment.

The installation customization process involves a set of steps that you perform after you complete the mainframe SMP/E installation. You provide setup and configuration information that is used to generate all of the sample JCL and configuration members in the *USERHLQ.USERSAMP* data set that require edits. You then run installation customization jobs that are generated based on the parameters that you specify to create a customized installation environment.

The installation customization process is based on the role of the Classic data server. The possible roles for a Classic data server are based on the data sources that you choose to access. You can customize an installation environment for one or more data sources. You specify the role of a Classic data server with the *SERVERROLE* parameter for the installation customization utilities. This parameter controls the installation components that you customize. You create installation data sets (*USERHLQ.USERSAMP* and *USERHLQ.USERCONF*) that contain the required components for the type of installation that you choose, and you customize only the parameters needed for that environment.

When you complete the installation customization process, an operational environment is established that you can build upon as needed. The environment includes a functional Classic data server and all of the services required for the specified role. All services are pre-configured during the customization process.

Overview of installation customization procedure

The installation customization process consists of the following basic steps:

1. The user samples allocation utility creates a working set of the *SCACSAMP* and *SCACCONF* data sets that contain all customized JCL and configuration members. This working set is referred to as the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets.
2. You gather site-specific configuration information needed to customize the environment and enter that information in the customization parameters file.

3. The installation customization utility generates customized JCL and configuration members and stores them in the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets that were created in the first step.
4. You allocate and initialize the following components by using generated customization jobs:
 - z/OS log streams
 - Configuration files
 - A zFS aggregate that you define and format
 - Metadata catalogs
5. You use the generated JCL and configuration members to start the runtime environment.

Installation customization components

The following table lists the components and sample JCL members that you use during the installation customization process.

Table 5. Summary of installation customization components distributed in the SCACSAMP data set.

Component name	Description
User samples allocation utility	<p>Allocates the <i>USERHLQ.USERSAMP</i> and <i>USERHLQ.USERCONF</i> data sets. Populates the <i>USERHLQ.USERSAMP</i> data set with a copy of the customization parameters file (CACCUSP2) and the installation customization utility JCL (CACCUSJ2). The SCACSAMP(CACCUSJ1) JCL runs this utility.</p> <p>SCACSAMP(CACCUSJ1) is the JCL that runs the user samples allocation utility. CACCUSJ1 is the only member in the distributed SCACSAMP data set that you edit. The JCL comments provide editing instructions.</p>
Installation customization utility	<p>Reads the customization parameters file <i>USERHLQ.USERSAMP(CACCUSP2)</i> and generates the necessary JCL and configuration members in the <i>USERHLQ.USERSAMP</i> and <i>USERHLQ.USERCONF</i> partitioned data sets. The <i>USERHLQ.USERSAMP(CACCUSJ2)</i> JCL runs this utility.</p> <p><i>USERHLQ.USERSAMP(CACCUSJ2)</i> is the generated JCL that submits the installation customization utility and generates all necessary JCL and configuration members.</p>
Customization parameters file	<p>Contains the installation and customization information that you specify in the form of parameter and value pairs to complete an installation and establish an initial functioning environment. This file is located in <i>USERHLQ.USERSAMP(CACCUSP2)</i>.</p>
<i>USERHLQ.USERSAMP(CACCFSL)</i>	<p>Generated JCL that runs the Administrative Data Utility (IXCMIAPU) to define the z/OS event log stream and a log stream for the diagnostic log for the Classic data server.</p> <p>For VSAM data sources, this member also creates a simple replication log for the IVP VSAM file when CDCRLGST is specified.</p>
<i>USERHLQ.USERSAMP(CECCRZCT)</i>	<p>Generated JCL to define and format a new zFS aggregate to use for a USS file system resident version of the system catalogs.</p> <p>Although you can continue using sequential data set resident system catalogs, zFS file system resident catalogs are recommended.</p>
<i>USERHLQ.USERSAMP(CACCATFG)</i>	<p>Generated JCL that allocates and initializes the configuration files and the metadata catalog for the Classic data server.</p>
<i>USERHLQ.USERSAMP(CACDS)</i>	<p>Generated JCL to start the Classic data server.</p>

Table 5. Summary of installation customization components distributed in the SCACSAMP data set. (continued)

Component name	Description
USERHLQ.USERSAMP(CACCUSVF)	Generated JCL to validate the installation.

User samples allocation utility

The user samples allocation utility allocates the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets and populates the *USERHLQ.USERSAMP* data set with a copy of the customization parameters file and the installation customization utility.

The user samples allocation utility performs these functions:

- Allocates the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets. These data sets are created with the same characteristics as the distributed SCACSAMP and SCACCONF data sets.
If you run the utility again, the *USERHLQ.USERSAMP* or *USERHLQ.USERCONF* data sets that already exist are reused. The utility replaces the customization parameters file and customization utility JCL members. All other members remain the same.
- Generates the customization parameters file *USERHLQ.USERSAMP(CACCUSP2)* and the installation customization utility JCL *USERHLQ.USERSAMP(CACCUSJ2)*. All input parameters specified for the samples allocation utility are populated in the generated CACCUSP2 and CACCUSJ2 members.

You use the SCACSAMP(CACCUSJ1) job to run the allocation utility. The JCL contains comments with editing instructions. You specify the following input as parameters:

CACINHLQ=CAC.V11R1M00

The value specified for the CACINHLQ keyword must be the high-level qualifier of the installation data sets that the SMP/E installation produces.

CACUSHLQ=USER.V11R1M00

The value specified for the CACUSHLQ keyword must be the high-level qualifier for the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets that the samples allocation utility creates or updates.

CACDUNIT=SYSALLDA

The value specified for the CACDUNIT keyword identifies the disk unit that is used when allocating the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets. This is an optional parameter.

CACDVOLM=

The value specified for the CACDVOLM keyword identifies the disk volume that is used when allocating the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets. This is an optional parameter.

CACSTGCL=

The value specified for the CACSTGCL keyword identifies the SMS storage class that is used when allocating the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets. This is an optional parameter.

CACMGTCCL=

The value specified for the CACMGTCCL keyword identifies the SMS management class that is used when allocating the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets. This is an optional parameter.

ISPFLQ=ISP

The value specified for the ISPFLQ keyword identifies the high-level qualifier for ISPF installation. The samples allocation utility runs a TSO batch application and uses TSO functions.

ISPFLANG=ENU

The value specified for the ISPFLANG keyword identifies the language prefix for the ISPF installation.

SERVERROLE=(role-name, ...)

The value of the SERVERROLE keyword specifies that the server environment being installed and customized contains the components required for the Classic data server environment. You can specify one or more roles for your Classic data server. If you specify multiple role names, you must separate the names with commas and enclose the names in parentheses.

CF_ADABAS

Specify this value to install the components required for a Adabas access.

CF_DATACOM

Specify this value to install the components required for CA-Datcom access.

CF_DB2

Specify this value to install the components required for DB2 access.

CF_IDMS

Specify this value to install the components required for CA-IDMS access.

CF_IMS

Specify this value to install the components required for IMS access.

CF_SEQ

Specify this value to install the components required for sequential file access.

CF_VSAM

Specify this value to install the components required for VSAM access.

The samples allocation utility produces a summary report that is written to the SYSTSPRT DD specified in the JCL. The report lists the status for allocating the *USERHLQ.USERXSAMP* and *USERHLQ.USERCONF* data sets and lists the members updated in the *USERHLQ.USERXSAMP* data set.

The following figure shows sample output written to SYSTSPRT.

***** Samples Allocation *****
Summary Report

CACCUSX1 compiled on 2012-09-13 15:31:02 by REXX370 3.48
Execution timestamp: 2012-09-13 15:31:02 MVS Product ID: z/OS 01.10.00 SMF ID: ABC System ID: ABC

Effective Parameters:

CACDUNIT: SYSALLDA
CACDVOLM:
CACINHLQ: CEC.V11R1M00
CACMGTCCL:
CACSTGCL:
CACUSHLQ: USER.V11R1M00.CF_IMS
ISPFHLQ: ISP
ISPFLANG: ENU
SERVERROLE: CF_IMS

Data set 'USER.V11R1M00.CF_IMS.USERSAMP' successfully allocated.
Data set 'USER.V11R1M00.CF_IMS.USERCONF' successfully allocated.

Member	Status
-----	-----
CACCUSJ2	Processed successfully in DDN(USERSAMP)

Member	Status
-----	-----
CACCUSP2	Processed successfully in DDN(USERSAMP)

Summary:
Members Successful: 2
Members in Error: 0
Members Not Replaced: 0
Members Processed: 2

Return Status: 0

Figure 5. Sample output for the samples allocation summary report

Installation customization utility

The installation customization utility generates the JCL and configuration members needed in the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets based on the values that you provide in the customization parameters file.

The installation customization utility performs these functions:

- Captures the customization settings that you provide in the customization parameters file *USERHLQ.USERSAMP(CACCUSP2)*.
- Applies the customization parameters to all JCL members associated with the specified *SERVERROLE* parameter and places the customized members in the *USERHLQ.USERSAMP* data set.
- Applies the customization parameters to all configuration members associated with the specified *SERVERROLE* parameter and places the customized members in the *USERHLQ.USERCONF* data set.

You use the *USERHLQ.USERSAMP(CACCUSJ2)* job to run the installation customization utility. You specify the following input as parameters:

CACINHLQ=CAC.V11R1M00

The value specified for the CACINHLQ keyword must be the high-level qualifier for Classic distribution data sets produced by the SMP/E installation. This value is automatically populated with the value previously specified as input to the user samples allocation utility.

CACUSHLQ=USER.V11R3M00

The value specified for the CACUSHLQ keyword must be the high-level qualifier for the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets that were created or updated by the user samples allocation utility. This value is automatically populated with the value previously specified as input to the user samples allocation utility.

MEMBER=(member-name, ...)

This is an optional parameter. The value specified for the MEMBER keyword identifies a list of one or more member names to process. Only a subset of the members associated with the specified SERVERROLE parameter is processed. If you specify multiple member names, you must separate the names with commas and enclose the names in parentheses.

All members are processed when this parameter is not specified.

OVERWRITE=YES | NO

The value specified for the OVERWRITE keyword indicates how to process existing members of target data sets, for example the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets.

- When you specify OVERWRITE=NO, existing members of the target data sets are not replaced. OVERWRITE=NO is the default.
- When you specify OVERWRITE=YES, existing members of the target data sets are replaced.

Example: OVERWRITE=NO is in effect. Members CACCFGDS and CACCFGUT already exist in the target data set. Member CACSX04 does not exist in the target data set.

```
Member  Status
-----  -----
CACCFGDS  CACCFGDS not replaced in DDN(USERSAMP)
CACCFGUT  CACCFGUT not replaced in DDN(USERSAMP)
CACSX04   Processed successfully in DDN(USERSAMP)
```

Example: OVERWRITE=YES is in effect. The processing status for the same members shown in the previous example appear as follows (whether or not any of these members previously existed in the target data set):

```
Member  Status
-----  -----
CACCFGDS  Processed successfully in DDN(USERSAMP)
CACCFGUT  Processed successfully in DDN(USERSAMP)
CACSX04   Processed successfully in DDN(USERSAMP)
```

The processing summary information produced at the bottom of the report identifies the number of members that were stored successfully and the number of members that were not replaced. For example:

```
Summary:
Members Successful:      90
Members in Error:       0
Members Not Replaced:   0
Members Processed:     90
```

SERVERROLE=(role-name, ...)

The value of the SERVERROLE keyword specifies that the server environment being installed and customized contains the components required for the Classic data server environment. You can specify one or more roles for your Classic data server. If you specify multiple role names, you must separate the names with commas and enclose the names in parentheses.

CF_ADABAS

Specify this value to install the components required for a Adabas access.

CF_DATACOM

Specify this value to install the components required for CA-Datcom access.

CF_DB2

Specify this value to install the components required for DB2 access.

CF_IDMS

Specify this value to install the components required for CA-IDMS access.

CF_IMS

Specify this value to install the components required for IMS access.

CF_SEQ

Specify this value to install the components required for sequential file access.

CF_VSAM

Specify this value to install the components required for VSAM access.

The utility produces a summary report that is written to the SYSTSPRT DD that is specified in the JCL. The report lists the partitioned data sets and the data set members that were processed. The final summary lists the total number of members processed, the number that were successful, and the number with errors.

The following figure shows sample output written to SYSTSPRT.

***** Installation Customization *****
Summary Report

CACCUSX2 compiled on 2012-08-15 08:46:51 by REXXC370 3.48
Execution timestamp: 2012-08-15 08:49:39 MVS Product ID: z/OS 01.10.00 SMF ID: SYE9 System ID:

Effective Parameters:

CACINHLQ: CEC.V11R1M00
CACUSHLQ: USER.V11R1M00.CF
OVERWRITE: No
SERVERROLE: CF_IMS
CF_VSAM

Processing parameters file: USER.V11R1M00.CF.USERSAMP Member: CACCUSP2

Processing Members for Product: All Role: Common

Member	Status
-----	-----
CACCFGDS	Processed successfully in DDN(USERSAMP)
CACCFGUT	Processed successfully in DDN(USERSAMP)
CACPRTL	Processed successfully in DDN(USERSAMP)
CACLGFLT	Processed successfully in DDN(USERSAMP)
CACXS04	Processed successfully in DDN(USERSAMP)

Processing Members for Product: Classic Federation Role: Common

Member	Status
-----	-----
CACBLDI	Processed successfully in DDN(USERSAMP)
CACCATFG	Processed successfully in DDN(USERSAMP)
CACCATLG	Processed successfully in DDN(USERSAMP)
CACCATMD	Processed successfully in DDN(USERSAMP)
CACCATRP	Processed successfully in DDN(USERSAMP)
. . .	
. . .	

Processing Members for Product: Classic Federation Role: CF_IMS

Member	Status
-----	-----
CACBMP	Processed successfully in DDN(USERSAMP)
CACDBB	Processed successfully in DDN(USERSAMP)
CACSVIMA	Processed successfully in DDN(USERCONF)
CACSVIMB	Processed successfully in DDN(USERCONF)
CACSVIMO	Processed successfully in DDN(USERCONF)
CACIMPAR	Processed successfully in DDN(USERSAMP)
CACIMROT	Processed successfully in DDN(USERSAMP)
. . .	
. . .	

Processing Members for Product: Classic Federation Role: CF_VSAM

Member	Status
-----	-----
CACCDEF	Processed successfully in DDN(USERSAMP)
CACSPCCC	Processed successfully in DDN(USERSAMP)
CACSPCCR	Processed successfully in DDN(USERSAMP)
CACSVVSM	Processed successfully in DDN(USERCONF)
CACCAPPL	Processed successfully in DDN(USERSAMP)
CACCMODE	Processed successfully in DDN(USERSAMP)
. . .	
. . .	

Summary:

Members Successful: 32
Members in Error: 0
Members Not Replaced: 0
Members Processed: 32

Working with the customization parameters file

These guidelines describe how to enter values in the customization parameters file.

The customization parameters file, *USERHLQ.USERSAMP(CACCUSP2)*, contains pairs of keyword and value settings used to customize JCL and configuration files in the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets.

The following sections provide guidelines for entering input into the customization parameters file, describe how the file is organized, and list the keyword and value settings that the customization parameters file contains. Other considerations include the use of job cards, pre-defined variables, and STEPLIB concatenations.

Input guidelines

The following guidelines describe how to enter values in the customization parameters file:

- Keyword and value pairs:
 - You cannot change the keyword component.
 - You must delimit the value component with double quotes ("").
 - Spaces are allowed before and after the keyword and value.
 - Values cannot span multiple lines.
- The minimum required parameters that you must change for a successful installation are denoted by an asterisk within parentheses at the end of the comment for that parameter. For example: `CACINHLQ="&CACINHLQ" HLQ of Classic product(*)`
- Comments:
 - An asterisk (*) in column 1 defines the line as a comment line.
 - Any input that you include after the first space after the value component is treated as comments.

File organization

The following table describes the organization of the customization parameters file.

Table 6. Organization of customization parameters file.

Section name	Section content
Common installation	Parameters that apply to all installations, such as the high-level qualifier for the Classic product installation.
Metadata catalog and configuration files	Parameters associated with the metadata catalog and configuration files needed for the data server.
Data server communication parameters	Parameters that define data source name and TCP/IP connection information needed to communicate with a Classic data server environment.
z/OS client parameters	Classic user ID and password used in files that are input into the z/OS sample client.
Security parameters	Security parameters that control user connections to the Classic data server.
Migration and upgrade parameters	Parameters that are specific to the migration of existing installations for each Classic product.
IMS	Parameters specific to installations that access IMS data.

Table 6. Organization of customization parameters file. (continued)

Section name	Section content
CICS VSAM	Parameters specific to installations that access CICS VSAM data.
DB2	Parameters specific to installations that access DB2 data.
Adabas	Parameters specific to installations that access Adabas data.
CA-IDMS	Parameters specific to installations that access CA-IDMS data.
CA-Datacom	Parameters specific to installations that access CA-Datacom data.
Sequential files	Parameters in the common section apply to sequential files.

Use of job cards

Job card information is defined in the common installation section of the customization parameters file. The following two-line job card information is used as a template when generating JCL members:

```
CACDJOB1="JOB (CLASSIC), 'CLASSIC JOB', CLASS=A, "  
CACDJOB2="MSGCLASS=X, NOTIFY=&SYSUID"
```

The CACDJOB1 value is placed after the job name in each generated JCL member. The CACDJOB2 value is provided on the second line of the job card in each JCL member.

The initial value for the job card keywords is populated from the job card that is specified on the JCL member.

Use of pre-defined variables

Many of the data set values in the customization parameters file contain pre-defined variables such as &CAC to reference previously defined high-level qualifiers. Most of the generated JCL members make use of inline PROC definitions. These variables reference the actual PROC variables. The following table describes what each variable defines:

Table 7. Pre-defined variables.

Variable	Description
&CAC	Classic product installation high-level qualifier
&USERHLQ	User SCACSAMP high-level qualifier
&IMS	IMS installation high-level qualifier

Library concatenations

For Classic data server parameters that require specific DD data set concatenation customization such as STEPLIB, parameters are provided for concatenation. You can specify the same parameter keyword multiple times. The order specified for the parameter keywords is the order in which the data sets will be included in the data set concatenation.

Customization parameters file settings:

The parameter keyword and value pairs in the customization parameters file are set to default values. You can modify these values to customize your installation.

The following table lists the parameters in each section of the customization parameters file, the parameter default values, and a description of each parameter.

Table 8. Parameter and default settings for SCACSAMP (CACCUSP2).

Parameter	Default value	Description
Common section		
CACINHLQ	CAC.V11R3M00	High-level qualifier of the installation data sets for the Classic product. This value is populated with the value specified for the CACINHLQ input parameter of the CACCUSJ1 job.
CACUSHLQ	USER.V11R3M00	High-level qualifier for the <i>USERHLQ.USERSAMP</i> and <i>USERHLQ.USERCONF</i> data sets. This value is populated with the value specified for the CACUSHLQ input parameter of the CACCUSJ1 job. This value is also the default high-level qualifier for the metadata catalog and configuration files for the Classic data server.
CACDUNIT	SYSALLDA	Disk unit that is used for the generated jobs that create data sets such as the configuration files for the Classic data server. This value is populated with the value specified for the CACDUNIT input parameter of the CACCUSJ1 job. If the value is " ", it is assumed that the site SMS rules will determine the data set allocation.
CACDVOLM	" "	Disk volume that is used for the generated jobs that create data sets such as the configuration files for the Classic data server. This value is populated with the value specified for the CACDVOLM input parameter of the CACCUSJ1 job. If the value is "", it is assumed that the site SMS rules will determine the data set allocation.
CACSTGCL	" "	SMS storage class that is used for the generated jobs that create data sets such as the configuration files for the Classic data server. This value is populated with the value specified for the CACSTGCL input parameter of the CACCUSJ1 job. If the value is "", it is assumed that the site SMS rules will determine the data set allocation.
CACMGTCL	" "	SMS management class that is used for the generated jobs that create data sets such as the configuration files for the Classic data server. This value is populated with the value specified for the CACMGTCL input parameter of the CACCUSJ1 job. If the value is "", it is assumed that the site SMS rules will determine the data set allocation.

Table 8. Parameter and default settings for SCACSAMP (CACCUSP2). (continued)

Parameter	Default value	Description
CACDJOB1	" "	Pre-populated value from the first line of the JOBCARD provided on the CACCUSJ1 job. The value is substituted into all generated JCL.
CACDJOB2	" "	Pre-populated value from the second line of the JOBCARD provided on the CACCUSJ1 job. The value is substituted into all generated JCL.
Metadata catalog and configuration files		
CACADMUS	CACUSER	User ID to which to grant SYSADM privileges are granted in the metadata catalog.
CATLGMB	150	Megabytes for the primary catalog allocation. For guidance in determining the actual number of MB you require, see Estimating the size of the metadata catalog.
CATPATH	/opt/IBM/isclassic111/catalog	USS file system path to the metadata catalog files. The named directory contains the following file names: cacat Data component cacindx Index component zFS file system resident catalogs are recommended (rather than other file systems, such as physical sequential). When this parameter is specified, it supersedes CACCATNM and CACIDXNM and NEWCATNM and NEWIDXNM.
CACCATNM	&USRHLQ..CATALOG	Suffix for the Version 11.3 metadata catalog data file created during the installation customization process. The &USRHLQ value is replaced in the generated JCL PROC with the high-level qualifier specified for the CACUSHLQ parameter.
CACIDXNM	&USRHLQ..CATINDEX	Suffix for the Version 11.3 metadata catalog index file created during the installation customization process. The &USRHLQ value is replaced in the generated JCL PROC with the high-level qualifier specified for the CACUSHLQ parameter.
CACCFGNM	&USRHLQ..CACCFGD	Suffix for the Version 11.3 configuration data file created during the installation customization process. The &USRHLQ value is replaced in the generated JCL procedure with the high-level qualifier specified for the CACUSHLQ parameter.
CACCFGIX	&USRHLQ..CACCFGX	Suffix for the Version 11.3 configuration index file created during the installation customization process. The &USRHLQ value is replaced in the generated JCL procedure with the high-level qualifier specified for the CACUSHLQ parameter.

Table 8. Parameter and default settings for SCACSAMP (CACCUSP2). (continued)

Parameter	Default value	Description
DIAGLGST	CF.DIAGLOG	z/OS log stream name for the diagnostic log for the Classic data server. If a log stream name is not specified, the log service is configured to write to the CACLOG DD data set which is a temporary data set.
DIAGLGDS	Y	Identifies whether the z/OS log stream should use DASD or the coupling facility: <ul style="list-style-type: none"> • Y: DASD • N: Coupling facility This value is valid when DIAGLGST is specified.
DIAGLGRT	7	Retention period, in days, to retain the log records before they are eligible to be deleted. This value is valid when DIAGLGST is specified.
DIAGLGSC	STG1	Storage class (STG_DATACLAS) for the log stream. This value is valid when DIAGLGST is specified.
DIAGLGSR	CCL1	Coupling facility structure name (STRUCTNAME). This value is valid when DIAGLGST is specified and the coupling facility is chosen (DIAGLGDS="N").
Classic data server communication parameters		
DSDSRCE	CACSAMP	Data source name for the query processor service. This is the data source name that clients use to connect to the Classic data server.
DSHOST	0.0.0.0	Host name or IP address where the Classic data server will run.
DSPORT	9087	Port number on which the connection handler service for the Classic data server will listen. This listen port is used to communicate with the CDA and other client applications.
z/OS client parameters		
CLNTUSER	CACUSER	User ID to use in z/OS client application samples such as the metadata utility connection string and the sample SQL file.
CLNTPSWD	CACPWD	Password associated with the user ID provided for the CLNTUSER parameter.
Security parameters		
CFSAFX	CACSX04	SAFEXIT load module that enables security for the Classic data server. Specifying a value enables security for the Classic data server. A value of " " disables security for the Classic data server.

Table 8. Parameter and default settings for SCACSAMP (CACCUSP2). (continued)

Parameter	Default value	Description
CFSAFVLD	N	VALIDATE=Y/N parameter on the query processor service for the SAFEXIT, when the exit is enabled by using the CFSAFX customization parameter. This value instructs the service to validate the access authority of a user by using additional security parameters: CFSAFNET - NETACCESS CFSAFSPC - Stored procedure SPCLASS CFSAFIMC - IMS PSB schedule class CFSAFIMP - IMS PSB prefix CFSAFDCE - CA-Datacom EXCLUDE CFSAFADC - Adabas ADACLASS
CFSAFNET	None	NETACCESS parameter on the query processor service for the SAFEXIT, when the exit is enabled and the CFSAFVLD customization parameter = Y. This value (Y/N) specifies if IP address validation is enabled.
CFSAFSPC	None	SPCLASS parameter on the query processor service for the SAFEXIT, when the exit is enabled and the CFSAFVLD customization parameter = Y. This value specifies the name of a class that is used to check for RACF-authorized use of stored procedure names.

Migration and upgrade parameters: The parameters in this section apply to installations that you need to migrate from a previous release of Classic federation to Version 11.3. If a migration is not required, use the default values. When migrating or upgrading, new catalog information is obtained from the parameters defined in the 'Metadata catalog and configuration files' section above.

OLDCAHLQ	CAC.V10R1M00	High-level qualifier for the pre-version 11.3 product installation.
OLDCTHLQ	CAC.V10R1M00	High-level qualifier for the pre-version 11.3 metadata catalog that is being upgraded.
OLDCPATH	None	If upgrading an existing zFS 11.3 catalog, use this parameter. When defined, this parameter supersedes OLDCATNM and OLDIDXNM. Specifies the USS file system path to the metadata catalog files. (for example, /opt/IBM/isclassic111/catalog). The named directory contains the following file names: cacat Data component cacindx Index component
OLDCATNM	&OLDCAT..CATALOG	Suffix for the pre-version 11.3 metadata catalog data file. The &OLDCAT value will be replaced in the generated JCL PROC with the high-level qualifier specified for the OLDCTHLQ parameter.

Table 8. Parameter and default settings for SCACSAMP (CACCUSP2). (continued)

Parameter	Default value	Description
OLDCATIX	&OLDCAT..CATINDEX	Suffix for the pre-version 11.3 metadata catalog index file. The &OLDCAT value will be replaced in the generated JCL PROC with the high-level qualifier specified for the OLDCTHLQ parameter.
OLDCFHLQ	CAC.V10R1M00	High-level qualifier for the pre-version 11.3 configuration files that are being upgraded. <ul style="list-style-type: none"> • If this value specifies a release prior to version 9.5 it will be the HLQ for the SCACCONF data set. • If this value specifies a version 9.5 release it will be the HLQ for the Classic data server configuration files (CACCFGD and CACCFGX).
OLDCFGNM	&OLDCFG..SCACCONF	If you are migrating from a release prior to version 9.5, this value specifies the SCACCONF data set that contains the configuration member. This data set is associated with the VHSCONF DD statement in the pre-version 9.5 startup JCL for the Classic data server.
OLDDSCFG	CACDSCF	If you are migrating from a release prior to version 9.5, this value specifies the configuration member name in the SCACCONF data set. This member is associated with the VHSCONF DD statement in the pre-version 9.5 startup JCL for the Classic data server.
OLDCFGNM	&OLDCFG..CACCFGNM	Suffix for the post-version 9.1 configuration data file. The & OLDCFG value is replaced in the generated JCL PROC with the high-level qualifier specified for the OLDCFHLQ parameter.
OLDCFGIX	&OLDCFG..CACCFGIX	Suffix for the post-version 9.1 configuration index file. The & OLDCFG value is replaced in the generated JCL PROC with the high-level qualifier specified for the OLDCFHLQ parameter.
NEWCFGNM	&NEWCFG..CACCFGNM	Suffix for the new version 11.3 configuration data file. The & NEWCFG value is replaced in the generated JCL PROC with the high-level qualifier specified for the NEWCTHLQ parameter.
NEWCFGIX	&NEWCFG..CACCFGIX	Suffix for the new version 11.3 configuration index file. The & NEWCFG value is replaced in the generated JCL PROC with the high-level qualifier specified for the NEWCTHLQ parameter.

IMS parameters: The parameters in this section are specific to Classic data servers that access IMS. See Setting up access to IMS for details about configuring Classic data servers for IMS access.

Table 8. Parameter and default settings for SCACSAMP (CACCUSP2). (continued)

Parameter	Default value	Description
IMSINHLQ	IMS	High-level qualifier (HLQ) for THE IMS installation libraries. This HLQ is applied to the IMS data sets in the STEPLIB sections of the generated JCL. It replaces the references to &IMS.. below in other keyword values such as IMSSTEPL.
IMSDFPSB	DEFPSB	Default PSB that the DRA initialization service or the ODBA service uses to access IMS.
IMSDRAUS	DRAUSER	User ID that the DRA initialization service uses to access IMS.
IMSSSID	SSID	IMS subsystem ID that the ODBA initialization service uses
IMSPZPSF	00	DRA startup table suffix (DFSPZPxx) that the DRA initialization service uses.
IMSSTEPL	&IMS..SDFSRESL	STEPLIB concatenation for files needed to access IMS. The IMSTEPL is a keyword that you can specify multiple times. It provides a STEPLIB concatenation. The order of the multiple IMSSTEPL keywords defines the order in which the files are included in the STEPLIB concatenation for the generated JCL members. The &IMS value is replaced in the generated JCL PROC with the high-level qualifier specified for the IMSINHLQ parameter.
IMSDBDLB	&CICS..DBDLIB	DBDLIB used by the Classic data server to locate DBDs during CREATE TABLE processing. The IMSDBDLB is a keyword that you can specify multiple times. It provides a DBDLIB concatenation. The order of the multiple IMSDBDLB keywords defines the order in which the files is included in the DBDLIB concatenation for generated the generated JCL members. The &IMS value is replaced in the generated JCL PROC with the high-level qualifier specified for the IMSINHLQ parameter.
CFSAFIMC	" "	IMS PSB schedule class parameter on the query processor service for the SAFEXIT when the exit is enabled and the CFSAFVLD customization parameter = Y. This parameter specifies the name of the RACF® resource class that is checked to determine whether the user has authority to schedule or access the PSBs associated with the tables referenced in a query

CICS VSAM parameters The parameters in this section are specific to Classic data servers that access CICS VSAM files. See Setting up access to CICS VSAM for details about configuring Classic data servers for CICS VSAM access.

Table 8. Parameter and default settings for SCACSAMP (CACCUSP2). (continued)

Parameter	Default value	Description
VSCINHLQ	CICSTSn.CICS	High-level qualifier (HLQ) for the CICS libraries. This HLQ is applied to the CICS data sets in the STEPLIB sections of the generated JCL. It replaces the references to &CICS.. below in other keyword values such as VSCSTEPL and VSCDFHCS.
VSCSTEPL	&CICS..SDFHLOAD	STEPLIB concatenation for files needed to access CICS. The VSCSTEPL is a keyword that you specify multiple times. It provides a STEPLIB concatenation. The order of the multiple VSCSTEPL keywords defines the order in which the files are included in the STEPLIB concatenation for the generated JCL members. The &CICS value is replaced in the generated JCL PROC with the high-level qualifier specified for the VSCINHLQ parameter.
VSCDFHCS	&CICS.. DBDLIB	DFHCS data set to use when defining the Classic federation programs and transactions to CICS.
VSCNETNM	CACCICS1	CICS NETNAME to use when defining the Classic federation programs and transactions to CICS.

DB2 parameters: The parameters in this section are specific to Classic data servers that access DB2. See Setting up access to DB2 for z/OS for details about configuring Classic data servers for DB2 access.

DB2INHLQ	DB2	High-level (HLQ) qualifier for the DB2 libraries. This HLQ is applied to the DB2 data sets in the STEPLIB sections of the generated JCL. It replaces the references to &DB2.. below in other keyword values such as DB2STEPL.
DB2PLAN	CAC10PLN	DB2 plan name that the DB2 call attachment facility (CAF) access service uses.
DB2DSN	DSN	DB2 subsystem ID that the DB2 CAF access service uses.
DB2STEPL	&DB2..SDSNLOAD	DB2 STEPLIB concatenation for files needed to access DB2. The DB2STEPL is a keyword that you can specify multiple times. It provides a STEPLIB concatenation. The order of the multiple DB2STEPL keywords defines the order in which the files are included in the STEPLIB concatenation for the generated JCL members. The &DB2 value is replaced in the generated JCL PROC with the high-level qualifier specified for the DB2INHLQ parameter.

Adabas parameters The parameters in this section are specific to Classic data servers that access Adabas. See Setting up access to Adabas databases for details about configuring Classic data servers for Adabas access.

Table 8. Parameter and default settings for SCACSAMP (CACCUSP2). (continued)

Parameter	Default value	Description
ADAINHLQ	ADABAS	High-level (HLQ) qualifier for the Adabas libraries. This HLQ is applied to the Adabas data sets in the STEPLIB sections of the generated JCL. It replaces the references to &ADA.. below in other keyword values such as ADASTEPL.
ADAEPSVC	251	SVC number of the Adabas nucleus.
ADAEPDID	32	Default database ID for Adabas.
ADALNK8	Y	Version of the Adabas system that is being accessed. The values are either version 8 or a prior version. <ul style="list-style-type: none"> • Y: Indicates version 8 or higher. • N: Indicates a version prior to version 8.
ADASTEPL	&ADA..LOAD	STEPLIB concatenation for files needed to access Adabas. The ADASTEPL is a keyword that you can specify multiple times. It provides a STEPLIB concatenation. The order of the multiple ADASTEPL keywords defines the order in which the files are included in the STEPLIB concatenation for the generated JCL members. The &ADA value is replaced in the generated JCL PROC with the high-level qualifier specified for the ADAINHLQ parameter.
CFSAFADC	None	ADAClass parameter on the query processor service for the SAFEXIT when the exit is enabled and the CFSAFVLD customization parameter = Y.

CA-IDMS parameters: The parameters in this section are specific to Classic data servers that access CA-IDMS. See Setting up access to CA-IDMS for details about configuring Classic data servers for CA-IDMS access.

IDMINHLQ	IDMS	High-level qualifier (HLQ) for the CA-IDMS libraries. This HLQ is applied to the CA-IDMS data sets in the STEPLIB sections of the generated JCL. It replaces the references to &IDMS.. below in other keyword values such as IDMSYCTL and IDMSTEPL.
IDMSYCTL	&IDMS..SYSCTL	CA-IDMS SYSCTL file that the Classic data server uses to access the CA-IDMS central version. The &IDMS value is replaced in the generated JCL PROC with the high-level qualifier specified for the IDMINHLQ parameter.
IDMGDMCL	GLBLDMCL	Global DMCL that the CA-IDMS central version uses. This value is specified in the SYSIDMS setting for the Classic data server. This parameter allows the CA-IDMS client layer to obtain access to the DDLMSG area to report errors if necessary.

Table 8. Parameter and default settings for SCACSAMP (CACCUSP2). (continued)

Parameter	Default value	Description
IDMDCMSG	&IDMS..SYSMSG.DDLDCMSG	Local data set for the DDLDCMSG area that the CA-IDMS central version uses. This parameter allows the CA-IDMS client layer to obtain access to the DDLMSG area to report errors if necessary. The &IDMS value is replaced in the generated JCL PROC with the high-level qualifier specified for the IDMINHLQ parameter.
IDMACLIB	&IDMS..DISTMAC	CA-IDMS MACLIB. This parameter is used when setting up security for accessing CA-IDMS from Classic federation to relink a new IDMSSTRT module. The &IDMS value is replaced in the generated JCL PROC with the high-level qualifier specified for the IDMINHLQ parameter.
IDMSTEPL	&IDMS..DBA.LOADLIB	STEPLIB concatenation for files needed to access CA-IDMS. The IDMSTEPL is a keyword that you can specify multiple times. It provides a STEPLIB concatenation. The order of the multiple IDMSTEPL keywords defines the order in which the files are included in the STEPLIB concatenation for the generated JCL members. The &IDMS value is replaced in the generated JCL PROC with the high-level qualifier specified for the IDMINHLQ parameter.
CA-Datacom parameters: The parameters in this section are specific to Classic data servers that access CA-Datacom. See Setting up access to CA-Datacom for details about configuring Classic data servers for CA-Datacom access.		
DCMINHLQ	DCOM	High-level qualifier (HLQ) for the CA-Datacom libraries. This HLQ is applied to the CA-Datacom data sets in the STEPLIB sections of the generated JCL. It replaces the references to &DC.. below in other keyword values such as DCMACLIB and DCMSTEPL.
DCMACLIB	&DC..CAI.DATAACOM.THLQ.CAIMAC	CA-Datacom MACLIB. This parameter is used to assemble the sample CACDCURT User Requirements Table. The &DC value is replaced in the generated JCL PROC with the high-level qualifier specified for the DCMINHLQ parameter.
DCMDDIDT	&USRHLQ..SCACSAMP(CACDCID)	DDIDENT data set that the Classic data server uses. It contains a user ID and password to connect to the CA-Datacom dictionary search facility. The Classic data server accesses the CA-Datacom dictionary during CREATE TABLE processing.
DCMUSER	DATAACOM-INSTAL	CA-Datacom user ID to use when accessing the CA-Datacom dictionary search facility. This user ID is placed in the DDIDENT data set specified by the DCMDDIDT parameter.

Table 8. Parameter and default settings for SCACSAMP (CACCUSP2). (continued)

Parameter	Default value	Description
DCMPASWD	NEWUSER	Password for the CA-Datcom user ID to use when accessing the CA-Datcom dictionary search facility. This password is placed in the DDIDENT data set specified by the DCMDDIDT parameter.
DCMSTEPL	&DC..CAI.CAILIB &DC..DATACOM.CHLQ.CUSLIB &DC..DATACOM.THLQ.CAILIB	STEPLIB concatenation for files needed to access CA-Datcom. The DCMSTEPL is a keyword that you can specify multiple times. It provides a STEPLIB concatenation. The order of the multiple DCMSTEPL keywords defines the order in which the files are included in the STEPLIB concatenation for the generated JCL members. The &DC value is replaced in the generated JCL PROC with the high-level qualifier specified for the DCMINHLQ parameter.
CFSAFDCE	None	EXCLUDE parameter on the query processor service for the SAFEXIT when the exit is enabled and the CFSAFVLD customization parameter = Y. This parameter specifies that the query processor service should not provide an ACEE address in commands sent to CA-Datcom.
Language Environment (LE) and compiler parameters		
CACLEHLQ	SYS1	High-level qualifier for the LE runtime SCEELKED data set.
CACCBCMP	IGYCRCTL	COBOL compiler name to use when compiling a stored procedure.
CACCBHLQ	SYSL	High-level qualifier for the SIGYCOMP data set.
CACCSTEP	&COBOL..SIGYCOMP	COBOL libraries to include in the STEPLIB for compile sample members. CACCSTEP is a keyword that you can specify multiple times. This keyword provides a STEPLIB concatenation. The order of the multiple CACCSTEP keywords defines the order in which the files is included in the STEPLIB concatenation for the generated JCL members. The &COBOL value is replaced in the generated JCL PROC with the high-level qualifier specified for the CACCBHLQ parameter.

Installing Classic data servers

The installation customization steps that you follow depend on the product that you are using and if you are creating a new installation or upgrading to Version 11.1 from a previous version of Classic federation.

About this task

Procedure

Follow the appropriate installation customization steps for either a new installation or an installation upgrade from an earlier version of Classic federation:

Procedure

- If you are creating a new Version 11.1 installation, follow the installation customization steps for a new Classic federation installation.
- If you are upgrading to Version 11.1 from a previous version, follow the steps for upgrading Classic federation.

Installing a new Classic federation data server

You can follow the installation customization process to install and customize a Classic federation data server.

Before you begin

Before you begin the installation customization process, you must complete the SMP/E installation and the steps required to “Setting up the installation environment” on page 20.

About this task

When setting up the metadata catalog, the default configuration creates and initializes the metadata catalog as a zSeries File System (zFS) file. The z/FS file provides significant performance and capacity improvements compared to the sequential or linear data set formats used in releases prior to V11.1.

The topic *Creating and initializing zFS metadata catalogs* provides more information about the use of zFS metadata catalogs.

Procedure

1. Edit the user samples allocation utility JCL in the installation samples member SCACSAMP(CACCUSJ1). Follow the instructions in the JCL to edit the job card and procedure variables. Then specify input parameters to the samples allocation utility. You specify the following input parameters:

CACINHLQ=CAC.V11R3M00

The value specified for the CACINHLQ keyword must be the high-level qualifier of the installation data sets that the SMP/E installation produces for Classic federation.

CACUSHLQ=USER.V11R3M00

The value specified for the CACUSHLQ keyword is the high-level qualifier for the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets that the samples allocation utility creates or updates.

CACDUNIT=SYSALLDA

The value specified for the CACDUNIT keyword identifies the disk unit that is used when allocating the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets. This is an optional parameter. You do not need to specify a value for CACDUNIT if SMS manages the data sets.

CACDVOLM=

The value specified for the CACDVOLM keyword identifies the disk volume that is used when allocating the *USERHLQ.USERSAMP* and

USERHLQ.USERCONF data sets. This is an optional parameter. You do not need to specify a value for *CACDVOLM* if SMS manages the data sets.

CACSTGCL=

The value specified for the *CACSTGCL* keyword identifies the SMS storage class that is used when allocating the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets. This is an optional parameter. Specify a value for *CACSTGCL* only when a specific storage class is required.

CACMGTCCL=

The value specified for the *CACMGTCCL* keyword identifies the SMS management class that is used when allocating the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets. This is an optional parameter. Specify a value for *CACMGTCCL* only when a specific management class is required.

ISPFHLQ=ISP

The value specified for the *ISPFHLQ* keyword identifies the high-level qualifier for ISPF installation. The samples allocation utility runs a TSO batch application and uses TSO functions.

ISPFLANG=ENU

The value specified for the *ISPFLANG* keyword identifies the language prefix for the ISPF installation.

SERVERROLE=role-name

The value specified for the *SERVERROLE* keyword identifies the server environment to install and customize. The role determines the JCL members that are customized and the services that are configured for the Classic data server. The list below contains the possible values. You can specify multiple values at the same time by using the following syntax:

SERVERROLE=(role-name1, role-name2)

Valid values for role-name:

SERVERROLE value	Installs components for ...
CF_ADABAS	Adabas access
CF_DATACOM	CA-Datcom
CF_DB2	DB2 access
CF_IDMS	CA-IDMS access
CF_IMS	IMS access
CF_SEQ	Sequential file access
CF_VSAM	VSAM access

2. Submit *SCACSAMP(CACCUSJ1)* to allocate the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets. Verify that all job steps result in a return code <= 4. This job populates the *USERHLQ.USERSAMP* data set with the necessary objects and the customization parameters file for Classic federation, *CACCUSP2* .
3. Edit the Classic federation customization parameters file *USERHLQ.USERSAMP(CACCUSP2)* to provide customization parameters.

This file will contain only the parameters that apply to the specified SERVERROLE. See the customization parameters file settings for details.

4. Submit the generated *USERHLQ.USERSAMP(CACCUSJ2)* customization utility JCL. Verify that all job steps result in a return code ≤ 4 . The *USERHLQ.USERSAMP* data set is populated with the customized JCL and objects needed to run a Classic data server for the specified SERVERROLE.
5. Optional: Define the z/OS log stream for the diagnostic log for the Classic federation data server. This step is only needed when the *DIAGLGST* or *EVENLGST* parameter in the *CACCUSP2* files specifies a log stream name.
 - a. Verify that you have the authority required to run the Administrative Data Utility (*IXCMIAPU*). The job that defines the logs runs this utility.
 - b. Submit the generated *USERHLQ.USERSAMP(CACCFSL)* JCL to define the log stream and logs.
 - c. Verify that all job steps result in a return code = 0.
6. Optional: If the *CATPATH* value specified that the metadata catalog files will reside in a new zFS file system aggregate, you must define and format the new zFS aggregate.
 - a. Verify that you have the authority required to run the following jobs:
 - SAF READ-access level authorization is required for the *SUPERUSER.FILESYS.PFSC* resource in the *UNIXPRIV* class to run the zFS administration command, *IOEZADM*.
 - SAF READ-access level authorization is required for the *SUPERUSER.FILESYS.MOUNT* resource in the *UNIXPRIV* class to perform *MOUNT* and *UNMOUNT* operations against USS file systems.
 - b. View and edit *USERHLQ.USERSAMP(CECCRZCT)* job cards.
 - c. Submit *USERHLQ.USERSAMP(CECCRZCT)*.
 - d. Verify that all job steps result in a return code = 0.
 - e. Mount the file systems using the sample *MOUNT* command provided in *USERHLQ.USERSAMP(CECCRZCT)*.
7. Submit the generated *USERHLQ.USERSAMP(CACCATFG)* JCL to allocate and initialize the metadata catalog and allocate the configuration files for the Classic data server. This job populates the configuration with the service definitions required for the specified SERVERROLE.
8. Submit the generated *USERHLQ.USERSAMP(CACDS)* JCL to start the Classic data server.
9. Run the validation job in the next step if security is enabled for the Classic federation data server. If security is not enabled, skip this step and continue with the next step to run the validation job.

Security is enabled for the Classic data server by setting the *CFSAFX* parameter in the installation customization parameters file. If security is enabled, the following steps are required before you run the validation job in the final step. If security is not enabled, continue with the next step.

Enabling security requires providing a password for the user access. The utilities used in the validation job require encrypted passwords to access the server. Set up the encrypted password for the validation job by following these steps:

- a. Edit the generated *USERHLQ.USERCONF(CACPWDIN)* member. This member provides a *PASSWORD=value* parameter for the password generator utility. Set the value to the TSO password for the User ID that you use to run the customization jobs.

- b. Submit the generated *USERHLQ.USERSAMP(CACENCRP)* JCL to run the password generator utility. This JCL updates the *USERHLQ.USERCONF(CACPWDIN)* with the encrypted value of the password provided in the previous step.
 - c. Edit the *USERHLQ.USERCONF(CACPWDIN)* member and copy the hex string value (x'<16-byte hexadecimal value>') for the ENCRYPTED= keyword.
 - d. Edit the generated *USERHLQ.USERCONF(CACMUCON)* member and replace the X'.ENCRYP PASSWD..' string with the hex string copied in the previous step.
 - e. Edit the generated *USERHLQ.USERSAMP(CACQRSYS)* member and replace the second line of the member with the hex string that you copied to ensure that the hex string starts in the first column
10. Submit the generated *USERHLQ.SCACSAMP(CACCUSVF)* validation job. Verify that all job steps result in a return code <= 4.
 11. You can now configure the Classic Data Architect to access the Classic data server, select from the sample tables, and map additional tables.

Upgrading a Classic federation installation

To upgrade a Classic federation environment to Version 11.3 from a previous version, complete the steps in the update process.

Procedure

1. Edit the user samples allocation utility JCL in the installation samples member *SCACSAMP(CACCUSJ1)*. Follow the instructions in the JCL to edit the job card and procedure variables. Then specify input parameters to the samples allocation utility. You specify the following input parameters:

CACINHLQ=CAC.V11R3M00

The value specified for the CACINHLQ keyword must be the high-level qualifier of the installation data sets that the SMP/E installation produces for Classic federation.

CACUSHLQ=USER.V11R3M00

The value specified for the CACUSHLQ keyword is the high-level qualifier for the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets that the samples allocation utility creates or updates.

CACDUNIT=SYSALLDA

The value specified for the CACDUNIT keyword identifies the disk unit that is used when allocating the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets. This is an optional parameter. You do not need to specify a value for CACDUNIT if SMS manages the data sets.

CACDVOLM=

The value specified for the CACDVOLM keyword identifies the disk volume that is used when allocating the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets. This is an optional parameter. You do not need to specify a value for CACDVOLM if SMS manages the data sets.

CACSTGCL=

The value specified for the CACSTGCL keyword identifies the SMS storage class that is used when allocating the *USERHLQ.USERSAMP*

and *USERHLQ.USERCONF* data sets. This is an optional parameter. Specify a value for CACSTGCL only when a specific storage class is required.

CACMGTCCL=

The value specified for the CACMGTCCL keyword identifies the SMS management class that is used when allocating the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets. This is an optional parameter. Specify a value for CACMGTCCL only when a specific management class is required.

ISPFHLQ=ISP

The value specified for the ISPFHLQ keyword identifies the high-level qualifier for ISPF installation. The samples allocation utility runs a TSO batch application and uses TSO functions.

ISPFLANG=ENU

The value specified for the ISPFLANG keyword identifies the language prefix for the ISPF installation.

SERVERROLE=role-name

The value specified for the SERVERROLE keyword identifies the server environment to install and customize. The role determines the JCL members that are customized and the services that are configured for the Classic data server. The list below contains the possible values. You can specify multiple values at the same time by using the following syntax:

`SERVERROLE=(role-name1, role-name2)`

Valid values for role-name:

SERVERROLE value	Installs components for ...
CF_ADABAS	Adabas access
CF_DATACOM	CA-Datcom
CF_DB2	DB2 access
CF_IDMS	CA-IDMS access
CF_IMS	IMS access
CF_SEQ	Sequential file access
CF_VSAM	VSAM access

2. Submit SCACSAMP(CACCUSJ1) to allocate the *USERHLQ.USERSAMP* and *USERHLQ.USERCONF* data sets. Verify that all job steps result in a return code <= 4. This job populates the *USERHLQ.USERSAMP* data set with the necessary objects and the customization parameters file for Classic federation, CACCUSP2 .
3. Edit the Classic federation customization parameters file *USERHLQ.USERSAMP(CACCUSP2)* to provide customization parameters. This file will contain only the parameters that apply to the specified SERVERROLE. See the customization parameters file settings for details.
4. Submit the generated *USERHLQ.USERSAMP(CACCUSJ2)* customization utility JCL. Verify that all job steps result in a return code <= 4. The *USERHLQ.USERSAMP* data set is populated with the customized JCL and objects needed to run a Classic data server for the specified SERVERROLE.

5. Optional: Define the z/OS log stream for the diagnostic log for the Classic federation data server. This step is only needed when the DIAGLGST or EVENLGST parameter in the CACCUSP2 files specifies a log stream name
 - a. Verify that you have the authority required to run the Administrative Data Utility (IXCMIAPU). The job that defines the logs runs this utility.
 - b. Submit the generated *USERHLQ.USERSAMP(CACCFSL)* JCL to define the log stream and logs.
 - c. Verify that all job steps result in a return code = 0.
6. Optional: If the system catalog files reside in a new zFS file system aggregate, you must define and format the new zFS aggregate.
 - a. Verify that you have the authority required to run the following jobs:
 - SAF READ-access level authorization is required for the SUPERUSER.FILESYS.PFSCCTL resource in the UNIXPRIV class to run the zFS administration command, IOEZADM.
 - SAF READ-access level authorization is required for the SUPERUSER.FILESYS.MOUNT resource in the UNIXPRIV class to perform MOUNT and UNMOUNT operations against USS file systems.
 - b. View and edit *USERHLQ.USERSAMP(CECCRZCT)* job cards.
 - c. Submit *USERHLQ.USERSAMP(CECCRZCT)*.
 - d. Verify that all job steps result in a return code = 0.
7. Submit the generated *USERHLQ.USERSAMP(CACCATUP)* member to upgrade the existing metadata catalog.
8. Allocate the configuration files and migrate the configuration.
 - If you are migrating from a Version 10.1 release or later, submit the generated *USERHLQ.USERSAMP(CACCFGM1)* member.
 - If you are migrating from a Version 9.5 release, submit the generated *USERHLQ.USERSAMP(CACCFGMD)* member.
 - If you are migrating from a pre-Version 9.5 release, submit the generated *USERHLQ.USERSAMP(CACCFGMV)* member.

These jobs allocate the configuration files for the Classic data server and run the configuration migration and maintenance utility, *CACCFGUT*, to migrate the configuration.

9. Submit the generated *USERHLQ.USERSAMP(CACDS)* JCL to start the Classic data server.
10. Run the validation job in the next step if security is enabled for the Classic federation data server. If security is not enabled, skip this step and continue with the next step to run the validation job.

Security is enabled for the Classic data server by setting the CFSAFX parameter in the installation customization parameters file. If security is enabled, the following steps are required before you run the validation job in the final step. If security is not enabled, continue with the next step.

Enabling security requires providing a password for the user access. The utilities used in the validation job require encrypted passwords to access the server. Set up the encrypted password for the validation job by following these steps:

- a. Edit the generated *USERHLQ.USERCONF(CACPWDIN)* member. This member provides a `PASSWORD=value` parameter for the password generator utility. Set the value to the TSO password for the User ID that you use to run the customization jobs.

- b. Submit the generated *USERHLQ.USERSAMP(CACENCRP)* JCL to run the password generator utility. This JCL updates the *USERHLQ.USERCONF(CACPWDIN)* with the encrypted value of the password provided in the previous step.
 - c. Edit the *USERHLQ.USERCONF(CACPWDIN)* member and copy the hex string value (x'<16-byte hexadecimal value>') for the ENCRYPTED= keyword.
 - d. Edit the generated *USERHLQ.USERCONF(CACMUCON)* member and replace the X'.ENCRYP PASSWD..' string with the hex string copied in the previous step.
 - e. Edit the generated *USERHLQ.USERSAMP(CACQRSYS)* member and replace the second line of the member with the hex string that you copied to ensure that the hex string starts in the first column
11. Submit the generated *USERHLQ.SCACSAMP(CACCUSVF)* validation job. Verify that all job steps result in a return code <= 4.
 12. You can now configure the Classic Data Architect to access the Classic data server, select from the sample tables, and map additional tables.

Installing the Classic Data Architect

To install the Classic Data Architect you extract an installation zip package and run IBM Installation Manager.

Before you begin

If you have installed an earlier beta version of the Classic Data Architect (CDA) Version 11.3, you must uninstall it first. See “Uninstalling the Classic Data Architect” on page 49.

Ensure that your client computer meets the following minimum system requirements:

Operating system:

- Microsoft Windows 8.1, 8, and 7

Memory

1024 MB.

Disk space

800 MB for both IBM Installation Manager and the Classic Data Architect.

Procedure

Start the CDA installation as a non-Administrator or as an Administrator.

Method	Description
To start the CDA installation as a non-Administrator	<ol style="list-style-type: none"> 1. Unzip the installation package to a temporary directory. 2. From the command line, change to the temporary directory and take one of the following actions: <ul style="list-style-type: none"> • On Windows: Run <code>userinst.exe</code>. • On Linux: Run <code>userinst</code>. 3. Proceed through the Installation Manager wizard.

Method	Description
To start the CDA installation as an Administrator	<ol style="list-style-type: none"> 1. Unzip the installation package to a temporary directory. 2. From the command line, change to the temporary directory and take one of the following actions: <ul style="list-style-type: none"> • On Windows: Run install.exe. • On Linux: Run install. 3. Proceed through the Installation Manager wizard.

Results

When the userinst or install program starts, Installation Manager is installed if it is not already on your computer and automatically started. Installation Manager is configured with the location of the repository (installation files) for IBM InfoSphere Classic Data Architect V11.3.

What to do next

You can launch the product:

- From Windows: **Start > Programs > IBM Classic Data Architect > IBM InfoSphere Classic Data Architect V11.3**. Alternatively, from a command line run `<installRoot>/eclipse.exe`.
- From Linux: **Applications > IBM Classic Data Architect > IBM InfoSphere Classic Data Architect V11.3**. Alternatively, from a command line run `<installRoot>/eclipse`.

Starting IBM Installation Manager

If you start the Classic Data Architect installation from the downloadable image, IBM Installation Manager starts automatically.

About this task

When you start the installation of Classic Data Architect from the downloadable image, either by running the install or userinst program, IBM Installation Manager automatically starts. If Installation Manager is not installed, it will be installed and automatically started.

If you already installed Installation Manager, you can start it by using one of the methods in the following procedure.

Procedure

Start the Installation Manager from Windows or Linux.

Method	Description
To start the Installation Manager from Windows	Click Start > All Programs > IBM Installation Manager > IBM Installation Manager .

Method	Description
To start the Installation Manager from Linux	Click Applications > IBM Installation Manager > IBM Installation Manager or alternatively change to <i>Installation Manager directory /eclipse</i> and run IBMIM.

Uninstalling the Classic Data Architect

You can uninstall the Classic Data Architect by using the IBM Installation Manager.

Before you begin

To uninstall the IBM InfoSphere Classic Data Architect product package, you must log in to the system by using the same user account that you used to install the product package. You must close the programs that you installed by using IBM Installation Manager.

About this task

You can use the **Uninstall** option in IBM Installation Manager to uninstall the IBM InfoSphere Classic Data Architect product package from a single installation location. You can also uninstall all of the installed packages from every installation location.

To uninstall the CDA product package, complete the steps in the following procedure.

Procedure

1. Start IBM Installation Manager.
2. On the Start page, click the **Uninstall** button.
3. On the Uninstall Packages page, from the **Installation Packages** list, select IBM InfoSphere Classic Data Architect version 11.3.0, and click **Next**.
4. On the Summary page, review the list of packages that will be uninstalled. The Complete page displays after the packages are removed.
5. Click **Finish**.

Installing the ODBC/CLI and JDBC clients

Install the ODBC driver and CLI, the JDBC client, or both on Linux, UNIX, and Windows systems to connect workstation client applications to the IBM InfoSphere Classic products on the mainframe.

Before you begin

Uninstall all beta versions of Classic ODBC/CLI and JDBC.

About this task

The ODBC/CLI and JDBC clients support both 32-bit and 64-bit operating systems on most platforms. Exceptions are noted on the following list:

- AIX®
- HP-UX IA64 (64-bit only)
- Linux x86

- zLinux
 - Red Hat Enterprise Linux
 - SUSE Linux Enterprise Server
- Solaris
- Windows

The following table lists the installation programs for ODBC/CLI and JDBC clients.

Table 9. ODBC/CLI and JDBC installation programs

Installation program	Operating system
cac111ax	AIX
cac111hpie64	HP-UX IA64
cac111lx	Linux
cac111zlx	zLinux
cac111so	Solaris
cac111wn.exe	Windows

To install the ODBC/CLI and JDBC clients:

Procedure

1. Log in as a user with administrator authority (user root on UNIX and Linux).
A non-root user can also install the client programs, but can update and uninstall only that user's instance.
2. Run the installation wizard.

Restriction:

If you do not use the default installation path, ensure that the directory you choose does not contain prior versions of the ODBC/CLI client.

The default installation paths are as follows:

Linux: /opt/ibm/isclassic111
UNIX: /opt/IBM/isclassic111
Windows: C:\Program Files\IBM\ISClassic111

If you install the client programs as a non-root user on UNIX or Linux, the default installation path is \$HOME/isclassic111.

Method	Steps
From CD	<p>Windows:</p> <ol style="list-style-type: none"> 1. Insert the Classic ODBC/CLI and JDBC client CD. 2. If autorun is enabled, the wizard will start automatically. If the wizard does not start, you can launch the wizard from a command prompt. <i>CD_drive:\cac111wn.exe</i> <p>UNIX and Linux:</p> <ol style="list-style-type: none"> 1. Insert and mount the Classic ODBC/CLI and JDBC client CD. 2. Start the wizard using the installation program that corresponds to the operating system as listed in Table 9 on page 50. <i>/mounted_volume/installation_program</i>
From ibm.com [®]	<ol style="list-style-type: none"> 1. On your system, create a temporary directory with at least 150 MB of free space. 2. Download the installation program that corresponds to the operating system into the temporary directory. 3. On Linux and UNIX systems, add execute permission to the installation program. 4. Start the wizard using the installation program that corresponds to the operating system as listed in Table 9 on page 50. <ul style="list-style-type: none"> • Windows: <i>Drive:\path_to_tempdir\cac95wn.exe</i> • UNIX and Linux: <i>cd</i> <i>/path_to_tempdir/installation_program</i> <i>chmod 755 ./cac111xx ./cac111xx</i>

Results

If you need to troubleshoot an install session, log information is available:

Windows:

install_path_log\install.log

UNIX, Linux:

install_path/_log/install.log

You can run the installation program with the following optional program arguments:

- help or -? for help (UNIX, Linux only)
- r to generate a response file
- f to input response file

Installing the ODBC and JDBC drivers from the command prompt

You can install the Classic ODBC and JDBC drivers without launching the graphic interface. The two alternate installation methods are called *silent* and *console*.

About this task

The console method is useful if you want to interact with the installation program but you do not have a system capable of displaying the graphic interface (for example, if you are accessing the system remotely in a secure shell).

The silent method, which is not interactive, is useful if you are accessing the systems remotely. The silent method can use settings that you provide in a response file, and enables you to run unattended installations or installations on multiple systems that require the same settings.

In the following steps, *installation_program* refers to one of the following programs:

Scenario	Program name
Installing IBM Classic ODBC/CLI and JDBC clients on a Windows system from CD:	<i>CD_drive:\cac111wn.exe</i>
Installing IBM Classic ODBC/CLI and JDBC clients on a Windows system from a download:	<i>Hard_Drive:\path_to_tempdir\cac111wn.exe</i>
Installing IBM Classic ODBC/CLI and JDBC clients on a Linux or UNIX system from CD:	<i>/mounted_cd_volume/cac111os</i>
Installing IBM Classic ODBC/CLI and JDBC clients on a Linux or UNIX system from a download:	<i>/path_to_tempdir/cac111os</i>

To install the ODBC and JDBC drivers from the command line:

Procedure

• Console method:

1. Run the installation program with the **-console** parameter.
 - ODBC/CLI and Java™ client for Windows, UNIX, or Linux:
installation_program -i console

• Silent method:

1. Optional: Create a response file to use for the silent installation.

You can create a response file by running the installation program in GUI or console mode with a form of the **-options** switch. You can edit the created response file in a text editor.

 - Using the installation wizard to create a response file only (for use in a later installation):
installation_program -options-template full_pathname_to_responsefile
 - Using the installation wizard to create a response file and install the product locally:
installation_program -options-record full_pathname_to_responsefile
2. Run the installation program with the **-silent** parameter.
 - ODBC/CLI and Java client for Windows, UNIX, or Linux:


```
installation_program -i silent
```

3. You can run the installation program with the following optional program arguments:

- help or -? for help
- r to generate a response file
- f to input response file

For example on Windows, UNIX or Linux:

```
installation_program -i console -r <path><properties.txt>
```

Results

If you need to troubleshoot an install session, log information is available:

Windows:

```
install_path\_log\install.log
```

UNIX, Linux:

```
install_path/_log/install.log
```

Uninstalling the Classic client components

Each Classic client component comes with an uninstall program. The uninstall program for the ODBC/CLI and JDBC clients can be run in interactive mode or silently.

Uninstalling the ODBC/CLI or JDBC client

You can uninstall the ODBC/CLI client and JDBC client by using the included uninstall facility.

About this task

To uninstall the ODBC/CLI or JDBC client:

Procedure

Run the uninstall wizard:

- **Windows:** Run the uninstall program. You can run it interactively or in silent mode. (The examples depict the default installation directory.)
 - **Interactive wizard:** Open the Add/Remove Programs Control panel, select IBM InfoSphere ODBC Administrator, and click **Change/Remove**.
 - **Interactive text-based console:**

```
cd C:\Program Files\IBM\ISClassic111\  
_uninst\uninstaller.exe -is:javaconsole -console
```
 - **Non-interactive or silent:**

```
cd C:\Program Files\IBM\ISClassic111\  
_uninst\uninstaller.exe -silent
```
- **Linux and UNIX:** As root user:
 1. Change to the directory where the ODBC/CLI or JDBC client is installed.
 2. Run the uninstall program. You can run it interactively or in silent mode. (The examples depict the default installation directory for Linux.)
 - **Interactive wizard:**

```
cd/opt/IBM/isclassic111/_uninst
./_uninst/uninstaller.bin
- Interactive text-based console:
cd/opt/IBM/isclassic111_uninst
./_uninst/uninstaller.bin -console
- Non-interactive or silent:
cd/opt/IBM/isclassic111_uninst
./_uninst/uninstaller.bin -silent
```

If you installed the client programs as a non-root user, substitute the following path for `/opt/ibm/isclassic111/_uninst`:

```
$HOME/isclassic111/_uninst
```

Results

If you need to troubleshoot an uninstall session, logs are available in the following locations:

Windows:

```
installation_path\_log\uninstall.log
```

Linux and UNIX:

```
installation_path/_log/uninstall.log
```

Chapter 3. Configuring

Configuring Classic federation involves planning your configuration, configuring data servers and the services that run in them, configuring access to data sources, mapping tables, configuring communications, and configuring clients.

Overview of configuring Classic federation

The main factors that determine how to configure Classic federation are the type of data server and the type of client that you use.

The Classic federation data server is designed for continuous operation. You can add new data sources and services as your use of the Classic federation expands, without affecting existing applications. You can also perform tuning and troubleshooting without stopping a data server.

To configure clients, you define the data sources that your application uses and you can define additional tuning and debugging parameters. Client configuration is straightforward. For example, to configure the ODBC client, you use the ODBC Administrator.

To establish communication, you configure a communication service that each data source will use to communicate with a data server.

Gathering information and securing your environment

Classic federation requires initial setup tasks, such as obtaining library authorizations and port assignments and securing your Classic data servers.

Obtaining library authorizations for the APF

Classic federation requires APF authorization for the installation load library.

About this task

The Classic data servers and utility programs use z/OS facilities and services that require APF authorization. The load library data sets must be defined to z/OS as APF authorized.

Procedure

Before setting up and configuring the data servers, ensure that the installation load library SCACLOAD is APF authorized.

Obtaining ports for Classic data server communication

Classic federation uses TCP/IP to communicate between various components, which requires a port number.

Before you begin

Obtain the port assignments that you need from your network administrator to ensure that you are using dedicated ports for the exclusive use of Classic federation.

About this task

Work with your network administrator to ensure that Classic data servers have authorization to use the assigned ports.

Procedure

You must obtain a port assignment for each of the following communication channels on the source server. You must define a port number as part of the COMMSTRING configuration parameter for the INIT service on the source server. Obtain a port assignment for each of the following communication channels on the Classic data server.

1. A port for connections with the Classic Data Architect and other clients.
You define this port number as part of the COMMSTRING configuration parameter for the INIT service. 9087 is the well-known and reserved port for connections between the Classic Data Architect and other clients and a connection handler service (INIT) that runs in the source server.

Securing a Classic data server

To secure your Classic data server, create security classes and profiles in your external security manager (ESM) and configure service-level parameters in the server.

About this task

Work with your Security Administrator to determine the appropriate levels of security for your site. For best results, secure as many operations as you can while maintaining site standards and performance. You can secure administrative connections to your Classic data server, remote operator commands, monitoring, table mapping, and catalogs.

When you secure the configuration data sets, use the following table to locate them. You specified high level qualifiers for these data sets in the customization parameters file when you performed the installation customization process.

Table 10. High level qualifiers and corresponding data sets.

High level qualifier (HLQ)	Function
CACCFGNM=&CAC..CACCFGD	Configuration data sets
CACCFGIX=&CAC..CACCFGX	
CACCATNM="&USRHLQ.CATALOG	Metadata catalog data sets
CACIDXNM="&USRHLQ.CATIDX	

Procedure

1. Determine which users require access to your Classic data servers and the level of access that each user requires.
Decide whether each user requires READ or CONTROL access to administrative functions, and remote console commands.
2. Ensure that each user has a valid z/OS user account.
3. If you grant different levels of access to system resources for multiple users, follow these steps.

- a. Ask your System Administrator to define the profiles and classes that you require by using the System Authorization Facility (SAF).
Follow the remaining steps for each applicable service:
 - Operator service (OPER)
 - Query processor service (QP)
- b. Ensure that VALIDATE=Y for the **SAFEXIT** parameter.
VALIDATE=Y is the default setting, so omitting the keyword has the same effect as specifying VALIDATE=Y.
If you use SAFEXIT with VALIDATE=N then the Classic data server only validates that each user has a valid user account and password.
- c. Optional: Override the default class and profile names in the *xxxCLASS* and *xxxPROF* parameters.

Security

To implement security, work with your Security Administrator to define any required classes and profiles, and then secure your Classic data servers and catalogs.

Use these different layers to secure your Classic federation environment:

- “z/OS-level security”
- “Server security”

z/OS-level security

The System Authorization Facility (SAF) is a z/OS interface that programs use to communicate with an external security manager (ESM), such as the Resource Access Control Facility (RACF). SAF and your ESM work together to grant access rights to system resources, such as the following:

- Classic data servers
- Services
- Catalogs

RACF *classes* organize profiles into groupings of related system resources. *Profiles* define security for specific users, groups, and protected resources. Your security administrator creates classes and profiles, then grants users or groups READ or CONTROL access to the resources in the profiles.

For more information about z/OS security, see the *z/OS Security Server RACF Security Administrator’s Guide*.

Server security

The following table describes specific tasks on a Classic data server and the required SAF access that a user needs to perform them. A user with CONTROL access automatically has READ access.

Table 11. Tasks and required user access

Task	Service to configure	Minimum required access
Monitor metrics	Monitoring service	READ
Issue remote console commands	Operator service	READ

The SAFEXIT service parameter

Secure your Classic data server by using the SAFEXIT service parameter. The following services have a SAFEXIT parameter:

- Monitoring service (MAA)
- Operator service (OPER)
- Query processor service (QP)

SAFEXIT and protected resources

If you define the SAFEXIT parameter by specifying the SAFEXIT value CACSX04,VALIDATE=N the Classic data server performs user ID and password authentication only. No other user validation takes place.

Use the SAFEXIT value CACSX04,VALIDATE=Y to grant multiple users different levels of access to system resources based on classes and profiles. For each of these services, you can override the default class and profile by specifying different z/OS class or profile names as values for service parameters. Your ESM then authenticates user access by checking the specified profiles.

Table 12. Default classes and profiles per service, with override service parameters

Service name	Default class	Default profile	Override class parameter	Override profile parameter
Monitoring service*	SERVAUTH	CEC.MONITOR	MONCLASS	MONPROF
Operator service	SERVAUTH	CEC.OPER	OPRCLASS	OPRPROF
Query processor service	Not applicable	Not applicable	Not applicable	Not applicable

* See the 'Monitoring service' section below for additional information.

You can supply values for these parameters during the installation customization process or by setting them in the Console Explorer in the Classic Data Architect. For example:

```
SAFEXIT="CACSX04,VALIDATE=Y,ADMCLASS=xxxxxxx,ADMPROF=yyyy.yyyyy"
```

Monitoring service

The monitoring service secures access to metrics.

A Classic data server provides different ways of accessing monitoring information, depending on your solution. Data Replication for

The Classic Data Architect accesses monitoring information by using the same user account that connected with the Classic data server.

Query processor service

The query processor service uses the SAF exit to authenticate users who map tables and run test queries.

Operator service

The operator service authenticates users who run remote console commands on the Classic data server, including the MTO command files run from the Classic Data Architect. Console users are generally system operators who make z/OS system console requests to a Classic data server.

The operator service does not secure operator commands that you enter from a z/OS console or equivalent interface, such as the System Display and Search Facility (SDSF). When you issue commands to the console you have implied authority to issue commands to the Classic data server, so

you must secure these command interfaces to prevent unrestricted access. Ensure that users who run remote operator commands have READ access to connect the data server and to issue the DISPLAY command, and CONTROL access for all other commands.

SAF exit

You can use the SAF exit for security validations. The SAF exit verifies access for user and client connections and controls the operations that a user can perform.

The SAF exit controls access to system resources based on classes and profiles.

For detailed information about activating the SAF exit and the SAF exit API, see Security: SAF exit.

Basic configurations for Classic federation for all data sources

This section contains procedures for creating basic configurations that apply to all data sources. You can follow these procedures step-by-step or use them as aids for creating more complex configurations.

Configuring data servers

When you configure a data server, you must also configure a query processor and the appropriate communication services so that your client applications can communicate with the data server.

Before you begin

To configure a data server for Classic federation, you must first create one or more basic data servers. You do this by following the installation customization process. You can also migrate a data server from a previous version.

About this task

You can modify the global parameters of your data server in the Console Explorer in the Classic Data Architect.

The data server might need more memory, depending on how many users are concurrently connected, how many queries are running concurrently, and whether the queries contain an ORDER BY clause or other predicates that require sorting. To increase the number of concurrent users that a single data server can support, raise the value of the **MESSAGEPOOLSIZE** parameter.

The resulting data server that results from the installation customization process has all of these required services. Each service is pre-configured. You can then modify a service to fit your application as needed.

Query processors

This documentation sometimes refers to the query processor as the *data source*. The service name for the query processor must match the name of the data source that you specify when you configure the clients.

You can configure different types of query processors, depending on whether you want to use z/OS resource recovery services (RRS) to perform two-phase commit operations:

- A single-phase commit query processor (CACQP)
- A two-phase commit query processor (CACQPRRS)

Define a separate data server for each of these query processor types. They cannot coexist in the same data server.

A service definition for a single-phase query processor is in the sample member *User.SCACCONF(CACSVQP)*. A service definition for a two-phase commit query processor is in the sample member *User.SCACCONF(CACSVRRS)*.

You can add or modify query processors in the Console Explorer in the Classic Data Architect. Alternatively, you can issue the z/OS operator command ADD to create a new query processor service or the SET command to modify an existing query processor service.

Connection handlers

To set up a TCP/IP connection handler to communicate with client applications, you need the host name or IP address where the data server runs and a unique port number.

To set up cross-memory services for developing client applications on the same z/OS system, you must configure a unique dataspace name and queue name.

To configure data servers:

Procedure

- Determine the number of data servers that your applications require.
 - Customize your installation so that you have a basic data server on the LPAR.
 - Configure a query processor. The query processor is pre-configured during installation customization. These guidelines provide additional information about configuration parameter settings for the query processor.
1. The values of the following configuration parameters identify the number of query processor tasks and user connections that the query processor will service:

INITIALTHREADS

Specifies how many query processor tasks to start during data server initialization.

MAXTHREADS

Specifies the maximum number of query processor tasks to start.

MAXUSERS

Specifies the number of user connections that the query processor services.

The following example sets the **INITIALTHREADS** parameter to 6 for the query processor service MY_QP:

```
SET,CONFIG,SERVICE=MY_QP,INITIALTHREADS=6
```

2. Optional: Create service override records to contain override parameters for specific users.

The following example creates a service override record for the user TESTUSER and the query processor service MY_QP:

```
ADD,CONFIG,USER=TESTUSER,SERVICE=MY_QP
```


3. Optional: Issue SET commands against the user override record to specify override parameters for that user.

The following example sets the value of the **MAXTHREADS** parameter to 15 for the user TESTUSER:

```
SET,CONFIG,USER=TESTUSER,SERVICE=MY_QP,MAXTHREADS=15
```

4. Activate the system exits that you need for security, accounting, record processing, and managing DB2 processing.
- Configure a connection handler.
 - Start the data server.
 - Repeat these steps for each additional data server that you require.

Mapping tables for Classic federation

In the Classic Data Architect, create relational tables that map to the data source you need to access.

About this task

You create the relational tables and views in a project in Classic Data Architect. Then, you promote these objects to a data server. You can also create stored procedures, modify PCB selection for IMS tables or indexes, and define occurs processing.

Procedure

1. Configure Classic Data Architect by creating prerequisite objects, creating connections to data servers, setting preferences, importing reference files, and granting privileges. See *Configuring Classic Data Architect*.
2. Create objects for specific data sources. See *Mapping data for Classic federation topics*.

Configuring client applications

Before you can run your application, you must configure the client. Client configuration depends on the operating system and the communication protocol that you use.

About this task

All clients support TCP/IP. To configure a client to communicate with the data server, perform the configuration steps for one of the clients.

Procedure

- Configure the JDBC client. See *Establishing connections from JDBC client applications to data servers*.
- Configure the ODBC client. See *Configuring ODBC data sources*.
- Configure a CLI client depending on the operating system that you use.
 - Configure the CLI client for UNIX. See *Configuring the UNIX, Linux, and Windows CLI client*.
 - Configure the CLI client for native z/OS. See *Configuring native z/OS CLI clients*.
 - Configure the CLI client for USS. See *Configuring the USS CLI client*.

Configurations for Classic federation for specific data sources

In addition to configuring the basic components required for Classic federation, setup requirements vary by data source. For example, additional configuration is required to setup the interface to some data sources, to configure initialization services, and to customize data server JCL and configuration parameters.

About this task

The following procedure provides general guidelines for configuring specific data sources and for setup steps that are common to all data sources:

Procedure

1. Follow the configuration steps for the setup requirements that are specific to the data source that you need to access.
 - For Adabas setup, see “Setting up access to Adabas databases.”
 - For CA-Datcom, see Setting up access to CA-Datcom.
 - For CA-IDMS setup, see Setting up access to CA-IDMS.
 - For DB2 for z/OS setup, see Setting up access to DB2 for z/OS.
 - For IMS setup, see Setting up access to IMS.
 - For sequential file setup, see Setting up access to sequential files.
 - For VSAM setup, see Setting up access to VSAM.
2. Follow the configuration tasks that are common to all data sources.
 - a. Optional: Create or upgrade a metadata catalog. A default set of metadata catalogs is created and initialized during the installation customization process. You need to define a new set of metadata catalogs if you set up a second data server. See Creating and initializing sequential metadata catalogs.
 - b. Configure logging for data servers. See “Defining logger services” on page 87.
 - c. Configure a TCP/IP connection handler to communicate with the Classic Data Architect. See “Configuring TCP/IP connection handlers” on page 88.
 - d. The basic configuration steps to configure communication for clients, start the data server, and map tables occur at this stage of configuration. See “Basic configurations for Classic federation for all data sources” on page 59.
 - e. Optional: Copy the content of your sequential metadata catalog into a Version 11.3 linear metadata catalog. See Creating and initializing linear metadata catalogs.

Configuring access to Classic federation data sources

This section contains procedures for configuring access to the Classic federation data sources.

Setting up access to Adabas databases

To set up access to Adabas databases, you must configure the data server to work with the ADARUN program and point to the Adabas runtime libraries.

About this task

After you configure your data server to access Adabas, you can then use the Classic Data Architect to validate the tables and views that you map to your Adabas databases.

Classic data servers use an Adabas link routine that includes a Classic before call exit. You must run the sample JCL in *User.SCACSAMP(CACADAL)* to build the required modules. The supplied sample *User.SCACSAMP(CACADADD)* provides control card information to the ADARUN program, which Adabas uses to manage access by user programs such as the Classic data server.

If you use the installation customization process to set up your initial data server, that process configures CACADAL and CACADADD for you. If you set up the data server by a different method, configure these sample members manually before you perform the steps.

Modify the CACADAL job to select one of the following two statements, depending on whether you are using an Adabas link routine at version 8 or above. The example specifies version 8:

```
//          SET ADALNKV8=Y           ADALNK AT OR AFTER V8
// *       SET ADALNKV8=N           ADALNK PRIOR TO V8
```

Set the following parameters in CACADADD:

SVC

Set SVC to the supervisor call (SVC) number of the Adabas nucleus.

DBID

Set DBID to the default database identifier in Adabas.

LNKNAME

If you are accessing Adabas by using a version 8 or higher Adabas link routine, set LNKNAME=CACAGBL to indicate that the data server requires the Classic version (CACAGBL) of the Adabas link globals module.

PROG

If you are accessing Adabas by using a version 8 or higher Adabas link routine, set PROG=RENTUSER to indicate that the data server requires the reentrant version of the Adabas link routine.

You configure access to Adabas in the JCL that runs your data server. You can find sample JCL in the sample member *User.SCACSAMP(CACCDS)*.

Procedure

1. Run the sample JCL in *User.SCACSAMP(CACADAL)*.
2. Add the Adabas runtime libraries to the STEPLIB concatenation.
3. Point to the sample *User.SCACSAMP(CACADADD)* in the DD CARD DD statement.

Examples

The first example shows a STEPLIB concatenation that points to the Adabas runtime libraries:

```
//STEPLIB DD DISP=SHR,DSN=&CAC..V11R3M00.SCACLOAD
// DD DISP=SHR,DSN=&ADA..LOAD
```

The next example shows how to reference CACADADD in the DD statement.

```
//DDCARD DD DISP=SHR,DSN=&CAC..SCACSAMP(CACADADD)
```

Setting up access to CA-Datcom

You need to configure the data server to enable access to CA-Datcom data. Optionally, you can also set up security.

Procedure

1. Define the CA-Datcom access service for the data server. See “Access service for CA-Datcom.”
2. Ensure that the CA-Datcom Multi-User Facility is authorized. See “Multi-User Facility authorization and CA-Datcom” on page 70.
3. Optional: Set up security for CA-Datcom access. See “CA-Datcom security” on page 70.
4. Configure the data server for validation. See “CA-Datcom setup for dynamic discovery from the Classic Data Architect” on page 71.

Access service for CA-Datcom

The Datcom access service creates and manages connections to CA-Datcom databases.

To access CA-Datcom databases, you must add and configure the Datcom access service.

Defining task areas:

To access CA-Datcom databases, you must first configure the Datcom access service.

About this task

Set these parameters to define connections with CA-Datcom databases:

URTPOOL

Specifies whether the Datcom access service manages a pool of user requirements tables (URTs) or manages each user requirements table separately. The possible values are TRUE or FALSE, and the default value is FALSE.

TASKAREASACQ

Specifies the maximum number of task areas that the Datcom access service can acquire, initialize, and maintain for the use of the query processor. The value of this parameter determines the number of concurrent connections with CA-Datcom.

Possible values range from 1 to 200.

TASKAREASRES

Specifies the number of task areas to reserve for the use of modification queries, which perform insert, update, or delete operations. The task areas that remain can perform modification queries or read-only queries.

You can change the configuration of the CA-Datcom access service in the Classic Data Architect or by issuing z/OS operator commands. The following example shows how to change the configuration by using the z/OS operator command SET,CONFIG:

```
F Job-Name,SET,CONFIG,SERVICE=Service-Name,Parameter-Name=Value
```

In the following example, you set the value of **URTPool** to TRUE for a Datcom access service with the service name DCOMIS. The job name of the data server is DS2.

```
F DS2,SET,CONFIG,SERVICE=DCOMIS,URTPool=TRUE
```

Restart the Datcom access service to put configuration changes into effect. The data server does not recognize configuration changes that you make while the Datcom access service is running.

For information about connecting to CA-Datcom databases and how these parameters interact, see the topic “Database access to CA-Datcom”.

Procedure

1. Set a value for **URTPool**.
2. Set a value for **TASKAREASACQ**.
3. Set a value for **TASKAREASRES**.

Database access to CA-Datcom:

The query processor and the Datcom access service manage access to CA-Datcom data for a Classic data server. Pre-allocated *task areas* provide the actual database connections.

The query processor converts SQL queries into native CA-Datcom commands. You can join the results of these queries with queries against other data sources, such as VSAM. To set up access to CA-Datcom databases, configure the Datcom access service to conform with site requirements:

- Whether to use *pooled* or *non-pooled* user requirements tables
- The number of connections with CA-Datcom

Restriction: The query processor services within in a single data server must access the same Multi-User Facility in CA-Datcom. Accessing a different Multi-User Facility requires a second data server that references a different CA-Datcom control library.

Pooled and non-pooled user requirements tables

Consider using pooled user requirements tables to improve the performance and throughput of your environment.

Pooled user requirements tables

Choose this option to reduce the number of times that the Datcom access service opens and closes user requirements tables. On completion of a query, pooled user requirements tables remain open for the duration of the related instance of the Datcom access service. All queries and stored procedures that reference a given user requirements table share the same instance of that table.

If you deploy pooled tables, the Datacom access service launches multiple task areas on a single, multi-threaded connection. Each concurrent transaction uses a separate thread as though it were an independent connection, while the Multi-User Facility in CA-Datacom recognizes only one connection.

Pooled user requirements tables have the following advantages:

- Reduced processing overhead related to repeated cycles of opening and closing user requirements tables

Alternatively, the CA-Datacom start up option **OPTIMIZE** can mitigate this issue.

- Reduced message traffic on the console related to opening and closing user requirements tables

Alternatively, you can disable the display of messages DB00101I and DB00102I at the CA-Datacom Multi-User Facility level.

Non-pooled user requirements tables

In a non-pooled environment, the Datacom access service closes the user requirements table when each transaction finishes processing. This approach launches an independent connection for each task area instead of a single, multi-threaded connection for all task areas. Transactions that reference the same user requirements table can share the same task area.

The **URTPOOL** parameter determines whether you use pooled or non-pooled user requirements tables:

URTPOOL

Specifies whether the Datacom access service manages a pool of user requirements tables or manages each user requirements table separately. The default value is **FALSE**.

TRUE

The Datacom access service opens a user requirements table the first time a query or stored procedure references it. The user requirements table remains open and until one of the following events occurs:

- You issue a **CLOSE** command against the user requirements table
- The Datacom access service stops
- The Classic data server stops

FALSE

The Datacom access service closes a user requirements table when the query or stored procedure that references it finishes processing.

Connections to CA-Datacom

You can minimize resource contention and improve overall throughput by specifying the maximum number of concurrent connections in the service definition for the Datacom access service. Other components can also impose limits on the number of connections:

- The query processor in the Classic data server
- The Multi-User Facility (MUF) in CA-Datacom

The way you configure these components depends on the impact on the Multi-User Facility and the databases. Work with your Datacom administrator to determine the number of concurrent connections that is appropriate for your site.

The query processor, the Datacom access service, or the Multi-User Facility can reject connection requests that exceed the maximum number of available task areas and issue an error message.

Query processors and concurrency management

Assuming that the settings in the query processor and the Multi-User Facility are not a limiting factor, the value that you specify for the **TASKAREASACQ** parameter determines the maximum number of concurrent connections that the data server can open with the Multi-User Facility.

To ensure that your query processor configurations are not a limiting factor, calculate the maximum number of concurrent queries in the service definition for the query processor:

Maximum-Number-Of-Queries = MAXTHREADS * MAXUSERS

The **MAXTHREADS** parameter represents the maximum number of query processor instances per service. The **MAXUSERS** parameter represents the maximum number of user connections to the query processor.

Workload analysis and concurrency management

A task area processes both read-only queries and modification queries. Read-only queries generate minimal contention because they require access to task areas for very short periods of time. Because they have less impact on the database, read-only queries that use the same user requirements table can share a single task area. Modification queries, which perform inserts, updates, and deletes, require exclusive control of a task area for the full duration of the query, and therefore cannot share a connection.

After you identify and analyze the mix of read-only queries and modification queries that you expect, specify a number of task areas that supports that workload and avoids resource contention.

The following configuration parameters for the Datacom access service determine the number of connections and their purpose:

TASKAREASACQ

Specifies the maximum number of task areas to acquire for accessing CA-Datacom. The behavior of this setting differs slightly, depending on the setting of **URTPOOL**:

URTPOOL=TRUE

The Datacom access service opens one connection to CA-Datacom to manage processing. **TASKAREASACQ** specifies the maximum number of threads that the Datacom access service uses on this single connection.

URTPOOL=FALSE

The Datacom access service opens a separate connection to CA-Datacom for each concurrent transaction, up to the maximum number of connections that you specify.

TASKAREASRES

Specifies the number of task areas to reserve for modification queries. Reserving some of the tasks areas for update queries reduces contention for task areas between read-only and modification queries.

If all reserved connections are actively processing modification queries, the Datacom access service can then assign modification queries to unreserved connections. Read-only queries use only the unreserved connections.

RRS processing

The Datacom access service treats an RRS-enabled query the same way as a modification query, in that RRS processing requires the exclusive use of a connection and cannot share it. The service closes each user requirements table on completion of an RRS-enabled query, so setting **URTPPOOL** to TRUE has no effect on RRS processing.

For more information about configuring the Datacom access service, see the topic "Defining task areas".

Examples

The examples are valid for any type of configuration except RRS. The examples assume that the query processor is not a limiting factor on the number of connections. In each example, **TASKAREASACQ=5** and **TASKAREASRES=2**.

Example 1: Read-only queries reference different user requirements tables

The maximum number of concurrent connections to CA-Datacom is three, because read-only queries that reference different user requirements tables cannot share the same task area. In this scenario, the Datacom access service does not use the two tasks that you reserve for update queries.

Example 2: Read-only queries reference the same user requirements table

The maximum number of concurrent connections to CA-Datacom is $3 * 32767$. In this scenario, the maximum number of tasks that you define in the Multi-User Facility determines the upper limit on the concurrent connections.

Read-only queries that reference the same user requirements table can share the same task area. In this scenario, the Datacom access service does not use the two tasks that you reserve for update queries.

Example 3: Modification queries that reference different user requirements tables

The maximum number of concurrent connections to CA-Datacom is five. Modification queries cannot share the same task area.

Example 4: Modification queries that reference the same user requirements table

The maximum number of concurrent connections to CA-Datacom is five.

Modification queries cannot share tasks areas, even when they reference the same user requirements table. The Datacom access service can use all task areas for modification queries, but uses the reserved tasks areas first if they are available.

Example 5: Three modification queries are running

The Datacom access service uses the two reserved tasks plus one unreserved task, leaving two task areas free for additional queries.

The maximum number of possible read-only queries depends on the user requirements tables that the queries reference.

- Possible number of read-only queries that reference different user requirements tables: 2
- Possible number of read-only queries that reference the same user requirements table: 2 * 32767

Opening and closing user requirements tables for database maintenance:

Before a CA-Datacom administrator takes a databases offline for maintenance, you might have to close user requirements tables that the Datacom access service opened.

About this task

You cannot take a database offline from a CA-Datacom Multi-User Facility (MUF) if applications, such as a Classic data server, have pooled user requirements tables (URTs) opened against the database. You must close pooled user requirements tables because they remain open after your queries run.

Use z/OS operator commands to close the required user requirements tables or to stop the Datacom access service. After maintenance is complete, you can reopen the user requirements tables.

A user requirements table can have any of the following statuses:

OPEN

The user requirements table is physically open.

CLOSED

The user requirements table is physically closed. You cannot access the closed user requirements table with new SQL statements until you restart the Datacom access service or issue an OPEN command against the user requirements table.

CLOSING

The user requirements table is in the process of closing, and physically closes when all SQL statements that reference it finish processing. When processing completes, the Use Count is zero and you can no longer access the user requirements table.

READY

The user requirements table is ready to open. You issued an OPEN command against the user requirements table, and no SQL statements physically opened the table yet.

Perform the steps in the following procedure to open and close user requirements tables for database maintenance:

Procedure

1. To display information about user requirements tables that the Datacom access service processed, issue the DISPLAY,URT command.

```
/f Job-Name,CMD,Service-Name,'DISPLAY,URT=[URT-Name|ALL]'
```

Where *Job-Name* is the name of the job or started task for the data server, *Service-Name* is the name of the Datacom access service, and *URT-Name* is the name of the user requirements table. The job name is CACDS for Classic federation.

Specify the ALL option to display all user requirements tables that this instance of the Datacom access service processed.

The following example describes the command output:

```
CACK009I DATACOM ACCESS SERVICE URT ACCESSED LIST
***** URT ACCESSED LIST *****
URT Name      Status      Use Count
-----
URT1          OPEN        0
URT2          CLOSED     0
```

2. Close the required user requirements tables.

- a. To close a user requirements table that you specify, issue the CLOSE,URT command:

```
/f Job-Name,CMD,Service-Name,'CLOSE,URT=URT-Name'
```

The following example describes the command output:

```
CACK010I URT CLOSE INITIATED FOR URT 'URT1' USE COUNT (0)
CAC00200I CMD,MYDCI,"CLOSE,URT=URT1"
CACK011I URT CLOSE COMPLETED FOR URT 'URT1'
* URT MUST BE RE-OPENED TO ENABLE ACCESS
```

The command takes effect when all outstanding SQL statements that access the user requirements table finish processing.

- b. To close all pooled user requirements tables, stop the Datacom access service by issuing the STOP,SERVICE command. :

```
F Job-Name,STOP,SERVICE=Service-Name
```

3. If shutdown operations are timing out, raise the value of the **RESPONSETIMEOUT** parameter against the controller service to enable you to close a larger number of pooled user requirements tables without exceeding the timeout limit.

```
SET,CONFIG,SERVICE=CNTRL,RESPONSETIMEOUT=Time
```

Express the value of *Time* as *nMS* (*n* milliseconds), *nS* (*n* seconds), or *nM* (*n* minutes), for example, 2S.

4. To open a user requirements table that you specify, issue the OPEN,URT command.

```
/f Job-Name,CMD,Service-Name,'OPEN,URT=URT-Name'
```

The following sample describes the command output:

```
CACK012I URT OPEN COMPLETED FOR URT 'URT1'
```

The physical open of the user requirements table actually occurs on the next SQL statement that references the user requirements table.

Multi-User Facility authorization and CA-Datacom

The CA-Datacom load library requires authorization because certain Multi-User Facility options are operational only when the Multi-User Facility is authorized.

If authorization is not present when the Multi-User Facility is started, the Multi-User Facility issues the following message: DB00210I - MULTI-USER NOT RUNNING AUTHORIZED.

CA-Datacom security

Your choices for security include internal CA-Datacom security (less secure), external security, or no security at all.

About this task

Data servers run as either a z/OS-started task or a batch job. The primary authorization ID is the user of the started task or batch job. In many installations, you specify the user for started tasks as plus signs ('+++++++'). Because the data server is a multiuser system, access to CA-Datcom under the authorization ID of the data server does not provide adequate security for your data.

For added security, use an external security package, such as RACF, ACF-2, or Top Secret, where each user must have a unique identity. During the connection process, the security system at your site verifies the user ID and password that you supply when you connect to the data server. The user ID in the accessor environment element (ACEE) authorizes database processing.

Restriction: All query processors in the data server must have the same SAFEXIT configuration. Deploying multiple query processors that have different configuration settings can cause the Datacom access service to reject your queries.

Procedure

- Specify SAFEXIT=CACSX04 in the service definition for all query processor services in the dataserver.
- Optional: Enable user-level security checking for each database request.
- Choose one of the following options, depending on whether you enable security checking for each request:

Option	Description
Check security for each database request	The accessor environment element that the SAF exit creates must be available outside the exit. Request a Computer Associates software solution and install that solution on your CA-Datcom system. Contact CA-Datcom Technical Support and request one of the following solutions: <ul style="list-style-type: none">• Solution TB10727 (CA-Datcom Version 9.0 only)• Solution TB20202 (CA-Datcom Version 10.0 only)• Solution TB31027 (CA-Datcom Version 11.0 only)
Do not check security for each database request	If the SAF exit is active in the data server, you must configure the SAF exit not to send an accessor environment element. Set the value of the EXCLUDE parameter to 2 in the SAF exit configuration file entry.

CA-Datcom setup for dynamic discovery from the Classic Data Architect

When a CREATE TABLE or CREATE INDEX statement that references a CA-Datcom table is executed, the CA-DATACOM Datadictionary Service Facility (DSF) is invoked to perform validation processing.

Procedure

- Ensure that the STEPLIB DD statement of the link-pack area provides access to the DSF load modules and the DDSRTLM (CA-Datacom System Resource Table). The required modules are located in the following CA-Datacom load libraries:

- CA1.CAILIB
- CA1.DATACOM.CHLQ.CUSLIB
- CA1.DATACOM.THLQ.CAILIB

Replace CHLQ and THLQ with the applicable high-level qualifier.

- Update the data server JCL to include a DDIDENT DD statement that references a text-data set or PDS member that contains DSF user ID and password information. A sample DDIDENT PDS member is located in SCACSAMP member CACDCID. The file referenced by the DDIDENT DD statement contains parameters that identify the user ID and password that allow access to DSF:

USER

Identifies an ENTITY-OCCURRENCE-NAME of a PERSON that is in production status. If no USER information is supplied, the DSF assumes there is no authorization.

PASSWORD

Identifies a QUALIFIER attribute that specifies the password for the PERSON that is identified in the USER parameter.

These access-information parameters are entered free form, in a standard-text file. You can enter the parameters on the same line, separated by a comma, or enter each parameter on a separate line. If a single parameter is entered more than once, the last occurrence of that parameter is used when accessing the Datadictionary Service Facility. The number of leading and trailing blanks or the number of spaces before and after the equal sign, is inconsequential. As with any file containing sensitive information, this access information file must be secured according to site specifications.

- Ensure that the CA-Datacom access service is active in the data server.
- Ensure that the URT load module that is identified on the ACCESS USING URT clause on the CREATE TABLE statement exists and is loadable by the data server.

Setting up access to CA-IDMS

You need to configure the data server to communicate with one or more CA-IDMS central versions to enable access to CA-IDMS data and to process CREATE TABLE and CREATE INDEX statements.

About this task

The CA-IDMS access component runs under the query processor in the data server. This component supports multiple users and does not require initialization services.

The batch access module, IDMS, provides access to CA-IDMS. This module can access CA-IDMS data in either central version or local mode. By default, all JCL and examples are configured to access CA-IDMS in central-version mode. If you have a specific need for local-mode access, see the CA-IDMS documentation about the necessary JCL changes to allocate and access CA-IDMS databases.

Procedure

1. Verify that the CA-IDMS MAXERUS setting is sufficient to support the data server access to CA-IDMS. See “Setting the maximum number of run units.”
2. Set up APF authorization of the CA-IDMS LOADLIB. See “Setting up APF authorization.”
3. Pass the correct user context to CA-IDMS run units. See “Setting up security for CA-IDMS access” on page 74.
4. Set up access to multiple CA-IDMS central versions. See “Accessing multiple CA-IDMS central versions from a single data server” on page 74.
5. Update the data server JCL to access any CA-IDMS central versions. See “Configuring the data server to access CA-IDMS central version” on page 75.
6. Update the data server JCL to be notified correctly when the CA-IDMS central version is not available. See “Configuring the data server for notification of central version shutdown” on page 76.

Setting the maximum number of run units

In central version mode, the CA-IDMS access component in the data server connects to the CA-IDMS system as an external run unit. The number of external run units available in a single CA-IDMS central-version region is a CA-IDMS configurable value.

An external run unit is established when the client issues an SQL OPEN to access a CA-IDMS-mapped table. An external run unit is also started when processing a CREATE TABLE or CREATE INDEX statement that references a CA-IDMS mapped table. The run unit is used to extract schema and subschema information from the CA-IDMS dictionary which is used to validate the CREATE TABLE and CREATE INDEX statements.

Each central version has a MAXERUS (maximum external run-units) value that limits the number of concurrent external connections. This value governs all external connection sources, such as batch jobs and CICS.

Setup and configuration of the data server requires the analysis of the MAXERUS value for CA-IDMS central versions accessed. Generally, when the data server accesses and updates CA-IDMS data, the number of query processor services running in a data server is the maximum number of concurrent run units that are active to CA-IDMS at any point in time.

Set the MAXERUS setting high enough to accommodate additional run-units that are opened due to the following data server processing:

- CREATE TABLE processing
- CREATE INDEX processing
- Select processing
- Update processing
- Delete processing
- Insert processing
- Join processing: each table referenced in the join has a separate run unit.

Setting up APF authorization

Cross-memory services and the SAF security exit require the STEPLIB for the data server to be APF authorized.

About this task

The CA-IDMS.LOADLIB is not usually APF authorized, and some utility programs in that library fail if they are run from an APF-authorized STEPLIB concatenation.

Procedure

1. Create a separate, authorized copy of the CA-IDMS.LOADLIB.
2. Create a separate, authorized copy of the CA-IDMS.DBA.LOADLIB.

Setting up security for CA-IDMS access

To establish security for CA-IDMS data, the SAF exit ensures that the user ID and password combination from the client is an authorized user of the z/OS system. In addition you need to ensure that the client is passing the correct user context in all run units that are established with the CA-IDMS central version.

The SAF system exit automatically validates all user IDs and passwords when a user establishes a connection to the data server. The SAF exit also validates that the user has read access rights to the catalog when catalog queries are issued.

Establishing the correct user context for CA-IDMS run units:

You must establish the correct user name for each run unit.

About this task

By default, all batch run units connect with a blank user context. Thus, in CA-IDMS, there is PUBLIC access to the data. To establish the correct user name for each run unit that is created under the data server, the CA-IDMS.LOADLIB module IDMSSTRT must be re-linked with a USRIDXIT module. Only the data server uses this IDMSSTRT module.

Procedure

1. Assemble the CACIDUXS exit source in the SAMPLIB using the CACIDUXT sample JCL in the SAMPLIB and link-edit the module into a new IDMSSTRT module.
2. Link the new module into a library that is only used by the data server. Do not place the module in the CA-IDMS.LOADLIB. If a separate APF-authorized LOADLIB exists, you can locate the new IDMSSTRT module in that library.
3. Include this library (linked to in the previous step) in the STEPLIB concatenation of the startup JCL for the data server before the CA-IDMS.LOADLIB.

What to do next

The creation of an IDMSSTRT module only ensures that all run units that are established with a CA-IDMS central version have the correct user name associated with them. The security of the CA-IDMS data is not ensured. For data to be secure, security enforcement must be active for the central version .

Accessing multiple CA-IDMS central versions from a single data server

You can access multiple central versions from a single data server.

About this task

The SYSCTL data set that is allocated to the data server identifies the default CA-IDMS central version for communication.

You can allocate multiple SYSCTL data sets with unique **DD** names and select the data sets on a table-by-table basis. Use the custom built CA-IDMS **ACCESS LOADMODs** that reference the appropriate **SYSCTL DD** name.

Perform the steps in the following procedure to build and reference a custom-access load module with the name, SYSCTL1:

Procedure

1. Code an assembler CA-IDMSOPTI module that contains the following assembler macro statement:

```
IDMSOPTI CENTRAL=YES,SYSCTL=SYSCTL1  
END
```
2. Assemble and link the CA-IDMSOPTI module using the supplied SAMPLIB member CACIDACM with the following link-edit control cards:
 - INCLUDE IDMSLOAD(IDMS)
 - ENTRY IDMS
 - NAME *new-module-name*(R)

Create a new name for the CA-IDMS module, because the default module must be left as-is for other CA-IDMS batch applications. Also, link the new module into a library that is accessible to the data server startup JCL.

What to do next

CA-IDMSOPTI modules can also manage default databases and other CA-IDMS specific parameters.

Configuring the data server to access CA-IDMS central version

You must make changes to the startup JCL for the data server to access a CA-IDMS central version.

Procedure

1. Add the authorized versions of the CA-IDMS.LOADLIB and the CA-IDMS.DBA.LOADLIB to the STEPLIB concatenation.
2. Add a **SYSCTL DD** statement for all central versions to which the data server will connect.
3. If you do not specify the CA-IDMS database to access with the DBNAME parameter on the CREATE TABLE statement, you need to set up a default dbnames mapping in the CA-IDMS dbnames table used by the central version. For example, the following dbname mapping specifies the DBNAME userdb as the default DBNAME in the dbnames table:

```
subschema ???????? maps to ???????? dbname userdb;
```

Example

The CA-IDMS.LOADLIB is a non-APF authorized library that removes authorization from STEPLIB if allocated to STEPLIB. If the data server is using any authorized services (cross-memory services and any SAF exits such as CACSX04 and CACSX07), S047 errors occur.

Configuring the data server for notification of central version shutdown

You must make changes to the data server JCL to ensure that the data server is notified properly when the CA-IDMS central version is not available.

Procedure

1. Include a SYSIDMS DD statement with the following parameters to ensure that the data server is notified correctly when the CA-IDMS central version is not available:

```
CVRETRY=OFF
REREAD_SYSCTL=ON
DMCL=<Specify DMCL name that contains the SYSTEM segment>
```

2. Include the data set for the DDLDCMSG area in the JCL for the batch job.
3. Ensure that the DMCL load module specified in the SYSIDMS parameter is available in the STEPLIB for the data server.

Setting up access to DB2 for z/OS

You need to configure the data server and configure the CAF initialization service or the RRS interface to enable access to DB2 for z/OS.

About this task

The setup requirements for access to DB2 for z/OS data also apply to validation setup for processing CREATE TABLE and CREATE INDEX statements that reference a DB2 data source.

To set up access to DB2 for z/OS data, complete the steps in the following procedure for each DB2 subsystem that you need to access:

Procedure

1. Bind an application plan. Before you access data in DB2 for z/OS, you must bind an application plan for the CAF service. Running the BIND job requires BINDADD authority.
 - a. Edit DB2DSN=DSN and DB2PLAN=CAC10PLN in the customization parameters file SCACSAMP (CACCUSP2) to specify values for the DB2 subsystem ID and plan name. Change DSN to the appropriate DB2 subsystem ID for your site. If your site requires plan names to match specific standards, you can change the plan name from CAC10PLN to a name that fits those standards.
 - b. Optional: Edit the bind control card CACBCNTL if you need to connect to additional DSNs, and bind a separate plan for each subsystem.
 - c. Run the bind job CACBIND. If you specified valid DB2 values in the customization parameters file, you do not need to edit the bind job CACBIND. This job uses the provided DBRM (MSLP2EC) to create a plan CAC10PLN (or the name that you specify in the customization parameters file).
 - d. Submit the CACBIND job.
 - e. Grant EXECUTE authority on the plan to the user ID under which the data server is running, and to individual users of Classic federation if the DB2 thread management exit is active.
2. Configure the Call Attachment Facility (CAF) initialization service or the Resource Recovery Services (RRS) interface.

- To access and update DB2 for z/OS data by using the CAF initialization service, import the CAF service definition. If you provided DB2 information in the customization parameters file, you can import the SCACCONF(CACSVCAF) member to define the CAF service with the MTO IMPORT command:

```
IMPORT,CONFIG,FILENAME=DSN:SCACCONF(CACSVCAF)
```
- To use the RRS interface for two-phase commit processing, see Two-phase commit.

Setting up access to IMS

You need to configure and enable an IMS interface to access IMS data.

Procedure

1. Decide which IMS interface to use for accessing IMS data. See “Overview of data server setup for IMS access.”
2. Configure an IMS interface.
 - To configure the DRA or ODBA interface, see “Setting up the DRA and ODBA” on page 78.
 - To configure the BMP or DBB interface, see “Setting up a BMP or DBB interface” on page 82.
3. Configure the data server for validation. See “IMS setup for dynamic discovery from the Classic Data Architect” on page 83.
4. Run the data server. See “Running the data server” on page 83.

Overview of data server setup for IMS access

Three interfaces are available for accessing IMS data. The setup requirements and required configuration steps differ for each interface. Understanding the differences enables you to decide which interface to use. Each interface has its own benefits and drawbacks, depending on your requirements for distributed transactions.

The data server supports the following interfaces that you can use to access and update IMS data:

- **The DRA interface**

The database resource adapter (DRA) interface is similar to the interface that CICS uses. The DRA is the interface to use when your client applications do not run distributed transactions with two-phase commit. For the DRA interface, you must configure and activate an IMS DRA initialization service to access your IMS data. The data server interfaces directly with IMS when you use the DRA interface.

- **The ODBA interface**

You must use the open database access (ODBA) interface when your client applications run distributed transactions with two-phase commit protocols. The ODBA interface must be used in conjunction with the two-phase commit query processor (CACQPRRS), and you need to configure and activate an IMS ODBA initialization service.

The ODBA interface uses the DRA interface to connect to IMS.

- **The BMP and DBB programming interfaces**

The BMP/DBB programming interface is a non-scalable interface that is primarily provided for initial development and testing. With this interface, a single PSB is used, and all access is serialized through that PSB. With the BMP/DBB interface, you can access IMS data in a BMP or a DBB environment. In these environments, you must configure and activate an IMS BMP or DBB

initialization service to access your IMS data. When you use any of these interfaces to access IMS data, the data server JCL requires changes.

There are setup requirements for accessing IMS data by using the DRA or ODBA interfaces.

DBCTL

DBCTL is an IMS database manager subsystem that runs in its own address space. DBCTL must be configured and running to use the IMS DRA or ODBA interface.

- DRA is the interface between the data server and DBCTL or between ODBA and DBCTL.
- ODBA is RRS-enabled to interface to IMS.

You can use an IMS DB/DC subsystem to access and update IMS data. A DB/DC subsystem is a superset of DBCTL. The environment includes the elements that the data server needs to access your IMS data by using either a DRA or ODBA interface.

If DB/DC or DBCTL is installed at your site, you need to perform the customization steps required to set up and configure the DRA and ODBA interfaces to access and update your IMS data.

Differences between DRA and ODBA

You can only use a single IMS interface (DRA, ODBA) within a data server address space.

The key differences between using the DRA or ODBA interfaces to access IMS data are described in described in the following table:

Table 13. DRA and ODBA interface differences

DRA	ODBA
In an update transaction, you can schedule only a single PSB. Scheduling occurs only if the transaction does not access any other data source.	The data server can schedule and update multiple PSBs as part of a single RRS transaction and changes to other data sources that are RRS-enabled.
A transaction can access a single IMS subsystem only.	Multiple PSBs can be scheduled that access different IMS subsystems located on the single image (LPAR) where the data server resides.
	The ODBA interface provides greater flexibility in PSB scheduling options
	The ODBA interface should only be used in conjunction with distributed transactions

Setting up the DRA and ODBA

To enable the DRA and ODBA, you need to make the DRA startup/router routine accessible to the data server.

Procedure

1. Copy the DRA startup/router routine (DFSPRRC0) into the SCACLOAD load library. Follow the instructions for defining the IMS STEPLIB in the customization parameters file. Use either of the following methods:

- Copy the routine from the IMS.SDFSRESL library (built by the IMS definition process).
 - Concatenate the IMS.SDFSRESL library to the ODBA STEPLIB.
2. Put the DRA startup table (DFSPZPxx load module) into a load library.

Setting up the DRA interface:

The default load module, DFSPZP00, is in the IMS.SDFSRESL library. The remainder of the DRA modules reside in a load library that is dynamically allocated by DFSPRRC0 (the startup).

The default DDNAME and DSNAME of this load library (IMS.SDFSRESL) are specified in the default startup table DFSPZP00. DFSPZP00 contains default values for the DRA initialization parameters. You can also create a new startup table to specify values other than the defaults.

Configuring a new startup table:

You can customize the DRA configuration by creating a new startup table.

Procedure

1. Code a new start-up table. Name it, for example, DFSPZP01. You might want to use DFSPZP00 as an example.
2. Specify the required values.
3. Copy any unchanged values from the default table.
4. Assemble and link the new module into a load library. If the new module is named DFSPZP01, the IMS DRA initialization, DRATABLESUFFIX configuration parameter, specifies a value of 01 in the DRA startup-table suffix.

What to do next

For information about creating a DRA startup table using the DFSPRP macro see the IMS documentation about the database resource adapter. For information about using the DRA start-up table to control performance, see the CICS documentation about performance.

Configuring the DRA interface:

You must configure the data server to access IMS data by using the DRA interface.

Before you begin

You must configure the IMS DRA service in the data server before you can access IMS data. Only one IMS service can be active in the data server.

About this task

Sample JCL, configuration files, and sample application programs are located in the SCACSAMP and SCACCONF libraries.

If you receive the message '-4902 The memory limit was exceeded' while using the IMS DRA interface to connect with IMS, increase the value of the global configuration parameter MESSAGEPOOLSIZE or the REGION parameter in the data server JCL.

Procedure

1. Provide IMS configuration information in the IMS section of the customization parameters file.
 - DRA user ID: Modify IMSDRAUS=DRAUSER to specify the default DRA user ID that is used to connect to and register with DBCTL. The DRAUSERID is the name by which the data server is known to DBCTL.
 - Default PSB name: Modify IMSDFPSB=DEFPSB to specify the name of a PSB. This PSB name is used when an IMS table is referenced by a CREATE TABLE statement that contains no PSB name.
 - DRA startup table suffix: If you want to use the default suffix, you can use the generated SCACCONF(CACSVIMA) member. Otherwise, edit the SCACCONF(CACSVIMA) member to specify this value. Specify the last two characters of the load module name that you created. For the default DRA startup table load module, specify 00.
2. Import the SCACCONF(CACSVIMA) member to define the DRA service. Issue the following MTO IMPORT command:

```
IMPORT,CONFIG,FILENAME=DSN:SCACCONF(CACSVIMA)
```

You can then specify the configuration parameters DRAUSERID, DEFAULTPSBNAME, and DRATABLESUFFIX on the DRA service definition by using the Classic Data Architect.

Setting up the ODBA interface:

To use the ODBA interface, make additional modules accessible. Also, you must set up startup router tables.

Procedure

1. Make the following modules accessible:

DFSCDLI0

The data server loads this module. DFSCDLI0 also contains the ALIAS name AERTDLI.

DFSAERG0

DFSCDLI0 loads this module.

DFSAERM0

DFSAERG0 attaches this module in the data server address space.

DFSAERA0

DFSAERM0 attaches this module for initialization to the specified IMS DB subsystem.

AERTDLI

The data server loads this module DL/I calls by using the ODBA interface.

2. Follow the procedures described in "Setting up the DRA interface" on page 79 for the DRA startup router table to make these modules available to the data server.
3. Name the DRA startup router tables that the ODBA interface uses with this format, DFSxxx0. The variable xxx is replaced with the name of the IMS subsystem identifier of the IMS subsystem that you connect to.

4. Create a DRA startup router table for each unique IMS subsystem. The subsystems are identified on the ODBA initialization Service Info Entry, or the SUBSYSTEM parameter in the CREATE TABLE statement for the IMS tables that you are accessing.

Configuring the ODBA interface:

You must configure the data server and activate the ODBA initialization service to access IMS data by using the ODBA interface.

Before you begin

You must configure the IMS DRA service in the data server before you can access IMS data. Only one IMS service can be active.

About this task

Sample JCL, configuration files, and sample application programs are located in the SCACSAMP and SCACCONF libraries.

Procedure

1. Provide IMS configuration information in the IMS section of the customization parameters file.
 - Default PSB name: Modify IMSDFPSB=DEFPSB to specify the name of a PSB. This PSB name is used when an IMS table is referenced by a CREATE TABLE statement that contains no PSB name.
 - Default subsystem ID: Modify IMSSSID=SSID to specify the IMS subsystem identifier to access,
2. Import the SCACCONF(CACSVIMO) member to define the ODBA service. Issue the following MTO IMPORT command:

```
IMPORT ,CONFIG,FILENAME=DSN:SCACCONF(CACSVIMO)
```

You can then specify the configuration parameters PSBPARM and SSNPARM on the ODBA service definition by using the Classic Data Architect.

Customizing JCL for DRA and ODBA:

You can customize the data server JCL to access IMS data by using the DRA interface.

Procedure

Provide IMS configuration information in the IMS section of the customization parameters file.

- High-level qualifier: Modify IMSINHLQ=IMS to specify the high-level qualifier for IMS data sets.
- SYSOUT class SOUTD: If you want to use the default suffix, you can use the generated SCACCONF(CACSVIMA) member. Otherwise, edit the SCACSAMP member CACDS.member to specify this value.

The SCACSAMP (CACDS) member JCL will be updated based on the customization parameters file. Otherwise, edit the SCACSAMP member CACDS.member to specify any additional values not included in the customization parameters file. Specify SYSOUT class SOUT at the beginning of the in-stream procedure.

Setting up a BMP or DBB interface

To use the BMP and DBB interface to access IMS data, you must perform setup tasks.

Procedure

1. Configure the data server and activate the BMP/DBB initialization service. See “Configuring the data server for a BMP or DBB interface.”
2. Modify the BMP and DBB data server JCL. See “Modifying BMP and DBB JCL.”

Configuring the data server for a BMP or DBB interface:

You can configure the data server to access IMS data by using either a BMP or DBB interface. You must configure the IMS BMP or DBB initialization service in the data server before you can access data. Only one IMS service can be active in the data server.

Procedure

1. Provide IMS configuration information in the IMS section of the customization parameters file.
2. Import the SCACCONF(CACSVIMB) member to define the BMP/DBB service. Issue the following MTO IMPORT command:

```
IMPORT,CONFIG,FILENAME=DSN:SCACCONF(CACSVIMB)
```

Modifying BMP and DBB JCL:

Modify JCL streams to access IMS data with a BMP or DBB interface.

Procedure

- Provide IMS configuration information in the IMS section of the customization parameters file.
 - High-level qualifier: Modify IMSINHLQ=IMS to specify the high-level qualifier for IMS data sets.
 - PSB name: Modify IMSDFPSB=DEFPSB to specify a PSB name.

The SCACSAMP (CACBMP) member JCL and SCACSAMP (CACBMP) member JCL will be updated based on the customization parameters file. Otherwise, edit these members to specify any additional values not included in the customization parameters file.

- For IMS BMP JCL only, modify the JCL for BMP access.
 1. Edit the BMP data server JCL stream, SCACSAMP member CACBMP. This member runs the data server as a batch job that is capable of accessing IMS data by using a BMP interface.
 2. Supply a valid job card.
 3. Modify the JCL to conform to site specifications and specify the following parameters at the beginning of the in-stream procedure: the data server high-level qualifier (CAC) or SYSOUT class (SOUT).
 4. Save the changes.
- For IMS DBB JCL only, edit the DBB data server JCL stream, SCACSAMP member CACDBB. This member runs the data server as a batch job that accesses IMS data by using a DBB interface.
 1. Supply a valid job card.

2. Modify the JCL to conform to site specifications. Specify the following parameters at the beginning of the in-stream procedure:
 - The data server high-level qualifier, CAC
 - SYSOUT class, SOUT.
3. Supply additional IMS information:
 - The correct ACB library for DBB access.
 - DD statements for the database files, if you are not using dynamic allocation (databases not defined to IMS).

Required: When you run DBB with IRLM=Y, IMS requires a non-swappable address space. IMS makes the address space non-swappable at initialization time automatically when you set the JCL SWAP parameter to SWAP=N. Otherwise, IMS automatically makes the region non-swappable when the first DL/I call is issued. This situation can result in a significant delay in processing the call. In addition, the IMS batch service definition in the data server configuration file needs to be placed immediately after the logger service to ensure the shortest possible timeframe to make the region non-swappable.

4. Save the changes.

IMS setup for dynamic discovery from the Classic Data Architect

To validate CREATE TABLE and CREATE INDEX statements that reference an IMS database, you need to update the data server JCL.

Before you begin

The user ID that is associated with the address space requires read permission to access the files that the DBDLIB DD statement references. The DBD libraries that the DBDLIB DD statement references do not require APF-authorization.

Procedure

Include a DBDLIB DD statement that references the IMS DBD library. Alternatively, the DBDLIB DD statement can reference libraries that contain the load module generated from the IMS DBDGEN. The libraries containing this load module are created for the databases referenced by the DBD-name in the CREATE TABLE statement. If the table references an IMS logical database, the physical DBDs referenced must also be accessible through the DBDLIB DD statement.

Running the data server

You are now ready to run the data server. Before you submit the JCL, ensure that the operational environment is set up properly.

Procedure

1. Ensure that the SCACLOAD library is APF authorized. If the SCACLOAD library is not APF authorized, you will receive a S047 abend when you attempt to run the data server. IMS libraries concatenated to SCACLOAD must also be APF authorized.
2. Ensure that the user ID that the data server runs with has access and execute authority for the data sets that are referenced in the data server JCL. Contact your security administrator to verify the authorizations.
3. Start the data server by submitting the data server job.

What to do next

To verify that the data server is operational, select the job while it is running. Look for the following message near the top of the listing:

```
CAC00103I DATA SERVER: V11.3 READY
```

To shut down the data server, you can use the z/OS MTO interface and issue a STOP command. If you do not have MTO authority, cancel the data server job.

Setting up access to sequential files

You need to configure the data server to enable access to sequential files.

Before you begin

If the SAF exit is enabled and an external security manager such as RACF, ACF2, or Top Secret is in use, the user ID associated with the data server also requires the necessary access privileges to process the data sets associated with the data source. After the end user ID is validated, the server ID is used to manage discrete operations for those users.

About this task

The setup requirements for access to sequential files also apply to validation setup for processing CREATE TABLE statements to map sequential files.

Procedure

Ensure that the corresponding DD statement exists in the server JCL that references the files. The DSN parameter on the DD statement identifies the physical file to be accessed.

Setting up access to VSAM

You need to configure the data server to enable access to VSAM files through the CICS VSAM, native VSAM, and DFSMStvs interfaces.

Before you begin

If the SAF exit is enabled and an external security manager such as RACF, ACF2, or Top Secret is in use, the user ID associated with the data server also requires the necessary access privileges to process the data sets associated with the data source. After the end user ID is validated, the server ID is used to manage discrete operations for those users.

Setting up access to CICS VSAM

To access VSAM data through CICS, the data server establishes a VTAM® LU 6.2 connection to CICS to initiate a transaction when a query begins and uses this transaction to communicate with CICS during query processing.

About this task

Consider two types of access to CICS VSAM files:

- Federated access
- Stored procedure access

Federated access

All configurations that require access to CICS VSAM files use federated access, which provides direct access to the files without performing functions or data transformations. To set up federated access, you define a VSAM connector program that runs in the CICS region. The VSAM connector discovers the attributes of a source VSAM file when you use the Classic Data Architect to map the relational tables in the data server.

The default name of the VSAM connector program is CACCIVS. Beginning in Classic federation Version 10.1 PTF rollup 2, this program provides access to extended-format VSAM data sets with CICS version 3.2 or higher.

Configure your user copy of the sample member CACCDEF in the USERSAMP data set to create the CICS definitions required for CICS VSAM support. The default CICS transaction name is EXV1, and if you change this name you must change it in the CICS definitions as well. Using the configured sample, your CICS administrator can follow the instructions in CACCDEF to install the VSAM connector definitions.

Stored procedure access

If you want to run stored procedures to perform functions or transform data, implement stored procedure access. Many sites implement both types of access. Stored procedure access requires that you run a bridge program CACSPBR in the CICS region.

When you set up CICS definitions, you supply a program name in a program definition. A sample program definition is in the load module CACSP62 in your user copy of the SCACLOAD data set. Your CICS administrator then installs the load module in a library that is in the DFHRPL concatenation.

Procedure

1. Configure VTAM resource definitions and CICS resource definitions to establish the CICS VSAM environment.

Both federated and stored procedure access require VTAM and CICS definitions. If you add stored procedure access, the CICS definitions are different. See the topic about sample VTAM and CICS definitions.

2. Using the sample member CACCDEF, configure the CICS transaction, program, connection, session, and file definitions and ask your CICS administrator to install them in the CICS target system.

3. Configure and start the VSAM initialization service in the Classic Data Architect to establish communications with the target CICS subsystem.

To enable the VSAM service, configure the service definition for the service class VSMS.

Setting up access to native VSAM

You need to configure the data server to access native VSAM.

About this task

You cannot use the VSAM service to access DFSMStvs.

Procedure

- Define the VSAM initialization service during the installation customization process by specifying the parameters in the VSAM section of the customization

parameters file. Otherwise, you can define the service by using the Classic Data Architect to add the service or with the MTO ADD command as shown in the following example.

```
ADD,CONFIG,SERVICE=VSAMSRV,SERVICECLASS=VSMS
```

- If you specify a DD name that references a VSAM file in a table definition, update the data server JCL with the corresponding DD statement.

Setting up access to DFSMStvs

Eligibility to access DFSMStvs files is determined dynamically by the data set attributes defined in the Integrated Catalog Facility (ICF) catalog.

Before you begin

An RRS query processor is required for access to DFSMStvs.

VSAM Record Level Sharing (RLS) and DFSMStvs with RRS must be installed.

About this task

- DFSMStvs does not support the VSAM service (CACVSMS).
- DFSMStvs support is restricted to access performed at the record level. DFSMStvs does not support the following VSAM functions:
 - Linear data sets
 - Control interval processing (CNV)
 - KSDS addressed access
 - Access to key range data sets
 - Access to temporary data sets
 - Clusters defined with the IMBED option
 - Improved control interval processing

For a complete list of limitations, see the DFSMStvs documentation about planning.

Procedure

1. Ensure that the installation customization process is completed which defines the RRS query processor service to the QPRR service class.
2. Use IDCAMS to define recoverable data sets. Use the DEFINE CLUSTER command for new data sets or use the IDCAMS ALTER command for existing data sets. To designate a file as recoverable, you must specify the UNDO or the ALL parameter. DFSMStvs access does not occur if NONE is specified.

Specify one of the following LOG parameters to assign the recovery attribute for a data set:

LOG

(NONE | UNDO | ALL)

NONE

Data set is non-recoverable. Undo or redo logging is not performed. NONE is the default

UNDO

Data set is recoverable. Only undo logging is performed.

ALL

Data set is recoverable. Both undo and forward recovery logging are performed

- If you specify forward recovery with the LOG(ALL) parameter, you must also define a forward recovery log stream with the LOGSTREAMID parameter.
3. If you specify a DD name that references a VSAM file in a table definition, update the data server JCL with the corresponding DD statement.

Logging for data servers

The logger service (CACLOG) accepts log messages from the services that are running in a data server and writes those messages either to a temporary or permanent data set or to a z/OS log stream.

To view log records without stopping the data server, you can configure the logger service to print the log messages to the SYSOUT DD by setting the **DISPLAYLOG** parameter to TRUE. However, the cost of writing data continuously in this way leads to high processing and storage overhead in the address space where the data server is running.

Optionally, store log records in a log stream. This approach also enables you to view the logs while the data server is running, and has these additional advantages:

- The log print utility consumes processing cycles only when you view or print records.
- The processing overhead is outside the data server address space.

You configure the logger service in the Console Explorer in the Classic Data Architect, or by issuing SET commands against the logger service.

Logging terminology

These basic terms help you to understand how logging works in your Classic environment.

Definitions

Diagnostic log

A z/OS log stream or data set that the logger service in a Classic data server uses to store informational, error, console, and trace messages based on setting of the TRACELEVEL parameter.

When writing to a z/OS log stream, the STREAMNAME parameter for the logger service defines the log. When writing to a data set, the CACLOG DD defined to the server or utility JCL defines this log.

Job log

A z/OS log related to a running data server or utility. This log contains console messages that the utility or the services running in the data server issue.

Log A file used to record changes made in a system.

Log stream

A sequence of data blocks that provides storage for IBM z/OS operations. z/OS identifies each log stream by its own unique identifier, or log stream name.

Defining logger services

You define the logger service by creating a data server during the installation customization process.

Before you begin

If you want the logger service to write log messages to a z/OS log stream, you must know the name of the log stream.

About this task

When you create a data server during the installation customization process, the logger service is pre-defined. The configuration parameters are set to the default values.

If you do not use a z/OS log stream, then the logger service writes log messages to the data set that the CACLOG DD specifies in the data server definition.

Unlike other services, the default value of the **TRACELEVEL** parameter for the logger service is 1. This value controls what other services send to the logger service and what the logger service writes to the log.

When the **DISPLAYLOG** parameter is set to TRUE, the logger service writes log records to the data set in the SYSOUT DD statement. The data server locks the log file, so this approach enables you to view log messages while the data server is running. However, using **DISPLAYLOG** has processing overhead and storage costs related to formatting and displaying the logs.

To avoid the resource consumption related to using **DISPLAYLOG**, you can configure the logger service to write to log streams instead. You can then use the log print utility to both read and print the logs. Any processing overhead occurs outside the address space of the data server, and only when the log print utility runs.

To modify the logger service definition, perform the steps in the following procedure:

Procedure

1. Modify the service definition for the logger service by using the Classic Data Architect or by issuing the SET configuration command against the service.
2. Follow these guidelines to modify the required configuration parameters:

LOGBUFSIZE

Changes the default size of the log buffer. The default log buffer size is 64 KB. Values can be between 4K and 1000K.

DISPLAYLOG

If you choose the **DISPLAYLOG** option, set this value to TRUE to mirror the log records to the data set that you specify in the SYSOUT DD statement.

STREAMNAME

If you choose to write the log records to a z/OS log stream, specify the name of the log stream.

3. Restart the data server to put your changes into effect.

Configuring TCP/IP connection handlers

Because Classic Data Architect communicates with the data server through a TCP/IP connection, you must configure a TCP/IP connection handler in the data server.

Before you begin

Before configuring a TCP/IP connection handler service, contact your network administrator to obtain the host name or IP address where you will be running the data server. Also, ask your network administrator for a unique TCP port number that is not being used by any other applications on that z/OS system.

Restrictions:

- The port number or service name must not be in use by any other application. The port number should be greater than 1024 because numbers 1 through 1024 are often reserved. The default configuration uses the well known port number 9087. If this port is assigned to another application, you need to change the sample configuration members.
- A z/OS system can support multiple TCP/IP stacks and multiple IP addresses. It is important that you validate your selected IP address with your network administrator.
- The sample TCP/IP connection handler definitions use the IP address 0.0.0.0 which normally resolves to the IP address of the local host. In most cases, you do not need to provide the actual IP address.

About this task

If you specify an IP string and port when you edit the customization parameters file during the installation customization process, or if you imported a configuration from another data server, a TCP/IP connection handler is already running in your data server. Follow these steps to change the address or port, or to set up a new connection handler.

If you want multiple users to have concurrent access to the data server, you can also follow these steps to add another connection handler.

You can modify or add connection handlers in the Console Explorer in the Classic Data Architect. Alternatively, you can issue SET commands against an existing service, such as the default INIT service, or issue ADD and SET commands against a new service name.

Procedure

- To modify the default connection handler, issue a SET,CONFIG command against the INIT service to modify the COMMSTRING parameter. The remaining default settings for the connection handler are adequate in most situations. The following example shows how to modify the default connection handler INIT:

```
SET,CONFIG,SERVICE=INIT,COMMSTRING=TCP/0.0.0.0/9087
```

The client connection string supports Internet Protocol Version 4 (IPv4) and Internet Protocol Version 6 (IPv6). This example uses IPv4.

- To add a new connection handler, issue ADD and SET commands and provide a new service name in place of the default name of INIT. The following example shows how to add a new connection handler with the name CH:

```
ADD,CONFIG,SERVICE=CH,SERVICECLASS=INIT
```

```
SET,CONFIG,SERVICE=CH,COMMSTRING=TCP/0.0.0.0/9087
```

Understanding Classic data server configuration methods

After completing the installation customization process, you can use Classic Data Architect to update the definitions of the configurations for the Classic data server.

The initial configuration of a Classic data server is accomplished with the installation customization process. When you complete the installation customization process, the installation environment includes an operational Classic data server and all required services. The services are pre-configured during the customization process. You can then build upon the operational environment as needed.

You can use tools to administer the configurations for the Classic data server. In most cases, you need to stop and then restart the data server for the configuration updates to take effect.

The tools that you can use include:

- **Classic Data Architect**—Use Classic Data Architect to modify configurations while the Classic data server is running.
By using the wizards that Classic Data Architect provides, you can make updates to a configuration for a Classic data server, including modifying global configuration parameters that affect server-wide configuration; and adding, modifying, or deleting configuration settings for services.
Classic Data Architect provides a master terminal operator (MTO) command interface that you can use to issue MTO commands remotely from the Console Explorer within Classic Data Architect after you establish an operator connection to the Classic data server.
- **Configuration-related operator commands**—Issue the configuration-related commands by using the z/OS MTO interface. You can use the following configuration-related MTO commands to administer data server configurations against a running Classic data server:
 - **IMPORT** and **EXPORT** commands that you can combine into a command file. This capability enables you to import and export configurations for a Classic data server for backup and recovery purposes. Another use of these commands is to export an existing definition, modify the exported definition, and then import that definition to another Classic data server. You can also use these commands to perform migration functions.
 - **ADD**, **SET**, **DELETE**, and **DISPLAY** commands that you can use to update and display configuration data for a Classic data server.
With these commands, you can update configuration information for global configuration parameters, services, service classes, and service lists.
- **Configuration migration and maintenance utility**—Use the CACCFGUT configuration migration and maintenance utility to update configurations for a Classic data server offline, when the Classic data server is not running.
The main purpose of the utility is for backup and recovery of configuration information. You can use the **EXPORT** and **IMPORT** commands to restore a configuration environment to a previous point in time.

XM protocol strings

Define XM protocol strings to specify the cross-memory queues that your environment uses for communication among Classic federation components that run on the same image.

Purpose

Cross-memory communication provides an efficient mechanism for communication among Classic components that run on the same logical partition (LPAR).

Syntax

To configure cross-memory communication, you specify matching protocol strings in the configuration settings of both components that use the cross-memory queue.

The syntax of an XM protocol string is as follows:

XM1/Dataspace-Name/Queue-Name/Dataspace-Size

where:

XM1

Indicates that the communication protocol is for cross-memory services. This segment is the same for all XM protocol strings.

Dataspace-Name

Indicates the name of the cross-memory dataspace that your environment uses to communicate on this cross-memory queue.

A dataspace name can be up to 16 characters long, can contain only alphabetic, numeric, and national characters, and cannot start with a numeric character. The first three characters must be unique within the job or started task that creates the dataspace.

Queue-Name

Indicates the queue name for this cross-memory queue. A queue name can be up to 16 characters long.

Dataspace-Size

Indicates the size, in megabytes, of the cross-memory queue that uses the dataspace.

Example: 2048 Mb = 2 Gb (the maximum size).

You can modify the dataspace name and queue name to conform to site standards. The dataspace names are valid for the entire LPAR, so avoid naming conflicts with cross-memory communication that other Classic services use on the same LPAR.

Example

The following example specifies an XM protocol string. The dataspace name is XSY1 and the queue name is XSYX. The size of the queue is 2048 Mb (2 Gb).

XM1/XSY1/XSYX/2048

Mapping tables for Classic federation

Use Classic Data Architect to create relational tables and views that map to data sources in supported nonrelational database management systems. With InfoSphere Classic Federation Server for z/OS, client applications can issue SQL queries against these tables to access data in the nonrelational databases. Client applications can also issue INSERT, DELETE, and UPDATE requests against the tables to modify the data in the nonrelational databases.

Before you begin

You must perform the following tasks on the data server where the query processor will run:

- Create and initialize a metadata catalog.
- Set up the configuration file.
- Start the data server.

About this task

You create the relational tables and views in a project in Classic Data Architect. Then, you promote these objects to a data server.

Procedure

1. Configure Classic Data Architect by creating prerequisite objects, creating connections to data servers, setting preferences, importing reference files, and granting privileges. See “Configuring Classic Data Architect.”
2. Create tables and views that client applications can issue SQL queries and updates against. See “Mapping data for Classic federation” on page 99.
3. Optional: Modify your tables or views. See “Viewing and modifying objects for Classic federation” on page 129.
4. Generate and run DDL to promote your tables, views, indexes, and stored procedures to a data server. See “Generating DDL” on page 142.
5. Optional: If you choose not to run the DDL from Classic Data Architect but from the metadata utility, export the DDL to a remote z/OS host. See “Exporting SQL to remote z/OS hosts” on page 142.

Configuring Classic Data Architect

Before you or other users can use Classic Data Architect to create objects and promote them to data servers, you need to perform several configuration tasks.

Before you begin

You must perform the following tasks on the data server where the correlation service will run:

- Set up and configure a data server.
- Start the data server.

Procedure

1. Create objects for organizing your work in Classic Data Architect. See “Creating objects to organize your work” on page 93.
2. Optional: Create a connection to at least one data server. If you are using a DB2 for z/OS database for Classic federation, create a connection to that database. See “Creating connections to data servers and to DB2 for z/OS” on page 95. Connections to data servers are required if you plan to:
 - Run DDL on a data server directly from Classic Data Architect. If you do not want to run the DDL from Classic Data Architect, you can export the DDL in an SQL file for batch processing.
 - Use the discovery process in Classic Data Architect to locate in Adabas or CA-IDMS databases the data structures that you want to map to.
3. Set various preferences. See “Setting preferences” on page 97.

4. Import Classic reference files into a data design project. See “Importing data definitions into projects” on page 94.
5. Grant privileges to users to create objects or to run commands on a data server. See “Granting privileges and privileges for performing actions on data servers” on page 97.

Migrating the workspace from a previous version of CDA

If you are migrating from a previous version of Classic Data Architect you may have to migrate your workspace the first time you start Classic Data Architect. The new views and perspective changes are not displayed until after the migration.

About this task

The procedure for migrating from an older version of Classic Data Architect varies depending on the version you are migrating from:

Procedure

Migrating from CDA v10.1

- The first time you start the new version of Classic Data Architect a dialog box will appear, asking you to confirm the migration of the workspace. Classic Data Architect will restart once the migration completes.

Migrating from CDA v9.5

- Reset the Data perspective by selecting **Window > Reset Perspective**.
- Display the new default perspective by selecting **Window > Open Perspective > Replication**.
- Recreate your data source connections in the Data Source Explorer.

Results

The workspace has been migrated. The new views, as well as the Replication perspective (previously called the Classic Replication Management perspective) and the Data perspective (previously called the Classic Integration perspective) are now available.

Creating objects to organize your work

You organize the tables, views, indexes, and stored procedures that you create in different containers within Classic Data Architect.

About this task

Workspaces

When you first open Classic Data Architect, you create a workspace that will contain your work. This workspace is located on your local workstation, and you do not share it with others. All users who work in Classic Data Architect have their own workspaces and their own projects within those workspaces. Within a workspace, there are two types of objects that you must create: data design projects and physical data models.

Data design projects

Within a project, you design the tables and views that you eventually will create in the metadata catalog on the data server. The type of project that you create in Classic Data Architect is called a *data design project*. You can

create any number of data design projects, using them to organize your physical data models, or you can put all of your physical data models into one project.

Physical data models

A physical data model is a collection of schemas that contain the tables that you create to map to the data in your data sources. Schemas can also contain views on those tables and stored procedures that you might want to use to perform operations on the result sets for queries. Instead of creating your tables directly in the metadata catalog on the data server and modifying them there, you create and modify your tables in the model and then promote them to a metadata catalog.

For example, you might have one data server for a test environment and another for a production environment. In a model, you can create a table and then run the DDL to create the table in the test data server to test the table. If you need to modify the table, you can drop the table from the metadata catalog in the test data server, modify the table in Classic Data Architect, then re-create the table in the metadata catalog and test it again. When you have a version of the table that you want to put into production, you can create the table in the production data server.

Procedure

1. Create a data design project. Open the New Data Design Project wizard by selecting **File > New > Data Design Project** from the menu bar at the top of Classic Data Architect. The data design project appears in the Data Project Explorer.
2. Create a physical data model with the database type Classic Integration. Open the New Physical Data Model wizard by right-clicking the data design project folder and selecting **New > Physical Data Model**. Select Classic Integration as the database type, and select the appropriate version. A new physical data model appears in the Data Models folder of your project. Close the Diagram1 tab because diagramming is a function in Eclipse but not a function in Classic Data Architect.

Importing data definitions into projects

Before you can create tables with Classic Data Architect, you must import data definition files into a data design project. For CA-Datcom, CICS VSAM, sequential, native VSAM, and IMS, you can import COBOL copybooks, PL/I include files, and DBDs. For CA-IDMS, you can import schema and subschema reports.

About this task

If you want to create tables that map to Adabas databases, you do not need to import files that describe the data structures that you want to map to. When you create an Adabas table, you provide information that tells Classic Data Architect how to locate the data structures in the Adabas database that you want to use.

If you want to create tables that map to CA-IDMS databases, you can either import schema and subschema reports into Classic Data Architect or you can tell Classic Data Architect how to locate the records and sets that you want to use in the CA-IDMS database.

When you import files into a project from a local or remote server, the files are organized into subfolders for that project according to the file types.

- COBOL copybooks are placed into a subfolder that is called **COBOL Copybooks**.
- PL/I include files are placed into a subfolder that is called **PL/I Include Files**.
- CA-IDMS schema and subschema reports are placed into a subfolder that is called **CA-IDMS Schemas/Subschemas**.
 - You can also tell Classic Data Architect to discover subschema information directly from CA-IDMS.
- DBD files are placed into a subfolder called **IMS DBDs**.
- SQL scripts are placed into a subfolder that is called **SQL Scripts**.

You can then use the files that you import into a project to create one or more tables in one or more schemas in your project.

To select the files to import, you can browse the local file system for files or connect to a z/OS server and browse data sets for members to download using FTP.

When you import a file from your local file system, the file extension is replaced. For example, a COBOL copybook `mycopybook.fd` is imported into the project as `mycopybook.cpy`. Imported files are given the following extensions:

Table 14. File types and extensions given when files are imported.

File type	Extension
COBOL copybook	cpy
PL/I include file	inc
Schema ¹	sch
Subschema ¹	sub
DBD	dbd

¹When you create tables using subschemas and schemas, the IDMS table wizard validates the selected schema and subschema pair to ensure they are related during the actual mapping process.

Procedure

1. In the Data Project Explorer, right-click the project folder and select **Import**.
2. Select **Classic Data Architect References** and click **Next**.
3. Use the wizard to select the location and verify the contents of the data definition files that you want to import, select the references, and specify the folder that you want to import the files into.
4. Optional: If you imported one or more copybooks or include files, you can validate the data definitions one at a time. In the appropriate folder for the file type, such as COBOL Copybook or PL/I Include File, right-click a data definition file and select **Validate <file type>**. If Classic Data Architect detects errors, an error dialog directs you to the Problems view, which is located on a tab in the lower right quarter of the window. Double click an error to open the file in the Editor with the cursor positioned on the line containing the error.

Creating connections to data servers and to DB2 for z/OS

Several of the tasks that you perform in the Classic Data Architect require connections to data servers and to DB2 for z/OS databases.

About this task

You must connect to a data server to run SQL DDL to promote your tables, views, and stored procedures to a metadata catalog.

If you plan to use Classic federation to query or update a DB2 for z/OS database, you must create a connection to that database from Classic Data Architect.

Procedure

- Create connections to one or more data servers. Open the New Connection wizard by right-clicking the Connections folder in the Data Source Explorer and selecting **New Connection**.

If you create a connection to a data server by using IPv6, you might need to provide a scope if you use a link-local address. The scope is typically the network interface name that you specify following the IPv6 address. The format is `ipv6 address%scope`. For example, you can specify the following value in the host field of the Driver properties window:

```
fe80::xxxx:xxxx:xxxx:xxxx%eth0
```

- If you are using a DB2 for z/OS database, create a connection to that database. Open the New Connection wizard by right-clicking the Connections folder in the Data Source Explorer and selecting **New Connection**.

Creating connections to the data server operator

Several of the tasks that you perform in Classic Data Architect require connections to the data server operator. You can create new operator connections, edit operator connections, import existing operator connections, export existing operator connections for use in a different workspace, and copy operator connections from the Data Source Explorer.

Before you begin

To connect to the server from Classic Data Architect, the operator service must be running on the data server.

To view server and service configuration information and active services, you must create a connection to the command operator service on the data server. When you create an operator connection, Classic Data Architect attempts to connect to the data server.

- If the connection succeeds, the new connection is added to the **Connections** folder in the Console Explorer. You can view connection details in the Properties view.
- If the connection fails, an explanation for the failure displays in the Properties View for the connection.

Procedure

- To create a new operator connection: Open the New Operator Connection wizard by using either of the following options:
 - Right-click the **Connections** folder and select **New connection**.
 - Right-click the New Connection icon in the Console Explorer.
- To edit an existing operator connection: Open the Edit Operator Connection wizard. Right-click a connection in the **Connections** folder in the Console Explorer and select **Edit connection**.

- To copy an existing operator connection, version 9.5 or greater, from the Data Source Explorer: Open the Copy Operator Connections wizard. Right-click the **Connections** folder and select **Copy Connections**.
- To import previously exported operator connections file: Right-click the **Connections** folder and select **Import Connections** to open the Import Connections dialog. If a connection with the same name already exists, its properties will be updated.
- To export operator connections to a file for use in a different workspace: Right-click the **Connections** folder and select **Export Connections** to open the Export Connections dialog.

Setting preferences

You can set various preferences in Classic Data Architect.

Procedure

1. Set global values that Classic Data Architect can use as default values in its wizards. Open the Classic Data Architect Preferences window by selecting **Window > Preferences** then expand the Classic Data Architect section of the tree.
2. Set the preferences on the various Classic Data Architect pages as needed (for Adabas, CICS VSAM, Connection Options, and so).
3. You can also specify the locale that the COBOL parser uses when validating COBOL copybooks that you want to base tables on. Expand the **Importer** then **COBOL** subsection of the Preferences window, and then select the **More COBOL options** tab. Set the locale in the **Compile time locale name** field.

Tip: For more information about COBOL and PL/I import preferences that are supported by Eclipse, see the preference pages that are associated with the importer.

Granting privileges and privileges for performing actions on data servers


If you want users of Classic Data Architect to perform actions in a metadata catalog on a data server, such as run SQL scripts or view objects, you must grant privileges to those users. You must also grant system-level privileges to any users who need to run commands on a data server.

About this task

The Privileges page of the Properties view for a database in a data design project only lists privileges. Adding privileges to the list does not automatically grant those privileges on a data server.

To grant a privilege, you must create the privilege in the Privileges page, generate the GRANT statement for that privilege, and then run the GRANT statement on a data server.

Procedure

1. Select the database in your data design project.
2. In the Properties view, click the **Privileges** tab. The table on the Privileges page lists the users who have one or more privileges.
3. Create a new privilege. Click the yellow icon () In the Grant System or Database Privilege window and specify these values:

Grantee

Type the ID of the user or group that you want to grant the privilege to, or select PUBLIC to grant the privilege to all users.

Type Select either SYSTEM or one of the following types:

\$ADABAS

Allows grantees to create, drop, and view Adabas tables and views on a data server.

\$CFI Allows grantees to create, drop, and view a metadata catalog on a data server.

\$DATACOM

Allows grantees to create, drop, and view CA-Datcom tables and views on a data server.

\$IDMS

Allows grantees to create, drop, and view CA-IDMS tables and views on a data server.

\$IMS Allows grantees to create, drop, and view IMS tables and views on a data server.

\$SEQUENT

Allows grantees to create, drop, and view sequential tables and views on a data server.

\$SP Allows grantees to create, drop, and view stored procedures on a data server.

\$VSAM

Allows grantees to create, drop, and view CICS VSAM and native VSAM tables and views on a data server.

Privilege

If you selected SYSTEM in the **Type** field, select one of the following privileges:

SYSADM

Grants all privileges on all objects that are in a metadata catalog. Users with this privilege can grant privileges and privileges to other users.

SYSOPR

Grants remote operator privileges to display reports for and manage a data server.

DISPLAY

Grants remote operator privileges for displaying reports for a data server.

4. When you want to promote the privileges to a data server, right-click the database and select **Generate DDL**. In the Generate DDL wizard, follow these steps:
 - a. On the **Options** page of the wizard, deselect all check boxes except **Fully qualified names**, **Quoted identifiers**, and **GRANT statements**.
 - b. On the **Objects** page, deselect all of the check boxes.
 - c. On the **Save and Run DDL** page, specify the name of the SQL file that the wizard will create. Verify that the GRANT statements are correct. Select the **Run DDL on server** check box.

- d. On the **Select Connection** page, select the connection to the data server, or create a new connection to a data server.
- e. On the **Summary** page, verify the actions that the wizard will perform and click **Finish**.
- f. In the Data Output view (which is by the Properties view by default), verify that the SQL statements ran successfully on the data server.

Revoking privileges for performing actions on data servers

Deleting privileges removes them only from a database in a data design project. Revoking privileges removes them from a data server.

About this task

The Privileges page of the Properties view for a database in a data design project only lists privileges. Deleting a privilege from that list does not automatically revoke that privilege from a data server.

Procedure

1. In the Data Project Explorer, right-click the database in which the privilege is listed.
2. On the **Privileges** page of the Properties view, select the **Revoke** check box for the privilege that you want to revoke.
3. Right-click the database and select **Generate DDL**. In the Generate DDL wizard, follow these steps:
 - a. On the **Options** page of the wizard, deselect all check boxes except **Fully qualified names**, **Quoted identifiers**, and **GRANT statements**.
 - b. On the **Objects** page, deselect all of the check boxes.
 - c. On the **Save and Run DDL** page, specify the name of the SQL file that the wizard will create. Verify that the REVOKE statements are correct. Select the **Run DDL on server** check box.
 - d. On the **Select Connection** page, select the connection to the data server.
 - e. On the **Summary** page, verify the actions that the wizard will perform and click **Finish**.
 - f. In the Data Output view (which is by the Properties view by default), verify that the SQL statements ran successfully on the data server.

Mapping data for Classic federation

Classic federation takes place when client applications query or update data sources by passing SQL requests to a query processor that is running on a data server.

The following list shows the tasks that you can perform for mapping data for Classic federation.

Creating Adabas tables and views for Classic federation

To query or update data in an Adabas database, you must create a relational table that maps to that database. You can also create a view on the table. You use the New Adabas Table wizard to create a table and a view.

Before you begin

- Configure the data server where you plan to run the query processor that will accept requests from client applications.

- Create a metadata catalog.
- Decide which data structures to map in your database and plan the indexes that you will need. To ensure optimal performance, you must map the underlying data correctly. This task includes ensuring that any indexes, keys, or units of data that are defined in your database are defined to the data server when you map the data. Almost any column that maps to a field definition table (FDT) or special descriptor table (SDT) definition can be used for an index.
- Configure a connection between the data server and your Adabas database.
- If you want to use Predict, know the Adabas file number of the Predict dictionary and the name of the view that you want to map to. If you are not using Predict, know the number of the Adabas file that you want to map to.

Restrictions

- Each column in the table that you create must be associated with a field in the file, a superdescriptor, or a subdescriptor.
- If Predict formatting is available, the following Predict formats are supported:
 - character (A,AL,AV)
 - binary (B) with length of 2 or 4
 - date (D, DS, DT)
 - floating point (F)
 - integer (I)
 - logical (L)
 - numeric packed and unpacked (N, NS, P, PS, U, US)
 - time (T, TS)

If only Adabas field formatting is available, the following formats are supported:

- alphanumeric (A)
- binary (B) with length of 2 or 4
- fixed point (F)
- floating point (G)
- packed decimal (P) and unpacked decimal (U)

About this task

For more information about creating tables and views that map to Adabas databases, see the related links for Adabas syntax diagrams and for views.

Procedure

1. Optional: Use the Adabas page of the Preferences window to set these default values:
 - The name of the Predict dictionary that you want to use.
 - The date format that Classic Data Architect should convert dates to.
 - The time format that Classic Data Architect should convert times to.
 - The maximum length for VARCHAR data.
 - The maximum length for LVARCHAR data.
 - The maximum number of occurrences for all fields that occur multiple times in an Adabas file.
 - Whether to use the User Synonym field from the Predict Dictionary when this field is defined.

2. Map your Adabas database to a relational table by using the New Adabas Table wizard.
 - a. Open this wizard by right-clicking either the database in your data design project or one of the schemas within the database. Select **Add Classic Object > Adabas table**.
 - b. Select the model and schema in which you want to create the table.
 - c. Choose whether or not to create a view on the table.
 - d. Choose whether to connect to your Adabas database through an existing connection to a data server or whether you want to create a new connection to a data server. Either data server must be configured to access the Adabas database.
 - e. Specify the format of dates and times, the lengths of VARCHAR and LVARCHAR data types, and the maximum number of occurs. The default values that appear are either the global defaults that are set for the Adabas database or the defaults that are set in the **Adabas** page of the Preferences window.
 - f. Specify whether you plan to use the table (and view, if you are creating one) for queries, updates, or both.
 - g. Provide either the Predict or Adabas information that is necessary for the discovery process.
 - h. Select the Adabas fields that you want to map to columns in your relational table.
 - i. Optional: In the ISN name field, provide a name for the column that maps to the Adabas Internal Sequence Number (ISN).
The default column name is ISN.
 - j. If you are creating a view, specify the criteria for the WHERE clause.
 - k. Modify the names of columns and provide null values.

When you finish the wizard, the new table appears under the selected schema. If you created a view, it also appears under the selected schema.
3. Optional: Modify the table properties or add privileges. Select the table, and make any changes in the Properties view.
4. Optional: Generate the DDL for the table. You can generate the DDL later, if you do not want to generate it now. You can also generate the DDL for all of the objects within the same schema. See “Generating DDL” on page 142.
 - a. Right-click the table and select **Generate DDL**.
 - b. In the Generate DDL wizard, follow these steps:
 - 1) Choose to generate CREATE statements.
 - 2) Choose to generate DDL for tables.
 - 3) Name the file in which to save the DDL within your project.
 - 4) Choose whether to run the DDL on a data server. After you run the DDL, check the Data Output view to determine if the DDL ran successfully.
 - 5) Choose whether to open the DDL for editing.
5. Optional: If you ran the DDL successfully on a data server, validate the table by running a test query against your Adabas database. Be sure that the data server is connected to that database.
 - a. In the Data Source Explorer, search your data server for the schema that you created the table in. Expand the schema and expand the **Tables** folder.
 - b. Right-click the table and select **Data > Sample Contents**.

- c. Check the Data Output view to determine whether the test query ran successfully.
6. Optional: If you created a view, you can generate the DDL for the view now or later. You can also generate a DDL for all of the objects within the same schema. See “Generating DDL” on page 142.
 - a. Right-click the view and select **Generate DDL**.
 - b. In the Generate DDL wizard, follow these steps:
 - 1) Choose to generate CREATE and ALTER statements.
 - 2) Choose to generate DDL for views.
 - 3) Name the file in which you want to save the DDL within your project.
 - 4) Choose whether to run the DDL on a data server. After running the DDL, check the Data Output view to find out whether the DDL ran successfully.
 - 5) Choose whether you want to open the DDL for editing.
7. Optional: If you ran the DDL successfully on a data server, validate the view by running a test query against your Adabas database. Be sure that the data server is connected to that database.
 - a. In the Data Source Explorer, search your data server for the schema that you created the view in. Expand the schema and expand the **Views** folder.
 - b. Right-click the view and select **Data > Sample Contents**.
 - c. Check the Data Output view to determine whether the test query ran successfully.

Creating CA-Datcom tables and views for Classic federation

To query or update data in a CA-Datcom database, you must create a relational table that maps to that database. You can also create a view on the table to filter record types or to filter rows and columns.

Before you begin

- Configure the data server where you plan to run the query processor that will accept requests from client applications.
- Create a metadata catalog.
- Decide which RECORD entity-occurrence and fields you want to map to.
- Create a URT for accessing the CA-Datcom table.
- Ensure that a COBOL copybook or (on the Windows platform) PL/I include file that references the database is in the appropriate folder in your data design project.

COBOL copybooks are stored in the **COBOL Copybooks** folder, and PL/I include files are stored in the **PL/I Include Files** folder.

The CA-Datcom Source Language Generator (SLG) Facility allows you to generate COBOL copy books for CA-Datcom tables. You can use member CACDCSLG in the SCACSAMP data set for help with generating these copybooks. This member invokes the CA-Datcom DDUTILITY.

If you use the CA-Datcom DDUTILITY to generate COBOL copybooks, edit the resulting copybooks to remove any \$ characters that might be present.

- Configure a connection between the data server and your CA-Datcom database, if you plan to generate and run the DDL to create the table in the metadata catalog.

Restrictions

- Fields that use the CA-Datcom null indicator are not supported.

About this task

One or more FIELD entities are associated with a table in a CA-Datcom database. These FIELD definitions describe the contents of the table. In the CA-Datcom documentation, FIELD entities are also referred to as *columns* when the SQL option is used to access the CA-Datcom table. The online help does not use the term *column* to refer to a CA-Datcom FIELD entity.

The table that you map to must have one or more CA-Datcom ELEMENT definitions associated with it. A CA-Datcom ELEMENT definition refers to one or more contiguous CA-Datcom FIELD entities and is the unit of data transfer between Classic federation and a CA-Datcom database.

Use the New CA-Datcom Table wizard to create the table and optionally the view. For more information about creating tables and views that map to CA-Datcom databases, see the related links for CA-Datcom syntax diagrams and for views.

Procedure

1. Map your CA-Datcom database to a relational table and optionally a view by using the New CA-Datcom Table wizard.
 - a. Open the wizard by right-clicking either the database in your data design project or one of the schemas within the database. Select **Add Classic Object > CA-Datcom table**.
 - b. Select the copybook or include file that you want to base the table on.
 - c. Choose whether to use the table for queries, updates, or both.
 - d. Choose whether to create a view on the table.
 - e. Provide information about which record entity you want to map to and the URT for accessing the corresponding CA-Datcom table.
 - f. Select the fields that you want to map to columns in your relational table.
 - g. If you are creating a view, specify the criteria for the WHERE clause.

When you finish the wizard, the new table appears under the selected schema. If you created a view, the view also appears under the selected schema.

2. Optional: Modify the table properties or add privileges. Select the table and make any changes in the Properties view.
3. Optional: Create one or more indexes on the table. See “Creating indexes” on page 116.
4. Optional: Generate the DDL for the table. You can generate the DDL later, if you do not want to generate it now. You can also generate the DDL for all of the objects within the same schema. See “Generating DDL” on page 142.
 - a. Right-click the table and select **Generate DDL**.
 - b. In the Generate DDL wizard, follow these steps:
 - 1) Choose to generate CREATE statements.
 - 2) Choose to generate DDL for tables. You can also choose to generate DDL for indexes.
 - 3) Name the file in which to save the DDL within your project.
 - 4) Choose whether to run the DDL on a data server. After you run the DDL, check the Data Output view to determine if the DDL ran successfully.
 - 5) Choose whether to open the DDL for editing.

5. Optional: If you ran the DDL successfully on a data server, validate the table by running a test query against your CA-Datacom database. Be sure that the data server is connected to that database.
 - a. In the Data Source Explorer, search your data server for the schema that you created the table in. Expand the schema and expand the **Tables** folder.
 - b. Right-click the table and select **Data > Sample Contents**.
 - c. Check the Data Output view to determine whether the test query ran successfully.
6. Optional: If you created a view, you can generate the DDL for the view now or later. You can also generate the DDL for all of the objects within the same schema. See “Generating DDL” on page 142.
 - a. Right-click the view and select **Generate DDL**.
 - b. In the Generate DDL wizard, follow these steps:
 - 1) Choose to generate CREATE and ALTER statements.
 - 2) Choose to generate DDL for views.
 - 3) Name the file in which to save the DDL within your project.
 - 4) Choose whether to run the DDL on a data server. After you run the DDL, check the Data Output view to determine if the DDL ran successfully.
 - 5) Choose whether to open the DDL for editing.
7. Optional: If you ran the DDL successfully on a data server, you can validate the view by running a test query against your CA-Datacom database. Be sure that the data server is connected to that database.
 - a. In the Data Source Explorer, search your data server for the schema that you created the view in. Expand the schema and expand the **Views** folder.
 - b. Right-click the view and select **Data > Sample Contents**.
 - c. Check the Data Output view to determine whether the test query ran successfully.

Creating CA-IDMS tables and views for Classic federation

To query or update data in a CA-IDMS database, you must create a relational table that maps to that database. You can also create a view on the table to filter record types or to filter rows and columns. You use the New CA-IDMS Table wizard to create the table and optionally the view.

Before you begin

- Configure the data server where you plan to run the query processor that will accept requests from client applications.
- Create a metadata catalog.
- Decide which records to map to and the best path through the database to access the records. The sets that are defined in the subschema for the records determine the path.
- Configure a connection between the data server and your CA-IDMS database. The data server must be able to access the CA-IDMS central version that contains the subschema definitions, schema definitions, and data of the records that are being mapped.

Restrictions

- The record path in the mapping must be from set owner to set member only.
- The owner DBKEY must be part of each member record that is included in the path.

About this task

In the New CA-IDMS Table wizard, you can map a single record or a specific path to as many as 10 records. You define a path by starting with a single record, and then navigating sets to additional records defined in the subschema. The subschema information that you use for mapping determines which records and sets are available. You can import the subschema information from a combination of CA-IDMS schema and subschema report files or directly from the CA-IDMS database by using Classic Data Architect's discovery process.

CA-IDMS schema and subschema reports are produced by running the CA-IDMS schema and subschema compilers and capturing the punched output into a z/OS data set. JCL to punch these reports is in the SCACSAMP library with the member name CACIDPCH.

When the data server returns SQL rows for a logical table that is mapped to a path, the data server returns an instance of the first record type that is mapped with each instance of related records down the defined path. See the example section below.

For more information about creating tables and views that map to CA-IDMS databases, see the related links for CA-IDMS syntax diagrams and for views.

Procedure

1. Map your CA-IDMS database to a relational table and optionally a view by using the New CA-IDMS Table wizard.
 - a. Open the wizard by right-clicking either the database in your data design project or one of the schemas within the database. Select **Add Classic Object > CA-IDMS table**.
 - b. Select the CA-IDMS schema and subschema to base the table on.
 - c. Choose whether to use the table for queries, updates, or both.
 - d. Choose whether to create a view on the table.
 - e. Provide information about how to access the CA-IDMS database.
 - f. For each record in the path, specify a COBOL copybook, select an 01 level if there is more than one 01 level, and then select the elements you that want to map as columns in your relational table.
 - g. Select the elements that you want to map to columns in your relational table.
 - h. If you are creating a view, specify the criteria for the WHERE clause.

When you finish the wizard, the new table appears under the selected schema. If you created a view, the view also appears under the selected schema.

2. Optional: Modify the table properties or add privileges. Select the table and make any changes in the Properties view.
3. Optional: Create one or more indexes on the table. See "Creating indexes" on page 116.
4. Optional: Generate the DDL for the table. You can generate the DDL later, if you do not want to generate it now. You can also generate the DDL for all of the objects within the same schema. See "Generating DDL" on page 142.
 - a. Right-click the table and select **Generate DDL**.
 - b. In the Generate DDL wizard, follow these steps:
 - 1) Choose to generate CREATE statements.

- 2) Choose to generate DDL for tables. You can also choose to generate DDL for indexes.
 - 3) Name the file in which to save the DDL within your project.
 - 4) Choose whether to run the DDL on a data server. After you run the DDL, check the Data Output view to determine if the DDL ran successfully.
 - 5) Choose whether you want to open the DDL for editing.
5. Optional: If you ran the DDL successfully on a data server, validate the table by running a test query against your CA-IDMS database. Be sure that the data server is connected to that database.
 - a. In the Data Source Explorer, search your data server for the schema that you created the table in. Expand the schema and expand the **Tables** folder.
 - b. Right-click the table and select **Data > Sample Contents**.
 - c. Check the Data Output view to determine whether the test query ran successfully.
 6. Optional: If you created a view, you can generate the DDL for the view now or later. You can also generate the DDL for all of the objects within the same schema. See “Generating DDL” on page 142.
 - a. Right-click the view and select **Generate DDL**.
 - b. In the Generate DDL wizard, follow these steps:
 - 1) Choose to generate CREATE and ALTER statements.
 - 2) Choose to generate DDL for views.
 - 3) Name the file in which to save the DDL within your project.
 - 4) Choose whether to run the DDL on a data server. After you run the DDL, check the Data Output view to determine whether the DDL ran successfully.
 - 5) Choose whether to open the DDL for editing.
 7. Optional: If you ran the DDL successfully on a data server, validate the view by running a test query against your CA-IDMS database. Be sure that the data server is connected to that database.
 - a. In the Data Source Explorer, search your data server for the schema that you created the view in. Expand the schema and expand the **Views** folder.
 - b. Right-click the view and select **Data > Sample Contents**.
 - c. Check the Data Output view to determine whether the test query ran successfully.

Example

A path of records might look like this:

```
PATH IS (EMPLOYEE, SET IS EMPL-DEP, DEPENDENT)
```

The CA-IDMS might contain these records:

```
EMPLOYEE DEPENDENTS (in EMPL-DEP set)
-----
BILL SMITH -> MARTHA -> BILLY -> SALLY
JANE WHELAN
SANRA JONES -> ROBERT
```

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

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

Creating CICS VSAM tables and views for Classic federation

To query or update data in a CICS VSAM data set, you must create a relational table that maps to that file. You can also create a view on the table to filter record types or to filter rows and columns. You use the New CICS VSAM Table wizard to create the table and optionally the view.

Before you begin

- Configure the data server where you plan to run the query processor that will accept requests from client applications.
- Create a metadata catalog.
- Decide which record elements you want to map in the CICS VSAM data set and plan the indexes that you will need. To optimize performance, you must map the underlying data correctly. This task includes ensuring that any indexes, keys, or units of data that are defined in your file are defined to the data server when you map the data.
- Configure a connection between the data server and your CICS VSAM data set.
- Ensure that a COBOL copybook or (on the Windows platform) PL/I include file that references the database is in the appropriate folder in your data design project.

COBOL copybooks are stored in the **COBOL Copybooks** folder, and PL/I include files are stored in the **PL/I Include Files** folder.

Restrictions

You can map to the following types of VSAM data sets:

- KSDS, ESDS, and RRDS files
- IAM files

About this task

For more information about creating tables and views that map to CICS VSAM data sets, see the related links for CICS VSAM syntax diagrams and for views.

Procedure

1. Map your CICS VSAM data set to a relational table and optionally a view by using the New CICS VSAM Table wizard.
 - a. Open the wizard by right-clicking either the database in your data design project or one of the schemas within the database. Select **Add Classic Object > CICS VSAM table**.
 - b. Select the copybook or include file to base the table on.
 - c. Choose whether to use the table for queries, updates, or both.
 - d. Choose whether to create a view on the table.
 - e. Provide information about which CICS file control table to use and how to access the CICS VSAM data set.
 - f. Select the elements that you want to map to columns in your relational table.

g. If you are creating a view, specify the criteria for the WHERE clause.

When you finish the wizard, the new table appears under the selected schema. If you created a view, the view also appears under the selected schema.

2. Optional: Select the table and, in the Properties view, modify any of its properties or add privileges.
3. Optional: Create one or more indexes on the table. See “Creating indexes” on page 116.
4. Optional: Generate the DDL for the table. You can generate the DDL later, if you do not want to generate it now. You can also generate the DDL for all of the objects within the same schema. See “Generating DDL” on page 142.
 - a. Right-click the table and select **Generate DDL**.
 - b. In the Generate DDL wizard, follow these steps:
 - 1) Choose to generate CREATE statements.
 - 2) Choose to generate DDL for tables. You can also choose to generate DDL for indexes.
 - 3) Name the file in which to save the DDL within your project.
 - 4) Choose whether to run the DDL on a data server. After you run the DDL, check the Data Output view to determine whether the DDL ran successfully.
 - 5) Choose whether you want to open the DDL for editing.
5. Optional: If you ran the DDL successfully on a data server, validate the table by running a test query against your CICS VSAM data set. Be sure that the data server is connected to the system where the file is located.
 - a. In the Data Source Explorer, search your data server for the schema that you created the table in. Expand the schema and expand the **Tables** folder.
 - b. Right-click the table and select **Data > Sample Contents**.
 - c. Check the Data Output view to determine whether the test query ran successfully.
6. Optional: If you created a view, generate the DDL for the view. You can generate the DDL later. You can also generate the DDL for all of the objects within the same schema. See “Generating DDL” on page 142.
 - a. Right-click the view and select **Generate DDL**.
 - b. In the Generate DDL wizard, follow these steps:
 - 1) Choose to generate CREATE and ALTER statements.
 - 2) Choose to generate DDL for views.
 - 3) Name the file in which you want to save the DDL within your project.
 - 4) Choose whether to run the DDL on a data server. After running the DDL, check the Data Output view to find out whether the DDL ran successfully.
 - 5) Choose whether you want to open the DDL for editing.
7. Optional: If you ran the DDL successfully on a data server, validate the view by running a test query against your CICS VSAM data set. Be sure that the data server is connected to the system where the file is located.
 - a. In the Data Source Explorer, search your data server for the schema that you created the view in. Expand the schema and expand the **Views** folder.
 - b. Right-click the view and select **Data > Sample Contents**.
 - c. Check the Data Output view to whether the test query ran successfully.

Creating tables and views for DB2 for z/OS databases

To query or update data in a DB2 for z/OS source, you must create a relational table that maps to that data source. You open a connection to a DB2 for z/OS subsystem and import the table or view into a physical data model. Then, you can generate and run the DDL.

Before you begin

In your data design project, you must have at least one physical model.

About this task

You can import DB2 tables, views, materialized query tables, and aliases. All of these objects become tables in a schema in a physical data model.

Dropping a DB2 object into a schema adds the object to that schema. Dropping a DB2 object into a data design project that has no schema causes Classic Data Architect to create a schema and add the DB2 object to it. Dropping an entire DB2 schema into a data design project adds that schema to the project and imports all of the DB2 objects that are in the DB2 schema.

When objects are created, they are created with DB2 column and, if applicable, indexes.

Columns that are not supported in Classic federation are not added.

Procedure

1. Create a connection to the DB2 for z/OS subsystem. Follow these steps:
 - a. In the Data Source Explorer, right-click the **Connections** folder and select **New Connection**.
 - b. In the New Connection wizard, under **Select a database manager**, expand **DB2 zSeries** and select the appropriate version.
 - c. Provide the connection information and click **Finish**.
2. In the Data Source Explorer, expand these objects:
 - a. the database
 - b. the **Schemas** folder
 - c. the schema of the table or view that you want to import
 - d. the **Tables** folder or the Views folder for the schema
3. Drag a DB2 object into the Data Project Explorer.

Dropping a DB2 object into a schema adds the object to that schema. Dropping a DB2 object into a data design project that has no schema causes Classic Data Architect to create a schema and add the DB2 object to it. Dropping an entire DB2 schema into a data design project adds that schema to the project and imports all of the DB2 objects that are in the DB2 schema.
4. In the Available Actions window, ensure that **Drop DB2 objects into a Classic model** is selected and click **OK**.
5. In the Specify DB2 Plan window, specify the name of the plan that contains the packages for the SQL statements that client applications will make against the DB2 objects. After you click **OK**, the DB2 objects appear in the your data design project.
6. Optional: Modify the table properties or add privileges. Select the table and make any changes in the Properties view.

7. Optional: Open the Generate DDL wizard and generate the CREATE statement for the table. With this wizard, you can generate the SQL DDL to define the tables and views and choose to run the DDL on a data server so that the tables and views are created in the metadata catalog for that data server. You can also edit the generated DDL before you run it.

After you run the DDL, the tables appear on the data server in the Data Source Explorer. To see the tables, expand the data server and navigate to **Schemas > table schema name > Tables**.

The views appear on the data server under **Schemas > view schema name > Views**.

If you want to generate and run the DDL for more than one object at a time, you can right-click a schema and select Generate DDL. The Generate DDL wizard will generate the DDL for all of the objects in the schema.

8. Optional: If you created the tables and views on the data server, run a test query on the tables.
 - a. In the Data Source Explorer, right-click a table or view and select **Data > Sample Contents**.
 - b. Look in the Data Output view to see the results of the test query.

Results

Creating IMS tables and views for Classic federation

To query or update data in an IMS database, you must create a relational table that maps to that database. You can also create a view on the table to filter record types or to filter rows and columns. Use the New IMS Table wizard to create the table and optionally the view.

Before you begin

- Configure the data server where you plan to run the correlation service that will process change data from your IMS database.
- Create a metadata catalog.
- Decide which segment you want to map and the required path for navigating to the segment from a physical root or index.
- Configure a connection between the data server and your IMS database.
- Ensure that the **IMS DBDs** folder in your project has a database definition file (DBD) that lists the segments from which you want to select the fields to map to columns.
- Ensure that a COBOL copybook or (on the Windows platform) PL/I include file for each IMS segment you want to map to is in the appropriate folder in your data design project.

COBOL copybooks are stored in the **COBOL Copybooks** folder, and PL/I include files are stored in the **PL/I Include Files** folder.

Restrictions

If you are creating a table that maps to an IMS database, use the table only for change capture or only for queries and updates. Typical mappings of an IMS database for queries contain column definitions for all or most of the data in the segments that are referenced by the table. For updates, restrictions on what can be inserted or updated can require you to create custom versions of your tables for updates only.

About this task

For more information about creating tables and views that map to IMS databases, see the related links for IMS syntax diagrams and for views.

Procedure

1. Map your IMS database to a relational table and optionally a view by using the New IMS Table wizard.
 - a. Open this wizard by right-clicking either the database in your data design project or one of the schemas within the database. Select **Add Classic Object > IMS table**.
 - b. Select the DBD file that you want to base the table on.
 - c. Choose whether to use the table for queries, updates, or both.
 - d. Choose whether to create a view on the table.
 - e. Provide information about how to access the IMS database.
 - f. For the each segment that is in the path, specify a COBOL copybook or include file, select the desired 01 level if there is more than one, and then select the elements that you want to map as columns.
 - g. If you are creating a view, specify the criteria for the WHERE clause.

When you finish the wizard, the new table appears under the selected schema. If you created a view, the view also appears under the selected schema.

2. Optional: Modify the table properties or add privileges. Select the table and make any changes in the Properties view.
3. Optional: Create one or more indexes on the table. See “Creating indexes” on page 116.
4. Optional: Generate the DDL for the table. You can generate the DDL later, if you do not want to generate it now. You can also generate the DDL for all of the objects within the same schema. See “Generating DDL” on page 142.
 - a. Right-click the table and select **Generate DDL**.
 - b. In the Generate DDL wizard, follow these steps:
 - 1) Choose to generate CREATE statements.
 - 2) Choose to generate DDL for tables. You can also choose to generate DDL for indexes.
 - 3) Name the file in which to save the DDL within your project.
 - 4) Choose whether to run the DDL on a data server. After you run the DDL, check the Data Output view to determine whether the DDL ran successfully.
 - 5) Choose whether to open the DDL for editing.
5. Optional: If you ran the DDL successfully on a data server, validate the table by running a test query against your IMS database. Be sure that the data server is connected to that database.
 - a. In the Data Source Explorer, search your data server for the schema that you created the table in. Expand the schema and expand the **Tables** folder.
 - b. Right-click the table and select **Data > Sample Contents**.
 - c. Check the Data Output view to determine whether the test query ran successfully.
6. Optional: If you created a view, generate the DDL for the view. You can generate the DDL later. You can also generate the DDL for all of the objects within the same schema. See “Generating DDL” on page 142.

- a. Right-click the view and select **Generate DDL**.
- b. In the Generate DDL wizard, follow these steps:
 - 1) Choose to generate CREATE and ALTER statements.
 - 2) Choose to generate DDL for views.
 - 3) Name the file in which to save the DDL within your project.
 - 4) Choose whether to run the DDL on a data server. After you run the DDL, check the Data Output view to determine whether the DDL ran successfully.
 - 5) Choose whether you want to open the DDL for editing.
7. Optional: If you ran the DDL successfully on a data server, validate the view by running a test query against your IMS database. Be sure that the data server is connected to that database.
 - a. In the Data Source Explorer, search your data server for the schema that you created the view in. Expand the schema and expand the **Views** folder.
 - b. Right-click the view and select **Data > Sample Contents**.
 - c. Check the Data Output view to determine whether the test query ran successfully.

Creating sequential tables and views for Classic federation

To query or update data in a sequential file, you must create a relational table that maps to that file. You can also create a view on the table to filter record types or filter rows and columns. You use the New Sequential Table wizard to create the table and optionally the view.

Before you begin

- Configure the data server where you plan to run the query processor that will accept requests from client applications.
- Create a metadata catalog.
- Decide on the sequential files that you want to map to.
- Ensure that a COBOL copybook or (on the Windows platform) PL/I include file that references the database is in the appropriate folder in your data design project.

COBOL copybooks are stored in the **COBOL Copybooks** folder, and PL/I include files are stored in the **PL/I Include Files** folder.

Restrictions

- Sequential data sets cannot be updated.
- Because sequential data sets do not have any native index definitions or keys, any request to access a sequential data set causes a table scan. You cannot use Data Architect to create indexes for tables that are mapped to sequential data sets.
- If you are mapping partitioned sequential data sets, a table must map to a single member within a partitioned data set.
- SQL access to extended partitioned data sets is not supported.
- When a table references a direct access data set, these data sets are referred to as BDAM (Basic Direct Access Method) data sets. BDAM data sets can be accessed using “keys” that consist of track addresses, block numbers, or a combination of the two. Classic Data Architect does not access a direct access data set using any of these techniques, but Classic Data Architect can sequentially retrieve the records that are stored in one of these direct access data sets.

About this task

Classic federation uses two methods to physically access a sequential file:

- The table definition can refer to the data set name. This method requires the data server to issue dynamic allocation requests before the file is physically opened. For Classic Federation to use dynamic allocation, the file must be cataloged.
- The table definition can reference the file by DD (statement) name. Accessing the file by DD name requires that the file is statically and permanently allocated to the server address space because the referenced DD statement must be added to the server JCL, and the DSN parameter on the DD statement identifies the physical file to be accessed.

The recommended technique is to use dynamic allocation to access a sequential file. When a file is dynamically allocated, the file disposition is share mode, which allows other applications to access the file concurrently, if the applications are not attempting to access the file in exclusive mode.

For more information about creating tables and views that map to sequential files, see the related links for sequential syntax diagrams and for views.

Procedure

1. Map your sequential file to a relational table and optionally a view by using the New Sequential Table wizard.
 - a. Open this wizard by right-clicking either the database in your data design project or one of the schemas within the database. Select **Add Classic Object > Sequential table**.
 - b. Select the copybook or include file to base the table on.
 - c. Choose whether to create a view on the table.
 - d. Provide information about how to access the sequential file.
 - e. Select the elements to map to columns in your relational table.
 - f. If you are creating a view, specify the criteria for the WHERE clause.

When you finish the wizard, the new table appears under the selected schema. If you created a view, the view also appears under the selected schema.

2. Optional: Modify the table properties or add privileges. Select the table and make any changes in the Properties view.
3. Optional: Generate the DDL for the table. You can generate the DDL later, if you do not want to generate it now. You can also generate the DDL for all of the objects within the same schema. See “Generating DDL” on page 142.
 - a. Right-click the table and select **Generate DDL**.
 - b. In the Generate DDL wizard, follow these steps:
 - 1) Choose to generate CREATE statements.
 - 2) Choose to generate DDL for tables.
 - 3) Name the file in which to save the DDL within your project.
 - 4) Choose whether to run the DDL on a data server. After running the DDL, check the Data Output view to determine whether the DDL ran successfully.
 - 5) Choose whether to open the DDL for editing.
4. Optional: If you ran the DDL successfully on a data server, validate the table by running a test query against your sequential file. Be sure that the data server is connected to the system where the sequential file is located.

- a. In the Data Source Explorer, search your data server for the schema that you created the table in. Expand the schema and expand the **Tables** folder.
 - b. Right-click the table and select **Data > Sample Contents**.
 - c. Check the Data Output view to determine whether the test query ran successfully.
5. Optional: If you created a view, generate the DDL for the view. You can generate the DDL later. You can also generate the DDL for all of the objects within the same schema. See “Generating DDL” on page 142.
 - a. Right-click the view and select **Generate DDL**.
 - b. In the Generate DDL wizard, follow these steps:
 - 1) Choose to generate CREATE statements.
 - 2) Choose to generate DDL for views.
 - 3) Name the file in which you want to save the DDL within your project.
 - 4) Choose whether to run the DDL on a data server. After you run the DDL, check the Data Output view to find out whether the DDL ran successfully.
 - 5) Choose whether you want to open the DDL for editing.
 6. Optional: If you ran the DDL successfully on a data server, validate the view by running a test query against your sequential file. Be sure that the data server is connected to the system where the sequential file is located.
 - a. In the Data Source Explorer, search your data server for the schema that you created the view in. Expand the schema and expand the Views folder.
 - b. Right-click the view and select **Data > Sample Contents**.
 - c. Check the Data Output view to find out whether the test query ran successfully.

Creating VSAM tables and views for Classic federation

To query or update data in a VSAM data set, you must create a relational table that maps to that file. You can also create a view on the table to filter record types or to filter rows and columns. Use the New VSAM Table wizard to create the table and optionally the view.

Before you begin

- Configure the data server where you plan to run the query processor that will accept requests from client applications.
- Create a metadata catalog.
- Decide which data structures to map in your data source and plan the indexes that you will need. To ensure optimal performance, you must map the underlying data correctly. This task includes ensuring that any indexes, keys, or units of data that are defined in your file are defined to the data server when you map the data.
- Ensure that a COBOL copybook or (on the Windows platform) PL/I include file that references the database is in the appropriate folder in your data design project.
COBOL copybooks are stored in the **COBOL Copybooks** folder, and PL/I include files are stored in the **PL/I Include Files** folder.

Restrictions

You can map to the following types of VSAM data sets:

- KSDS, ESDS, and RRDS files

- IAM files

About this task

For more information about creating tables and views that map to VSAM data sets, see the related links for VSAM syntax diagrams and for views.

Procedure

1. Map your VSAM data set to a relational table and optionally a view by using the New VSAM Table wizard.
 - a. Open this wizard by right-clicking either the database in your data design project or one of the schemas within the database. Select **Add Classic Object > VSAM table**.
 - b. Select the copybook or include file to base the table on.
 - c. Choose whether to use the table for queries, updates, or both.
 - d. Choose whether to create a view on the table.
 - e. Provide information about how to access the VSAM data set.
 - f. Select the elements that you want to map to columns in your relational table.
 - g. If you are creating a view, specify the criteria for the WHERE clause.

When you finish the wizard, the new table appears under the selected schema. If you created a view, the view also appears under the selected schema.

2. Optional: Modify the table properties or add privileges. Select the table and make any changes in the Properties view.
3. Optional: Create one or more indexes on the table. See “Creating indexes” on page 116.
4. Optional: Generate the DDL for the table. You can generate the DDL later, if you do not want to generate it now. You can also generate the DDL for all of the objects within the same schema. See “Generating DDL” on page 142.
 - a. Right-click the table and select **Generate DDL**.
 - b. In the Generate DDL wizard, follow these steps:
 - 1) Choose to generate CREATE statements.
 - 2) Choose to generate DDL for tables. You can also choose to generate DDL for indexes.
 - 3) Name the file in which to save the DDL within your project.
 - 4) Choose whether to run the DDL on a data server. After you run the DDL, check the Data Output view to determine whether the DDL ran successfully.
 - 5) Choose whether you want to open the DDL for editing.
5. Optional: If you ran the DDL successfully on a data server, you can validate the table by running a test query against your VSAM data set. Be sure that the data server is connected to the system where the file is located.
 - a. In the Data Source Explorer, search your data server for the schema that you created the table in. Expand the schema and expand the Tables folder.
 - b. Right-click the table and select **Data > Sample Contents**.
 - c. Check the Data Output view to find out whether the test query ran successfully.
6. Optional: If you created a view, generate the DDL for the view. You can generate the DDL later. You can also generate the DDL for all of the objects within the same schema. See “Generating DDL” on page 142.

- a. Right-click the view and select **Generate DDL**.
- b. In the Generate DDL wizard, follow these steps:
 - 1) Choose to generate CREATE and ALTER statements.
 - 2) Choose to generate DDL for views.
 - 3) Name the file in which to save the DDL within your project.
 - 4) Choose whether to run the DDL on a data server. After you run the DDL, check the Data Output view to determine whether the DDL ran successfully.
 - 5) Choose whether to open the DDL for editing.
7. Optional: If you ran the DDL successfully on a data server, validate the view by running a test query against your VSAM data set. Be sure that the data server is connected to the system where the file is located.
 - a. In the Data Source Explorer, search your data server for the schema that you created the view in. Expand the schema and expand the **Views** folder.
 - b. Right-click the view and select **Data > Sample Contents**.
 - c. Check the Data Output view to determine whether the test query ran successfully.

Creating indexes

After you define a table, you can define indexes on the table to map existing indexes on the underlying data. An index identifies columns in a table that correspond to a physical index that is defined against a data source.

Procedure

1. In the Data Project Explorer, right-click the table that you want to define an index on and select **Add Classic Object > Index**. The New Index wizard opens.
2. Name the index, specify whether it should be unique, and select the columns to base the index on.
3. If the index is for an IMS table, specify the method that Classic federation can use to select a PCB to access your IMS database.
4. If the index is for a CICS VSAM or native VSAM table, specify the DS or DD name for the index.

Creating stored procedures

Use the Create Stored Procedure window to define a stored procedure to perform work that cannot be done with SQL DELETE, INSERT, SELECT, and UPDATE operations.

Procedure

1. Open the Create Stored Procedure window by right-clicking either the database in your data model or a schema and selecting **Add Classic Object > Stored Procedure**.
2. On the Stored Procedure Definition page, specify where in your project you want to define the stored procedure and specify values to be used in the DDL that is generated for the stored procedure.
3. On the Parameter Definition page, specify at least one parameter. You can use parameters for input, output, or both.
4. Click **Finish** to add the stored procedure to your project.

Results

After the stored procedure is listed in your data model, you can select it and edit any of its properties in the Properties view. You can also set privileges on it.

When the stored procedure is ready, you can generate and run the DDL for creating the stored procedure in a metadata catalog.

If you need to modify the stored procedure after it exists in a metadata catalog, you must generate and run the DDL to drop the stored procedure from the metadata catalog. Then, you can re-create the stored procedure with the modified settings and generate and run the DDL.

Modifying the PCB selection for IMS tables or indexes

You can modify the PCB selection for an IMS table or IMS index.

Procedure

1. Open the Modify PCB Selection wizard by right-clicking the IMS table or IMS index and selecting **Modify PCB Selection Method**.
2. Select how you want Classic federation to select the PCBs that are needed to access the table.

Mapping tables and views for redefined data

Redefined data uses alternate record layouts for the same storage area, based on record types.

About this task

To read redefined data, define a table with an associated view for each type of record. To insert, update, delete, or capture changes to redefined data, define a separate table for each record type.

Each table you map for redefined data must contain columns that identify common key information and a column for the type code field. These columns are followed by type-specific columns.

In the following PL/I example, the UNION attribute defines two alternative record mappings for the same storage area. The RECORD_TYPE variable specifies whether the data that follows describes employee information or address information.

```
DCL 01 EMPLOYEE_ADDRESS_RECORD,
    05 EMP_ID          CHAR(6),
    05 RECORD_TYPE    CHAR(1),
    05 RECORD_INFO UNION,
        10 EMPLOYEE_INFORMATION,
            15 LAST_NAME      CHAR(20),
            15 FIRST_NAME     CHAR(20),
            15 DATE_OF_BIRTH  PIC '(8)9',
            15 MONTHLY_SALARY DECIMAL(7,2),
            15 FILLER         CHAR(48),
        10 ADDRESS_INFORMATION,
            15 ADDRESS_LINE_1 CHAR(30),
            15 ADDRESS_LINE_2 CHAR(30),
            15 ADDRESS_CITY   CHAR(20),
            15 ADDRESS_STATE  CHAR(2),
            15 ADDRESS_ZIP    PIC '(5)9';
```

Procedure

1. Map two tables, each with an associated view.
In the example, you define one base table and view for employee information, and another base table and view for address information.
2. Use the view or the table, depending on whether you query or update the data.
 - a. To query redefined data, supply filtering information in the view definition when you map the table. This approach simplifies queries in client applications, because the queries do not require WHERE clause filtering.

Restriction: You cannot update a view.

- b. To insert, update, or delete redefined data, your application must use the base table name and provide WHERE filtering.

Supply a WHERE clause and use the type code value to filter the records.

What to do next

Tip: To perform change capture on redefined data, mark the view for change capture when you map the table.

For additional examples of working with redefined data, see Record types in data definition examples.

Array processing

Groups of repeating fields in a record layout are called *record arrays*.

If you map a record array into a table, the DDL that is generated for the table includes the BEGINLEVEL statement to mark the start of a record array and the ENDLEVEL statement to mark the end of the record array.

The following topics provide guidelines for mapping record arrays:

Record arrays:

A group of data items in a database that have multiple occurrences within a single record in the database are referred to as *record arrays*.

Typically, you map tables and views by importing data definitions in COBOL copybooks or PL/I include files. The copybooks or include files can contain array definitions. Classic Data Architect maps array definitions that you have not flattened into columns by converting the definitions to statements within BEGINLEVEL and ENDLEVEL blocks. The application then generates the Data Definition Language (DDL) that contains the BEGINLEVEL statements. You run the DDL on the data server to create a user table.

You can optimize performance when you query array data. If you want to insert, update, delete, or perform change capture on array data, you must flatten the structure. See the related array topics for information about working with array data.

For information about working with COBOL, see the *Enterprise COBOL for z/OS Language Reference* and the *Enterprise COBOL for z/OS Programming Guide*. For information about working with PL/I, see the *WebSphere Developer for System z PL/I for Windows Language Reference* and the *Enterprise PL/I for z/OS Programming Guide*.

Fixed-length arrays

Fixed-length array constructs define an array in which the number of instances does not change. For example, an employee record can include the employee's dependent information (spouse and children). Because an employee can have multiple dependents, you can declare an array of dependent information within an employee record by specifying a COBOL OCCURS clause or a PL/I DIMENSION (DIM) attribute. The following example of a COBOL OCCURS clause defines a fixed-length array.

```
05 DEPENDENTS-ARRAY OCCURS 20 TIMES
```

The array appears 20 times in the source database record, regardless of how many dependents the employee has.

NULL IS processing

The query processor on the data server skips null array instances as SQL ROW candidates at runtime. In the example of the dependents array, if an employee has three dependents and the array occurs 20 times, 17 null instances of the array do not appear as a row in a result set.

You can include a **NULL IS** value in DDL to identify a given array instance as null, based on a comparison value. A comparison value can identify a null array instance in the following ways:

- The start of the first column in the array instance matches the comparison value (**NULL IS null-value**).
- Each character in the array instance matches a single-character comparison value (**NULL IS ALL null-value**).
- The start of a specified column in the array matches the comparison value (**NULL IS column-name EQUAL null-value**).

In the following example of **NULL IS ALL** DDL, the single character X is the comparison value. If each character in the array instance is X, that instance of the array is null.

```
MAXOCCURS 20 NULL IS ALL X
```

Null instances of a record array are not returned as a row in the result set unless ALL instances of the array are NULL. If all instances of the array are NULL, then Classic Federation returns a single row for the non-array information in the record and sets the array data items to NULL. In the dependents array example, the employee has no dependents.

Variable-length arrays

Another common record array construct defines variable-length data. The number of array instances is dependent on the value of a data item that precedes the array in the structure, such as NUMBER-OF-DEPENDENTS. In COBOL, the array declaration is as follows:

```
05 NUMBER-OF-DEPENDENTS PIC 9(4) COMP.  
05 DEPENDENTS-ARRAY OCCURS 1 TO 20 TIMES  
  DEPENDING ON NUMBER-OF-DEPENDENTS.
```

The next example shows a similar array declaration in PL/I that uses the REFER attribute to point to the variable containing the value for the number of dependents.

```
5 NUMBER_OF_DEPENDENTS BIN FIXED(15),
5 DEPENDENTS_ARRAY DIM(N1 REFER(NUMBER_OF_DEPENDENTS)),
```

PL/I has no equivalent syntax for the COBOL DEPENDING ON clause, so the data server calculates the number of array instances automatically based upon array offset, array size, and record or segment length. The formula is as follows:

$$\langle \text{Number-of-array-instances} \rangle = (\langle \text{record-length} \rangle - 1 - \langle \text{array-offset} \rangle) / \langle \text{array-size} \rangle$$

The number of instances does not appear in the DDL generated by the wizard. The number is calculated when the data server processes the DDL and validates the table information prior to creating the table definition in the catalog.

Restrictions

You cannot map a table that is based on nested variable-length arrays or a table that contains fixed columns after a variable-length array. The reason for this is that columns subsequent to variable-length array constructs do not appear at a predictable offset.

The New Table wizard supports PL/I REFER constructs, with the following restrictions:

- The REFER clause must occur in the first dimension of a multi-dimensional array.
- The array must have no later siblings, and no parents with siblings.
- The REFER clause must be in the upper bound, and the lower bound must be 1. The upper bound must reflect the actual count, and contains the actual count only when the lower bound is 1.

If the PL/I parser encounters a REFER clause that is too complex to process, the parser ignores the entire structure and the validation process displays an error related to the array definition.

Performance considerations with multiple record arrays:

When you map tables containing record array definitions, result sets from queries can return so many rows that performance suffers.

When a table has multiple record array definitions that you map as arrays, queries that reference the table yield large result sets. To avoid performance problems, do one of the following:

- Map a separate table for each record array definition.
- Flatten the array structure by mapping a separate column for each field and instance.

If you map a separate table for each array, federated queries can read the data. If you flatten the structure, client applications can insert, update, and delete rows.

When you map a record array definition as an array, Classic Data Architect converts the data definition to SQL columns. Each instance of the array is combined with the non-array data items to create SQL rows. For example, if the record in the database for the employee ID 123456789 contains three dependents, three distinct rows are returned for that record. The following query returns three rows in the result set:

Query:

```
SELECT EMP_ID, NUMBER_OF_DEPENDENTS, DEP_ID, DEP_NAME FROM  
CAC.EMPL WHERE EMP_ID = '123456789';
```

Result set:

EMP_ID	NUMBER_OF_DEPENDENTS	DEP_ID	DEP_NAME
123456789	3	111223333	Depen1
123456789	3	222334444	Depen2
123456789	3	333445555	Depen3

Calculating the size of result sets

You can use a formula to calculate the number of rows in the result set from a query on a table that contains arrays, before you supply any filtering predicates on the WHERE clause. The number of rows in the result set is the Cartesian product of these items:

*<Number of instances in each record array> * <Number of physical records>*

With variable-length arrays, you can't calculate the number of rows in the result set unless you know how many instances exist for each array.

Examples

The following example demonstrates how multiple arrays can lead to performance problems arising from large result sets.

1. An employee has these data items:
 - Four dependents
 - Two emergency contacts
 - Three assignments
2. The database has 200 employee records

A query generates $4 * 2 * 3 = 24$ rows for this employee. If all the employee records have an average Cartesian product of 24 based on the data items stored in arrays, then a query of all the employee records yields $24 * 200 = 4800$ rows in the result set.

In the next example, a COBOL copybook has two OCCURS clauses that define arrays that contain data about dependents and professional organizations:

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

If you use the New Table wizard in Classic Data Architect to create a table definition for this record layout, and you map both arrays to the same table, the result set for a single employee record is the Cartesian product of DEPENDENTS-ARRAY and ORGS-ARRAY.

The query processor does not return null array instances. In this example, an employee has four dependents and two professional organizations. A query generates $4 * 2 = 8$ rows for that employee.

Creating a separate table for each record array in a table definition:

You can improve the performance of federated queries that read record array data if you map a separate table for each record array in the table definition.

Before you begin

Before you run the New Table wizard, import a data definition file into your data design project that describes the structure of the data in the source database, such as a COBOL copybook, a PL/I include file, or a CA-IDMS schema with subschemas. If your source is CA-IDMS or Adabas you can connect to the server to retrieve the information directly from the source database.

About this task

Each table you map separately consists of a single record array definition that contains the column definitions unique to a single array instance. Any given column appears in each instance of the array. For example, if your employee table has an array that describes employee dependents, the table structure might look like this:

KEY	DEP_LAST_NAME	DEP_FIRST_NAME	DEP_GENDER	DEP_ID	DEP_DOB
-----	---------------	----------------	------------	--------	---------

You can map a separate table for each array by running the New Table wizard in Classic Data Architect once for each array, or you can copy and edit table objects as described in the following steps.

To map a separate table for each array in a base table by using the Data Project Explorer view:

Procedure

1. Run the New Table wizard once to create a parent table object using the array processing option **Create record array**.
2. Make one copy of the table object for each record array.
3. Delete columns in each copied table object until the columns that remain contain the key information and array data that you want.
4. Optional: Delete the columns that contain array definitions from the parent table if you want to use the non-array data for queries.

Record array definitions for federation and change capture:

Federated queries can read record array data if you create a separate table for each array. You can insert, update, delete, and capture changes to record array data by flattening the column structure.

Record array definitions contain column definitions, and possibly additional record array definitions. To read, manipulate, or capture changes to record array data, create the required tables or column structure.

To query record array data, map a separate table for each record array in the data definition. To insert, update, or delete array data, flatten the array structure by choosing the array processing option **Expand occurrences** when you map the table in the New Table wizard.

A flattened structure provides a separate column for each array instance and field. For example, you map a record array for employee dependents that contains five fields:

1. DEP_LAST_NAME
2. DEP_FIRST_NAME
3. DEP_GENDER
4. DEP_ID
5. DEP_DOB

If you want to support up to ten dependents, you must map 50 columns with names that uniquely identify each instance and field. In this example, the column names range from DEP_LAST_NAME_1 to DEP_DOB_10.

Restriction: You cannot insert, update, or delete array data when you map the structure as an array because of mismatched insert and update logic. Adding a new dependent updates a single record in the source database, but the change inserts a new row into the logical table on the data server. Flatten the structure of the table to perform inserts, updates, or deletes.

Change capture

Flatten the structure of a table to capture changes to array data.

You cannot map record array definitions for change capture. An ALTER TABLE statement on a table containing array structures mapped as arrays will fail, because change capture must send one notification per change. The mapping must return exactly one row per physical record to provide an accurate notification to consuming applications.

Array definition examples:

Use these examples to help you understand COBOL and PL/I array definitions, so that you can map your tables and views correctly.

When you create relational tables or views on your data server, Classic Data Architect uses data definitions that are contained in COBOL copybooks or PL/I include files to generate data definition language (DDL). You then run the DDL on your data server to create user tables in the metadata catalogs.

COBOL examples

Example: COBOL definition of a fixed-length array.

The DEPENDENTS-ARRAY clause defines an array structure of four fields that has a fixed length and repeats exactly 20 times. You can use the NULL-IS parameter in a CREATE TABLE or CREATE VIEW statement to specify a null value for empty array instances.

```
01 EMPLOYEE-RECORD.  
   05 EMP-LAST-NAME           PIC X(20).  
   05 EMP-FIRST-NAME         PIC X(20).  
   05 EMPID                   PIC 9(9).  
   ....  
   05 DEPENDENTS-ARRAY OCCURS 20 TIMES  
       10 DEP-ID              PIC 9(9).
```

```

10 DEP-NAME                PIC X(20).
10 DEP-DOB                 PIC 9(6).
10 DEP-RELATIONSHIP-TO-EMPL PIC X.

```

Example: COBOL definition of a variable-length array.

The array can appear 1 to 20 times, depending on the value of the variable NUMBER-OF-DEPENDENTS specified in the DEPENDING ON clause.

```

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

```

Example: CREATE TABLE statement based on the COBOL copybook.

The DDL maps a subset of the data items:

- Employee ID
- Number of dependents
- Dependent ID
- Dependent name

```

CREATE TABLE CAC.EMPL .....
(
  EMP_ID SOURCE DEFINITION
  DATAMAP OFFSET 40 LENGTH 9 DATATYPE C
  USE AS CHAR(9),
  NUMBER_OF_DEPENDENTS SOURCE DEFINITION
  DATAMAP OFFSET 49 LENGTH 2 DATATYPE H
  USE AS SMALLINT,
  BEGINLEVEL 1 OFFSET 51 LENGTH 36 OCCURS 20
  DEPENDING ON COLUMN NUMBER_OF_DEPENDENTS,
  DEP_ID SOURCE DEFINITION
  DATAMAP OFFSET 0 LENGTH 9 DATATYPE C
  USE AS CHAR(9),
  DEP_NAME SOURCE DEFINITION
  DATAMAP OFFSET 9 LENGTH 20 DATATYPE C
  USE AS CHAR(20),
  ENLEVEL 1
)

```

PL/I samples

Example: A PL/I include file that describes an employee record.

The DIMENSION (DIM) attribute defines a fixed-length array structure that repeats exactly 20 times. You can use the NULL-IS parameter in a CREATE TABLE or CREATE VIEW statement to specify a null value for empty array instances.

```

DCL 1 EMPLOYEE_RECORD BASED,
     5 EMP_LAST_NAME CHAR(20),
     5 EMP_FIRST_NAME CHAR(20),
     5 EMPID PIC '(9)9',
     ....
     5 DEPENDENTS_ARRAY DIM(20)

```



```

10 DEP_ID PIC '(9)9',
10 DEP_NAME CHAR(20),
10 DEP_DOB PIC '(6)9',
10 DEP_RELATIONSHIP_TO_EMPL CHAR(1);

```

Example: A PL/I include file that defines a variable-length array.

The PL/I DIM attribute specifies that the number of dependents determines the number of array instances. The array can appear 1 to 20 times, depending on the value of the variable NUMBER_OF_DEPENDENTS specified in the REFER attribute.

```

DCL 1 EMPLOYEE_RECORD BASED,
    5 EMP_LAST_NAME CHAR(20),
    5 EMP_FIRST_NAME CHAR(20),
    5 EMPID PIC '(9)9',
    ....
    5 NUMBER_OF_DEPENDENTS BIN FIXED(15),
    5 DEPENDENTS_ARRAY DIM(N1 REFER(NUMBER_OF_DEPENDENTS)),
    10 DEP_ID PIC '(9)9',
    10 DEP_NAME CHAR(20),
    10 DEP_DOB PIC '(6)9',
    10 DEP_RELATIONSHIP_TO_EMPL CHAR(1);

```

Example: CREATE TABLE statement based on a PL/I include file.

The DDL maps a subset of the data items:

- Employee ID
- Number of dependents
- Dependent ID
- Dependent name

The data server calculates the number of array instances automatically when you run the DDL on the data server to create the table. Nevertheless, restrictions on mapping columns that appear in the structure after variable-length arrays still apply.

```

CREATE TABLE "AA"."EMPLOYEE_RECORD" DBTYPE SEQUENTIAL
DS "D"
(
    "EMPID" SOURCE DEFINITION
    DATAMAP OFFSET 40 LENGTH 9
    DATATYPE C
    USE AS CHAR(9),
    "NUMBER_OF_DEPENDENTS" SOURCE DEFINITION
    DATAMAP OFFSET 49 LENGTH 2
    DATATYPE H
    USE AS SMALLINT,
    BEGINLEVEL 1 OFFSET 51 LENGTH 0
    OCCURS DEPENDING ON COLUMN "NUMBER_OF_DEPENDENTS",
    "DEP_ID" SOURCE DEFINITION
    DATAMAP OFFSET 0 LENGTH 9
    DATATYPE C
    USE AS CHAR(9),
    "DEP_NAME" SOURCE DEFINITION
    DATAMAP OFFSET 9 LENGTH 20
    DATATYPE C
    USE AS CHAR(20),
    ENLEVEL 1 );

```

Creating views on existing tables

You can create a view on a table that already exists in a data design project, with the SQL builder, with the SQL editor, or with the Properties view.

Creating views on existing tables with the SQL builder

To create a view on a table that already exists either only in your project or also in a metadata catalog, you can use the SQL builder, a graphical utility that builds the SELECT statement for the view for you. After you create the SELECT statement, you can add the view to a schema in your project.

Before you begin

Restrictions

If you are creating a view for change capture, these restrictions apply:

- The view cannot reference more than one table. This includes tables in the FROM clause or the WHERE clause as in the case of sub-selects.
- The view cannot reference another view.
- The view must reference all of the columns in the base table.
- The base table must not map to record arrays.

Procedure

1. In the Data Project Explorer, expand the physical data model in which you are working. Expand the database in which you are working. Right-click the **SQL Statements** folder and select **New SQL Statement**.
2. In the New SQL Statement window, follow these steps:
 - a. Ensure that SELECT is selected in the **Statement template** field.
 - b. Give the statement a descriptive name.
 - c. Ensure that the **SQL builder** radio button is selected.
 - d. Click **OK** to open the SQL builder.

The title that appears for the SQL builder is the name that you gave to the SELECT statement. For example, if your SELECT statement is named TEST, the title for the SQL builder is TEST.
3. In the SQL builder, add the tables on which to base the SELECT statement for your view. You can add a table in one of two ways:
 - Right-click the middle section of the SQL builder and select **Add Table**. In the Add Table window, select a table to add to the SQL builder and click **OK**.
 - Left-click one of the tables in the schema in which you are creating the view and drag the table to the middle section of the SQL builder.
4. Build the SELECT statement for the view. For help building a SELECT statement, press F1 while in the SQL builder and follow the link to the online help for the SQL builder.
5. Optional: Test the SELECT statement. The tables that are referenced by the view must already exist on the data server. To test the SELECT statement, right-click the statement and select **Run SQL**. Look in the Data Output view to see whether the statement ran successfully.
6. In the Data Explorer view, generate and name the view::
 - a. In the **SQL Statements** folder of your physical data model, right-click the SELECT statement and select **Generate > View**. The view appears in the same schema as the tables that it references.

- b. Click the name of the view once, pause, then click it again to highlight the name. Give the view the name that you want.
7. Select the view and use the Privileges page of the Properties view to grant privileges on the view.
8. Optional: Generate the DDL for the view. Right-click the view and select **Generate DDL** to open the Generate DDL wizard. With this wizard, you can generate the SQL DDL to define the view, and you can choose to run the DDL on a data server so that the view is created in the metadata catalog for that data server. You can also edit the generated DDL before you run it.

After you run the DDL, the view appears on the data server in the Data Source Explorer. To see the view, expand the data server and then expand **Schemas > the schema of the view > Views**.

If you want to generate and run the DDL for more than one object at a time, you can right-click a schema and select **Generate DDL**. The Generate DDL wizard will generate the DDL for all of the objects in the schema.

9. Optional: If you created the view on the data server, run a test query on the view.
 - a. In the Data Source Explorer, right-click the view and select **Data > Sample Contents**.
 - b. Look in the Data Output view to see the results of the test query.

Creating views on existing tables with the SQL editor

To create a view on a table that already exists either only in your project or also on in a metadata catalog, you can use the SQL editor, a text editor that lets you write the SELECT statement for the view. After you create the SELECT statement, you can add the view to a schema in your project.

Before you begin

Restrictions

If you are creating a view for change capture, these restrictions apply:

- The view cannot reference more than one table. This includes tables in the FROM clause or the WHERE clause as in the case of sub-selects.
- The view cannot reference another view.
- The view must reference all of the columns in the base table.
- The base table must not map to record arrays.

Procedure

1. In the Data Project Explorer, expand the physical data model in which you are working. Expand the database in which you are working. Right-click the **SQL Statements** folder and select **New SQL Statement**.
2. In the New SQL Statement window, follow these steps:
 - a. Ensure that **SELECT** is selected in the **Statement template** field.
 - b. Give the statement a descriptive name.
 - c. Ensure that the **SQL editor** radio button is selected.
 - d. Click **OK** to open the SQL editor.
3. Write the SELECT statement for the view.

4. Optional: Test the SELECT statement. The tables that are referenced by the view must already exist on the data server. To test the SELECT statement, right-click the statement and select **Run SQL**. Look in the Data Output view to see whether the statement ran successfully.
5. In the Data Explorer view generate and name the view:
 - a. In the **SQL Statements** folder of your physical data model, right-click the SELECT statement and select **Generate > View**. The view appears in the same schema as the tables that it references.
 - b. Click the name of the view, pause, then click again to highlight the name. Give the view the name that you want.
6. Select the view and use the Privileges page of the Properties view to grant privileges on the view.
7. Optional: Generate the DDL for the view. Right-click the view and select **Generate DDL** to open the Generate DDL wizard. With this wizard, you can generate the SQL DDL to define the view, and you can choose to run the DDL on a data server so that the view is created in the metadata catalog for that data server. You can also edit the generated DDL before you run it.
 After you run the DDL, the view appears on the data server in the Data Source Explorer. To see the view, expand the data server and then expand **Schemas > the schema of the view > Views**.
 If you want to generate and run the DDL for more than one object at a time, you can right-click a schema and select **Generate DDL**. The Generate DDL wizard will generate the DDL for all of the objects in the schema.
8. Optional: If you created the view on the data server, run a test query on the view.
 - a. In the Data Source Explorer, right-click the view and select **Data > Sample Contents**.
 - b. Look in the Data Output view to see the results of the test query.

Creating views on existing tables with the Properties view

To create a view on a table that already exists either only in your project or also on in a metadata catalog, you can create an empty view in your project and then use the Properties view to create the SELECT statement.

Before you begin

Restrictions

If you are creating a view for change capture, these restrictions apply:

- The view cannot reference more than one table. This includes tables in the FROM clause or the WHERE clause as in the case of sub-selects.
- The view cannot reference another view.
- The view must reference all of the columns in the base table.
- The base table must not map to record arrays.

Procedure

1. In the Data Project Explorer, expand the physical data model in which you are working. Expand the database in which you are working. Right-click the schema in which you want to create the view and select **Add Classic Object > View**. In the Data Project Explorer, a view is created under the schema.
2. Name the view.

3. Select the view, and on the SQL page of the Properties view, type the SELECT statement.
4. On the Privileges page of the Properties view, grant privileges on the view.
5. Optional: Generate the DDL for the view. Right-click the view and select **Generate DDL** to open the Generate DDL wizard. With this wizard, you can generate the SQL DDL to define the view, and you can choose to run the DDL on a data server so that the view is created in the metadata catalog for that data server. You can also edit the generated DDL before you run it.
After you run the DDL, the view appears on the data server in the Data Source Explorer. To see the view, expand the data server and then expand **Schemas > the schema of the view > Views**.
If you want to generate and run the DDL for more than one object at a time, you can right-click a schema and select **Generate DDL**. The Generate DDL wizard will generate the DDL for all of the objects in the schema.
6. Optional: If you created the view on the data server, run a test query on the view.
 - a. In the Data Source Explorer, right-click the view and select **Data > Sample Contents**.
 - b. Look in the Data Output view to see the results of the test query.

Viewing and modifying objects for Classic federation

You can view and modify the properties of the different objects that you create in Classic Data Architect for Classic federation. You can also change the selection of columns in tables and change the path of records for CA-IDMS tables.

View and modify the properties of tables, columns, indexes, views, and stored procedures

When you click on an object in the Data Project Explorer, pages that describe the attributes of the object appear in the Properties view.

Change the selection of columns in tables

For all tables, use the Change Column Selection wizard to replace columns that exist in a table or to append new columns to a table.

To open this wizard, right-click a table and select **Modify Table > Update Columns**.

Change the path of records and sets in a CA-IDMS table and change the name of the table

Open the Modify CA-IDMS Table wizard by right-clicking a CA-IDMS table and selecting **Modify Table > Modify Table**.

Column properties

Column properties are shown in the Properties view. You can use the Properties view only to view the properties of a column. You cannot modify any of the properties.

The Properties view for a column contains the following information.

General page

Displays the name of the column.

Type page

Displays the SQL data type that is assigned to the column.

Classic Column Info

Displays the SQL data type that is assigned to the column and the position and length of the column.

Classic Array Info

If the column participates in an array, this page displays information about all arrays the column belongs to.

Documentation

Lets you add comments to a column.

Database properties

Use the Properties view to view or modify information about the properties of a database in a data design project.

You can view settings in the Data Source Explorer and modify them in a physical data model in the Data Project Explorer. Select the icon for the database and properties that you want to manage.

General page

This page displays the name of the database, as well as the type of the database and its version. The type is Classic Integration.

Privileges page

This page lists the role-based privileges that apply to different grantees. The following information is provided for each grantee:

Type The catalog database object to which you are applying security, such as SYSTEM or \$VSAM.

Privilege

The ID associated with the authority level that you are granting, such as DISPLAY, SYSADM, or DBADM.

Grantable

If checked, indicates that the grantee can grant the privilege to others.

Grantor

The ID of the user or role that grants the privilege for the specified type.

Revoke

If checked, indicates that the grantee can revoke the privilege from others.

Documentation page

Optionally, this page provides additional information about the database.

Annotation page

This page, which is found only in the Data Project Explorer, enables you to add property and value pairs.

Index properties

Index properties are shown in the Properties view. You can use the Properties view to view or modify the properties of an index.

If the index already exists in a metadata catalog on a data server and you want any changes that you make to the index to be reflected in the metadata catalog, you must follow these steps:

1. Drop the index from the metadata catalog. You can generate the DDL to drop the index by right-clicking the index and selecting **Generate DDL**. In the Generate DDL wizard, select the **DROP statements** check box.
2. Run the generated DDL on the data server.
3. Make your changes to the index.
4. Generate the DDL to create the index. You can generate this DDL by opening the Generate DDL wizard and selecting the **CREATE statements** check box.
5. Run the DDL on the data server.

The Properties view for an index contains the following information:

General page

Allows you to change the name of the index.

If the index is an alternate index for a CICS VSAM table, the **FCT name** field displays the name of the CICS table that contains the information used by CICS file control for accessing the VSAM data set.

In the index is an alternate index for a VSAM table, these controls appear:

DS Specifies that the information from which to create the index is contained in a data set.

DD Specifies that the information from which to create the index is contained in a data set with a DD name.

Name Type the name of the data set or DD card in which the information for the index is located.

Details page

Lists the columns that are in the index. You can modify this list.

PCB selection page (for IMS only)

Displays the method that Classic federation will use to select PCBs for accessing the index.

Documentation

Lets you add comments to the index.

Stored procedure properties

Stored procedure properties are shown in the Properties view. You can use the Properties view to modify the properties of a stored procedure.

If the stored procedure already exists in a metadata catalog on a data server and you want the changes that you make to the stored procedure to be reflected in the metadata catalog, you must follow these steps:

1. Drop the stored procedure from the metadata catalog. You can generate the DDL to drop the stored procedure by right-clicking the stored procedure and selecting **Generate DDL**. In the Generate DDL wizard, select the **DROP statements** check box.
2. Run the generated DDL on the data server.
3. Make your changes to the stored procedure.
4. Generate the DDL to create the stored procedure. You can generate this DDL by opening the Generate DDL wizard and selecting the **CREATE statements** check box.

5. Run the DDL on the data server.

The Properties view for a stored procedure contains the following information:

General page

Property	Description
Name	<p>Type the name of the stored procedure. The name cannot be a single asterisk, even if you specify it as a delimited identifier ("*").</p> <p>The name is implicitly or explicitly qualified by a schema. The name, including the implicit or explicit qualifier, must not identify an existing stored procedure at the current server.</p> <ul style="list-style-type: none"> The unqualified form of a procedure name is an SQL identifier. The unqualified name is implicitly qualified with a schema name according to the following rules: <p>If the statement is embedded in a program, the schema name is the authorization ID in the QUALIFIER bind option when the plan or package was created or last rebound. If QUALIFIER was not specified, the schema name is the owner of the plan or package.</p> <p>If the statement is dynamically prepared, the schema name is the SQL authorization ID in the CURRENT SQLID special register.</p> The qualified form of the procedure name is an SQL identifier (the schema name) followed by a period and an SQL identifier. The schema name can be 'SYSIBM' or 'SYSPROC'. It can also be 'SYSTOOLS' if you have SYSADM or SYSCtrl privileges. Otherwise, the schema name must not begin with 'SYS' unless the schema name is 'SYSADM'. <p>The owner of the procedure is determined by how the CREATE PROCEDURE statement is invoked:</p> <ul style="list-style-type: none"> If the statement is embedded in a program, the owner is the authorization ID of the owner of the plan or package. If the statement is dynamically prepared, the owner is the SQL authorization ID in the CURRENT SQLID special register.
Label	Type a label. This label is displayed in visual diagrams, if you use visual diagrams.
Result set	Specifies the maximum number of query result sets that the stored procedure can return. The default is 0, which indicates that there are no result sets. The value must be 0 or 1.
Language	<p>Specify the language interface convention to which the procedure body is written:</p> <p>Assembler indicates that the stored procedure is written in Assembler.</p> <p>C indicates that the stored procedure is written in C or C++.</p> <p>COBOL indicates that the stored procedure is written in COBOL.</p>
Parameter style	Specify the convention to use for passing parameters to and returning the value from procedures.

Property	Description
External name	Type the name of the load module that exists on the data server for loading the stored procedure.
Deterministic	Specifies whether the stored procedure is deterministic or nondeterministic.

Parameters page

Shows the parameters for the stored procedure. You can add or remove parameters.

Source page

This page is not supported by Classic Data Architect.

Privileges page

Shows the privileges for the stored procedure. You can grant or revoke the EXECUTE privilege on the stored procedure.

Documentation

Lets you add comments to a stored procedure.

Table properties

Table properties are shown in the Properties view within Classic Data Architect. You can use the Properties view to modify the properties of a table.

If the table already exists in a metadata catalog on a data server and you want any changes that you make to the table to be reflected in the metadata catalog, you must follow these steps:

1. Drop the table from the metadata catalog. You can generate the DDL to drop the table by right-clicking the table and selecting **Generate DDL**. In the Generate DDL wizard, select the **DROP statements** check box.
2. Run the generated DDL on the data server.
3. Make your changes to the table.
4. Generate the DDL to create the table. You can generate this DDL by opening the Generate DDL wizard and selecting the **CREATE statements** check box.
5. Run the DDL on the data server.

The Properties view for a table contains the following information:

- General page
- Columns page
- Source information page
- Source columns page
- Path information page
- Source elements page
- Source fields page
- Segments page
- PCB selection page
- Privileges page
- Documentation page

General page

Property	Description
Name	Type the name of the table.
Schema	Displays the schema in the two-part name for the table.
Source DBMS	Displays the type of DBMS in which the source data is located.
Change capture	(For all data sources except CA-Datcom and DB2 for z/OS) Changes Select this value if you want to use the table for change capture. None Select this value if you do not want to use the table for change capture.
XM URL	For a native VSAM table that is being used for change capture, the name of the data space and the name of the cross-memory (XM) queue to use. The change-capture agent that is capturing changes to the native VSAM table writes change data to this XM queue. The format of the XM URL is <i>XM1/name_of_data_space/name_of_queue</i>

Columns page

Lists the columns of the table.

Source information page

The Source information page contains source information for:

- Adabas, Table 15
- CA-Datcom, Table 16
- CA-IDMS, Table 17 on page 135
- CICS VSAM, Table 18 on page 136
- DB2 for z/OS, Table 19 on page 137
- IMS, Table 20 on page 137
- sequential files and native VSAM data sets, Table 21 on page 138

Table 15. Source information for Adabas

Property	Description
File DBID	Optional: Type the identifier of the database in which the Adabas file is stored. This Adabas file is either the file that is identified in the File number field or the file that is referenced by the Predict view. The default value is 0. The identifier must be between 1 and 65535.
View name	Type the name of the Predict view that describes the contents of an Adabas file that has fields that you want to map to columns. Classic Data Architect retrieves the Adabas Field Description Table (FDT) information for the Adabas file that is referenced by the view. If you want Classic Data Architect to access an Adabas file's FDT directly, do not provide a view name. Instead, provide the number of the Adabas file in the File number field.

Table 16. Source information for CA-Datcom

Property	Description
Table name	Type an identifier of 1 to 32 characters for the CA-Datcom table that the Classic table definition references. The name follows CA-Datcom/DB entity naming conventions.

Table 16. Source information for CA-Datcom (continued)

Property	Description
Status/Version	Select or type the status and version of the CA-Datcom table that contains the elements that you want to map to. The status and version can contain explicit values of TEST, PROD, HIST, or a value that begins with a T or H followed by a three digit number.
URT name	<p>Type the name of the User Requirements Table (URT) that is used to access the CA-Datcom table that contains the elements that you want to map to. The name must exist in a data set that is referenced by the servers STEPLIB DD statement or reside in the link pack area. The name follows z/OS load module naming conventions.</p> <p>A URT must be provided on every request for service sent to CA-Datcom. Every service request is validated against an open User Requirements Table. This technique provides security (by restricting access) and efficient allocation of CA-Datcom resources. When you define your User Requirements Tables, consider the security implications. You must decide whether you want to have one User Requirements Table per CA-Datcom table that you map into the metadata catalog or have only a few User Requirements Tables for all CA-Datcom tables that you map into the metadata catalog. If you define only a few User Requirements Tables, you will have more relaxed security.</p>

Table 17. Source information for CA-IDMS

Property	Description
Subschema name	Displays the name of the subschema that was obtained through a remote connection to the CA-IDMS database or from the local subschema file that you specified.
Schema name	Displays the name of the schema that was obtained through a remote connection to the CA-IDMS database or from the local schema file that you specified.
Schema version	Type a valid 4-digit integer between 0 and 9999 that, together with the schema name, uniquely identifies a CA-IDMS schema. The schema version follows CA-IDMS schema version naming conventions.
Data dictionary	Type an identifier of 1 to 8 characters for the CA-IDMS database for the dictionary that contains the schema and subschema definitions. The data server binds to this dictionary to gather information from the schema and subschema when the data server creates the logical table. The identifier follows CA-IDMS database naming conventions.
Data database	Type an identifier of 1 to 8 characters for the CA-IDMS database that contains the user data that the data server will access at runtime.
Access load module	Type an identifier of 1 to 8 characters for the CA-IDMS batch access module to be used to communicate with the CA-IDMS central version that hosts the user data. The CA-IDMS identifier follows z/OS load module naming conventions.

Table 17. Source information for CA-IDMS (continued)

Property	Description
VSAM information	RRDS Specifies that a record in the subschema has a mode of VSAM and is not a member of a VSAM index set.
	KSDS Specifies that a record in the subschema either has a mode of VSAM and is a member of a VSAM key-sequenced data set, or has a mode of VSAM CALC.
	ESDS Specifies that a record in the subschema either has a mode of VSAM and is a member of a VSAM entry-sequenced data set, or has a mode of VSAM CALC.

Table 18. Source information for CICS VSAM

Property	Description
FCT name	Type the name of the CICS file name entry for the VSAM data set.
Local APPLID	<p>Type the name, or VTAM APPLID, that identifies the application program that the data server uses to communicate with its partner CICS transaction. The LUNAME follows VTAM naming conventions, and has a maximum of eight characters.</p> <p>You specify the APPLID in the ACBNAME parameter of a macro definition that the VTAM administrator uses in the VTAM configuration. See the sample definitions in SCACSAMP member CACCAPPL. The sample LUNAME values are CACCICS1 and CACCICS2, and you can modify them.</p> <p>The LUNAME must be active on the image where the data server is running. Issue the following operator command to determine if the LUNAME is active:</p> <pre>D NET, ID=<APPLID></pre> <p>You must also define the LUNAME to CICS in the NETNAME parameter of a CONNECTION definition.</p>
CICS APPLID	Type the name of the VTAM APPLID for the CICS region to which you are connecting. The CICS APPLID must match the value of the APPLID parameter in the system initialization table (DFHSIT macro) of the CICS subsystem that hosts the target VSAM data sets. This identifier follows VTAM naming conventions, and has a maximum length of eight-characters.
Logmode	Type the name of the VTAM logon mode table that controls session parameters for the VTAM conversation between the local LU and the CICS LU. The maximum length of this identifier is eight characters. The logon mode table is a z/OS load module that is accessible to VTAM. See the sample definition in SCACSAMP member CACCMODE.
Transaction ID	Type the name of the partner CICS transaction that the data server uses to validate the relational tables that you create to map the data. The maximum length of this identifier is four characters. The CICS transaction ID must match the value of the TRANSACTION parameter in the transaction definition. See the sample transaction definition in SCACSAMP member CACCDEF. The sample CICS transaction ID is EXV1, and you can modify it.

Table 18. Source information for CICS VSAM (continued)

Property	Description
Record exit	<p>Optional: Use the exit name and length parameters to pass control to a record exit. You can use record exits only for Classic federation.</p> <p>Type a short native identifier for the name of a record processing exit to invoke to decompress sequential records when the file is accessed. The exit must either exist in a data set that is referenced by the server's STEPLIB DD statement or reside in the link pack area. The exit name follows z/OS load module naming conventions.</p>
Maximum length	Type the maximum length (in bytes) of the buffer that is needed by the record exit to decompress a record.
Network name	Type the name of the network that hosts the CICS subsystem where you access VSAM data sets. The network name follows VTAM naming conventions, and has a maximum length of eight characters. The network name must match the value in the NETWORK VTAM macro definition on the local image, which identifies the remote SNA network.

Table 19. Source information for DB2 for z/OS

Property	Description
Creator	Displays the schema of the table.
Table	Displays the name of the table.
Subsystem ID	Displays the ID of the DB2 subsystem in which the table is located.
Plan	Displays the name of the DB2 application plan.
Type	<p>Displays the type of object that the new table is mapped to.</p> <p>Accessing DB2 data requires binding an application plan for use by the DB2 Call Attach Facility (CAF) service. You can give the plan whatever name you want based on site-installation standards.</p> <p>When you import a DB2 table into Classic Data Architect, you are creating a table that you can create in a metadata catalog. This field shows that the table is mapped to a DB2 table.</p>

Table 20. Source information for IMS

Property	Description
DBD name	Displays the name of the IMS DBD (database definition) that the table references.
DBD type	Displays the name of the IMS DBD (database definition) that the table references.
Leaf segment	Displays the name of the leaf segment.
Index root	<p>Optional: Type a name for either of these two objects:</p> <ul style="list-style-type: none"> • The physical or logical root segment of the IMS database that is identified by the DBD. • The perceived root segment of the IMS database of a secondary data structure that is created by a secondary index definition that exists in the DBD. <p>The default index root is the root segment of the physical or logical database that is referenced by the DBD.</p>

Table 20. Source information for IMS (continued)

Property	Description
IMS subsystem	<p>Optional: Type the 4-character name for the IMS subsystem that is used by the ODBA interface to access the IMS database that is identified by the DBD. The IMS subsystem ID is used only when the server is operating in an RRS two-phase commit environment. The IMS subsystem ID follows IMS naming conventions for subsystem identifiers.</p> <p>The IMS subsystem ID must correspond to the value that is specified on the IMSID parameter on the IMSCTRL macro in the system definition of the target online IMS subsystem that is used to access or update the IMS data.</p> <p>The IMS subsystem ID value is ignored for other forms of IMS data access (DRA or BMP/DBB/DLI) and when the table mapping is used for change capture.</p>
PSB name	<p>Optional: Type the name of the PSB that is scheduled to access the IMS database that is identified by the DBD. This name is used if you are using a DRA or ODBA interface to access IMS data. The standard PSB corresponds to a PSB definition that is defined to the IMS online system that is being accessed. The PSB also corresponds to a PDS member under the same name in the active ACB library of the source IMS subsystem. The standard PSB name follows z/OS load module naming conventions.</p>
Join PSB name	<p>Optional: Type the name of the PSB that is scheduled to access the IMS database that is identified by the DBD. The name is used if you are using a DRA or ODBA interface to access IMS data. The JOIN PSB corresponds to a PSB definition that is defined to the IMS online system that is being accessed. The PSB also corresponds to a PDS member under the same name in the active ACB library of the target IMS subsystem. The JOIN PSB name follows z/OS load module naming conventions. The JOIN PSB is scheduled when an SQL SELECT statement is executed that contains a JOIN predicate that references multiple IMS tables and this is the first table referenced in the JOIN.</p>

Table 21. Source information for sequential files and native VSAM data sets

Property	Description
DS	Specifies that the information from which to create the table is contained in a data set.
DD	Specifies that the information from which to create the table is contained in a data set with a DD name.
Name	Type the name of the data set or DD card in which the information for the table is located.
Record exit	Type the name of a record processing exit that is invoked to decompress sequential records when the file is accessed. The exit must exist in a data set that is referenced by the servers STEPLIB DD statement or reside in the link pack area. The exit name follows z/OS load module naming conventions.
Maximum length	Type the maximum length (in bytes) of the buffer that is needed by the record exit to decompress a record.

Source columns page

Lists the columns in the table.

Path information page (for CA-IDMS only)

Lists the records and sets with elements that are mapped to columns in the table.

Source elements page (for CA-IDMS only)

Lists the elements that are mapped to columns in the table.

Source fields page (for IMS only)

Lists the fields that are mapped to columns in the table.

Segments page (for IMS only)

Lists the segments that contain fields that are mapped to columns in the table.


PCB selection page (for IMS only)

Displays the method that Classic federation will use to select PCBs for accessing the table.

Privileges page

Lists that privileges that are granted on the table. Click the add button (



) to add privileges. Click the delete button () to remove privileges.

Documentation page

Lets you add comments to the table.

View properties

View properties are shown in the Properties view. You can use the Properties view to display and modify the properties of a view.

If the view already exists in a metadata catalog on a data server and you want any changes that you make to the view to be reflected in the metadata catalog, you must follow these steps:

1. Drop the view from the metadata catalog. You can generate the DDL to drop the view by right-clicking the view and selecting **Generate DDL**. In the Generate DDL wizard, select the **DROP statements** check box.
2. Run the generated DDL on the data server.
3. Make your changes to the view.
4. Generate the DDL to create the view. You can generate this DDL by opening the Generate DDL wizard and selecting the **CREATE statements** check box.
5. Run the DDL on the data server.

The Properties view for a view contains the following information:

General page

Name Displays the name of the view in an editable field.

Schema

Displays the schema that contains the view.

Change capture

Sets the DATA CAPTURE flag on the view. This field is available only if the view meets all three of these criteria:

- The view references only one table.
- The view references all of the columns in the base table.
- The view references an Adabas, CA-IDMS, CICS VSAM, IMS, or native VSAM table.

CHANGES

Specifies to capture changes that are made to the data that the view references.

NONE

Specifies not to capture changes that are made to the data that the view references.

Columns page

Lists the columns that are referenced in the view.


SQL page

Displays the SELECT statement for the view in an editable field. Click the **Validate** button to check the statement for syntax errors.

Privileges page

Lists that privileges that are granted on the view. Click the add button (



) to add privileges. Click the delete button () to remove privileges.

Documentation page

Lets you add comments to the view.

Adding or replacing columns in tables based on data definition files

Use the Append Column wizard to add or replace columns in user tables that are based on data definition files, such as COBOL copybooks or PL/I include files.

Before you begin

The data definition file that contains the columns that you want to use must be in the appropriate folder in your project, for example, the **COBOL Copybooks** folder.

About this task

You can use columns from the data definition file on which the table is based, or you can use columns from a different file.

Procedure

1. Right-click the table and select **Modify Table > Update Columns**.
2. On page one of the wizard, select the data definition file that contains the columns that you want to use.
If the table is for an IMS data source, select the segment for the columns that you either want to add columns to or that you want to replace with different columns.
3. On page two of the wizard, select the data that you want to map to new columns.
4. On the summary page, verify that the table contains the columns that you want. Click **Finish** to generate the updated model for the table.

Modifying the selection of records in tables for CA-IDMS databases

Use the Modify CA-IDMS Table wizard to change the selection of records in an existing table before the DDL for the table is run on a data server.

Before you begin

The subschema and schema reports that you select must be identical to the reports that you used when you created the table. However, the names of the files that contain those reports can be different from the names of the files that you originally used.

You can provide the information on which to base the logical table in one of two ways:

- You can import schema and subschema files that were punched from the CA-IDMS dictionary and transferred via FTP to your workstation. These files must be located in the **CA-IDMS References** folder of your data project.
- You can tell Classic Data Architect to obtain the schema information that is associated with all records, sets, and areas that are listed in the required subschema directly from the CA-IDMS dictionary.

You produce CA-IDMS schema and subschema reports by running the CA-IDMS schema and subschema compilers and capturing the punched output into a z/OS data set. Sample JCL that you can use to punch these reports is in member CACIDPCH of the SAMPLIB data set.

Procedure

1. Open the Modify CA-IDMS Table wizard by right-clicking the logical table that you want to modify and selecting **Modify CA-IDMS table**.
2. In the wizard, modify the selection of records:
 - a. On the first page, verify that Classic Data Architect is getting the schema and subschema information from the correct location. If you are using local files that contain the subschema and schema reports and you want to use different files, browse your **CA-IDMS References** folder for the new files. The subschema and schema reports in those files must be identical to the reports that you used when you created the table.
 - b. On the second page, you can modify the information that helps the data server locate the data structures in your database, and you can change the way that the table will be used.
 - c. On the third page, you can rename the table. You can also modify the path of up to ten records and sets from which you want to choose the elements that will constitute the columns in your table.
 - d. Complete a separate wizard page for every record and set that you include in the path to select the elements that you want to map to columns in the table.
 - e. On the summary page, verify that the table contains the columns that you want. Click **Finish** to generate the model for the table.

Populating metadata catalogs

Classic Data Architect can generate the SQL DDL statements that describe the tables, views, stored procedures, and other objects that you create.

After the DDL statements are generated, you can run them from Classic Data Architect, or you can export them to the z/OS system where your data server is located and run the DDL using the metadata utility.

Generating DDL

When you finish designing your objects, you generate the DDL that you use to promote those objects to a metadata catalog on a data server.

Before you begin

If you choose to run the DDL after it is generated, you must first do the following tasks:

- Open a connection to a data server.
- Create a set of metadata catalogs on the data server.
- Set up connectivity from the data server to your data sources.

About this task

When the DDL is generated, you can choose to run it on a data server. You can also choose to open the DDL in an editor.

If you do not choose to run the DDL immediately after it is generated, you can run it later by opening the **SQL Scripts** folder, right-clicking the file with the DDL, and selecting **Run SQL**.

Procedure

1. Open the Generate DDL wizard in either of these two ways:
 - Right-click the schema in which the objects are located and select **Generate DDL**. You can choose which objects in the schema you want to generate DDL for.
 - Right-click the object that you want to generate DDL for and select **Generate DDL**.
2. Follow the pages of the wizard to make these selections:

Which DDL statements to generate

You can generate ALTER, COMMENT ON, CREATE, DROP, and GRANT statements. You can also choose whether to use fully qualified names and quoted identifiers.

Which objects to generate DDL for

The available objects depend on which object you right-clicked to open the Generate DDL wizard.

Where to create the file, which statement terminator to use, whether to run the DDL on a data server, and whether to open the DDL file for editing

The page on which you make these choices displays the DDL that will be generated.

3. After reviewing your settings, click **Finish**.

Exporting SQL to remote z/OS hosts

The DDL that you generate for your tables, views, and stored procedures is saved to the SQL Scripts folder of your project. You can export the files that contain those scripts to the z/OS host where your data server is located.

Procedure

1. Open the Export SQL window by right-clicking the file that you want to export and selecting **Export SQL**.
2. Provide the connectivity information for connecting to the remote z/OS host, and provide the location in which you want to save the DDL scripts.

Results

Then, you can use the metadata utility to run the scripts and populate a metadata catalog.

Configuring communications between data servers and clients

This section describes the communication configuration options and data server configuration options that are required for cross-memory services, IBM WebSphere MQ, and TCP/IP communications.

Communication between data servers and client applications

A data server can accept connections from both local and remote client applications. Configuration parameters on the data server and on the client provide for the connections.

The following parameters are required to configure a connection between a client application and a data server:

- DATASOURCE configuration parameter
This parameter specifies two subparameters:
 - Data source name: Identifies which query processor in the data server handles requests for all client applications connecting to the query processor by means of the information in the second subparameter. The data server master configuration file must include a service definition for the query processor. The service name specified for the query processor must match the data source name.
 - Communications compound address: Establishes a communications path to the data server that contains the query processor to which the client connects. The data server configuration must include a service definition for a connection handler. This connection handler service must specify this compound address in the COMMSTRING configuration parameter. A data server configured in this way listens for connections on that address. A client application then connects to that address and a communication session results.
- The data server must have a query processor service of class QP or QPRR defined.
- The data server must have a connection handler service of class INIT defined.

Configuring data servers to use Cross Memory to communicate with local client applications

If a client application accesses a data server locally, the client application and the data server have the option to communicate by means of Cross Memory. The default communication method is TCP/IP.

Before you begin

You need to know the service names of the query processors that will handle the queries issued by the client applications.

A client application and a data server can also communicate by means of TCP/IP.

Procedure

1. Update the definition for the connection handler service to change the `COMMSTRING` parameter to use XM as the communications protocol. The `COMMSTRING` parameter consists of the following information:
 - Protocol identifier: The Cross Memory protocol identifier, XM1.
 - Data space name: The name of the data space to use. The name can contain up to four characters.

Each data space can support up to 400 concurrent client applications, although in practice this number might be lower due to resource limitations. To support a larger number of client applications on a data server, configure multiple connection handler services, each with a different data space name.
 - Queue: The name of the queue to use. The name can contain up to four characters.
 - Data space size: The size, in megabytes, of the cross-memory queue that uses the data space.

Example:

```
SET,CONFIG,SERVICE=INIT, COMMSTRING='XM1/CAC/CAC'
```

2. Update the `DATASOURCE` parameter for the client configuration to use XM as the communications protocol. You can use the `USERHLQ.USERCONF(CACINIZ)` member as an example. Change the `DATASOURCE` parameter as follows to use the same XM definition used on the connection handler `COMMSTRING` parameter:

```
DATASOURCE = CACSAMP XM1/CAC/CAC
```

3. Activate the connection by using one of the following methods:
 - Stop and restart the data server.
 - Stop and restart the connection handler service.

TCP/IP for communication between data servers and client applications

TCP/IP communication requires that you define the IP address of the TCP/IP communications stack that the data server is running on and specify a listen port number, in addition to the TCP/IP protocol identifier TCP.

Multiple sessions are created on the specified port. The number of sessions carried over the port is the number of concurrent users to be supported plus one for the listen session that the connection handler uses to accept connections from remote clients. If the TCP/IP implementation that you are using requires the specification of the number of sessions that can be carried over a single port, you must ensure that the proper number of sessions are defined. If you do not define the correct number of sessions, client applications will not be able to connect to the data server after the defined number of connections are active. A single TCP/IP connection handler service can accept connections from 2048 concurrent users.

You specify the IP address and port number on the `COMMSTRING` parameter for the definition of the connection handler service.

If you specify an incorrect IP address, the connection handler service fails during initialization. If you specify a port number that is assigned to another application, unpredictable results occur for both the data server and the application that is using the port.

Configuring data servers to use TCP/IP to communicate with client applications

You can configure the data server to support TCP/IP for communications with local z/OS client applications or remote client applications.

Before you begin

You need to know the service names of the query processors that will handle the queries issued by the client applications.

The port number or service name must not be in use by any other application, and should be greater than 1024. The sample configuration members use port number 5001. If this port is assigned to another application, you need to change the sample configuration members to reference the port number that your network administrator assigned for Classic federation.

About this task

A z/OS system can support multiple TCP/IP stacks and multiple IP addresses. It is important that you validate your selected IP address with your network administrator.

The sample configuration members use the 0.0.0.0 IP address notation that informs TCP/IP to resolve this address to the local host IP address where the data server is running.

Procedure

1. Update the definition for the connection handler service to change the COMMSTRING parameter to use TCP/IP as the communications protocol. The COMMSTRING parameter consists of the following information:
 - Protocol identifier: The TCP/IP protocol identifier.
 - TCP. Host name: The IP address or hostname for the z/OS image on which the data server is running.
 - Port number: The port number to use. The name can contain up to four characters. Example using Internet Protocol Version 4 (IPv4) :
`SET,CONFIG,SERVICE=INIT,COMMSTRING='TCP/0.0.0.0/9087';`
2. Update the DATASOURCE parameter for the client configuration to use TCP/IP as the communications protocol. You can use the USERHLQ.USERCONF(CACINIZ) member as an example. Change the DATASOURCE parameter as follows to use the same TCP/IP definition used on the connection handler COMMSTRING parameter:
`DATASOURCE = CACSAMP TCP/0.0.0.0/9087`
3. Activate the connection by using one of the following methods:
 - Stop and restart the data server.
 - Restart the TCP/IP connection handler service.

Configuring clients

This section contains procedures for configuring the Classic federation clients: JDBC, ODBC, and CLI.

JDBC client

The JDBC client provides access to the data server from Java and Java-based tools.

The JDBC client is compliant with JDBC 3.0 and requires Java Virtual Machine, Version 1.3 or later. The JDBC client is distributed as a JAR (Java archive) file.

The JDBC architecture consists of the following components:

- The JDBC application that performs processing and invokes JDBC methods to submit SQL statements and retrieve results.
- A Type 4 JDBC driver that uses a proprietary protocol to communicate with the server and process JDBC API calls. The JDBC driver processes JDBC method invocations, submits data requests to a specific data source, and returns results to the application.

Establishing connections from JDBC applications to data servers

Your JDBC applications can connect to data servers with TCP/IP.

Procedure

1. Load the driver class, `com.ibm.cac.jdbc.Driver`.

The following fragment of Java code loads the driver and its supporting classes:

```
Class.forName("com.ibm.cac.jdbc.Driver");
```

2. Connect to the data server by using TCP/IP as the communication protocol in the URL.

The name of the data source can be a minimum of 1 character and a maximum of 18 characters. The DATASOURCE name corresponds to the query processor service name. The service name is defined in the CACQP task entry for the data server configuration. You can define one or more query processors in the configuration file. The DATASOURCE name must correspond to the query processor that the client connects to.

Complete the URL with the following information:

tcp/hostname or IP address/port number or name of service

hostname or IP address

The z/OS system that hosts the data server. This value with the port number or service name identifies the data server that the JDBC client connects to. If the z/OS system is registered with your network name server, you can use the host name. Otherwise, use the IP_address.

port number or name of service

Supplies the host port number or service name of the data server. This value with the host name or IP address identifies the data server to which the JDBC client connects. If the data server is registered with a network name server, you can use the data server name. Otherwise, you must use the TCP port number, which is the decimal value of the socket number.

If you try to connect to the data server by using IPv6, you might need to provide a scope if you are using a link-local address. The scope is typically the network interface name that you specify following the IPv6 address. The format is `ipv6 address%scope`.

Example

The following fragment of Java code connects to a data server with the data source name CACSAMP by using TCP/IP. The code returns a connection object that is named CACConnection:

```
java.sql.Connection CACConnection =
    java.sql.DriverManager.getConnection(
        "jdbc:cac:CACSAMP:tcp/192.168.0.132/9087",
        "userid",
        "password");
```

The following example shows a URL for the JDBC driver that uses an IPv6 connection:

```
jdbc:cac:CACSAMP:tcp/fe80::xxxx:xxxx:xxxx:xxxx%eth0/9087
```

Batch operations, scrollable ResultSets, and SQL warnings with JDBC

The JDBC client includes support for batch operations, scrollable ResultSets, and SQL warnings in the JDBC 3.0 specification.

Batch operations

- For `java.sql.Statement` objects, an `executeUpdate`, `executeQuery`, or `execute(sql)` method with an UPDATE, DELETE, or INSERT statement causes the update to be run even when batch operations are pending.
- For `java.sql.PreparedStatement` objects, an `executeUpdate`, `executeQuery`, or `execute()` method with an UPDATE, DELETE, or INSERT statement causes the update to be run even when the batch operations are pending. The parameter markers are set before the first `addBatch` operation on the statement.
- The `executeBatch()` method returns an array of integers that indicate the number of rows that are affected. The method stops, if there is an error in execution of any of the statements, and returns only the number of integers that are successfully executed. If a statement returns a `ResultSet` object, the `executeBatch` method treats the object as a failure and returns the array of integers up to that point.

Updatable, scrollable ResultSets

- The `ResultSet.deleteRow()`, `ResultSet.updateRow()`, and `ResultSet.insertRow()` methods are supported, and their changes are visible. The changes made by other methods are not visible to the application after the `ResultSet` object is created.
- The `ResultSet.getXXX()` methods work after the values are updated when an `insertRow` method is created.
- In the case of the `updateRow` method, the `ResultSet.getXXX()` methods return the new values for the `updatedRow` method after a `ResultSet.updateXXX()` method. Otherwise, the `ResultSet.getXXX()` methods return the old values.

The `createStatement`, `prepareStatement`, and `prepareCall` statements are affected by the scrollable ResultSets feature. The types and concurrencies supported are as follows:

```
TYPE_FORWARD_ONLY
TYPE_SCROLL_INSENSITIVE
CONCUR_READ_ONLY
CONCUR_UPDATABLE
```


By default, these statements create a statement that creates a result set that is TYPE_FORWARD_ONLY and CONCUR_READ_ONLY.

SQL warnings

The JDBC driver support for the java.sql.SQLException class supports SQLWarnings. The SQLWarning objects are especially useful when you use the JDBC driver to run DDL statements. The following methods can return multiple SQLWarning objects:

- java.sql.Connection.getWarnings()
- java.sql.Statement.getWarnings()
- java.sql.ResultSet.getWarnings()

The SQLException class and methods return meaningful error messages, with error substitution, that identify objects with errors more precisely.

ODBC clients

The Microsoft Open Database Connectivity (ODBC) interface allows applications to use Structured Query Language (SQL) to access data in database management systems. The ODBC clients provide access to data on servers from Windows, UNIX, and Linux.

ODBC architecture consists of the following components:

- The ODBC-compliant application performs processing and calls the ODBC functions to submit SQL statements and retrieve results.
- The driver manager loads drivers on behalf of an application.
- The driver processes ODBC function calls, submits SQL requests to a specific data source, and returns results to the application.

The driver manager and the client appear to an application as one unit that processes ODBC function calls.

Configuring ODBC data sources

ODBC data sources are registered and configured by using the Microsoft ODBC Administrator. Configuration parameters that are unique to each data source are maintained through this utility.

Before you begin

You must have the following information available when you add and configure an ODBC data source:

- The data source name.
- TCP/IP information:
 - The IP address or the host system where the server runs
 - The port number that is assigned to the TCP/IP connection handler in the service definition of the server

Before you configure the ODBC client, the Windows client must be set up for the TCP/IP connection handler.

About this task

You can define many data sources on a single system. For example, a single IMS system can have a data source called MARKETING_INFO and a data source called

CUSTOMER_INFO. Each data source name needs to provide a unique description of the data.

Procedure

1. Open the ODBC Data Source Administrator notebook.
 - a. Select **Start > Settings > Control Panel**.
 - b. Double-click **Administrative Tools**.
 - c. Double-click **Data Sources (ODBC)**.
2. On the User DSN page, click **Add**.
3. Select IBM WebSphere Classic ODBC Driver from the list.
4. Click **Finish**.
5. Select TCP/IP to use with the data source that you are configuring.

The ODBC Driver Setup notebook displays. In the setup notebook, you can enter the values for the TCP/IP parameters needed for communication with the data source. These parameters must match the values specified in the configuration of the data source.

- a. **Optional:** On the Code Pages page, enable client and server code pages. If you select a bidirectional server and client code page, bidirectional layout transformation is enabled. Set the configuration parameters for bidirectional language support:
 - SHAPING
 - SYMMETRIC SWAPPING
 - TEXT ORIENTATION
 - TEXT PRESENTATION

If required, override the global converter settings at the data source level. To override converter settings and bidirectional language options, use the **Advanced** tab.

Configuring the ODBC driver on Windows

You can configure the ODBC driver by using the ODBC Administrator by selecting **Start > Programs > InfoSphere Classic Tools V11.1 > InfoSphere Classic ODBC Administrator**.

Configuring the ODBC driver on Linux and UNIX

You must use a configuration file to define ODBC data sources on Linux and UNIX systems.

About this task

The configuration file resides in the following installation directories:

- For Linux: `/opt/ibm/isClassic113/cli/lib/cac.ini`
- For UNIX: `/opt/IBM/isclassic113/cli/lib/cac.ini`
- For the 64-bit version, in the 64-bit lib subdirectories. HP-UX only offers a 64-bit driver in the default lib directory.

The following configuration example uses the Data Direct driver manager to access a data source defined as CACSAMP on the host.

Procedure

1. Open the cac.ini file as shown in the following example.

```

*****
* Sample configuration file *
*****
* messages and codes catalog
NL CAT = /opt/IBM/isclassic113/cli/lib
NL = US English
* default datasource location
DEFLOC = CACSAMP
DATASOURCE = CACSAMP tcp/111.111.111.111/nnnn
* performance and memory parameters
FETCH BUFFER SIZE = 32000
MESSAGE POOL SIZE = 1000000

```

2. Edit the DATASOURCE configuration parameter. If the application communicates with multiple data servers or with multiple data sources within a data server, you must define a DATASOURCE configuration parameter for each data server or data source that you want to access.

Optional: You can enable a DATASOURCE override by specifying the following parameters:

```

DATASOURCE = <data source name> <communication protocol>/<ip address>/
<port number>/<server codepage>/<client codepage>/<text presentation value>/
<text orientation value>/<symmetric swapping value>/<shaping value>/

```

For example:

```

DATASOURCE = CACSAMPB tcp/111.111.111.112/1112 IBM-420/IBM-1256/VISUAL/RTL/ON/ON

```

3. Change the communication string for the DATASOURCE parameter to specify the method to communicate with the data server. You can use TCP/IP from the client to access the data server. For example, specify the following DATASOURCE definition for TCP/IP communication:

```

DATASOURCE = sourcename tcp/hostname/portnumber

```

4. Optional: Define code page information with the CLIENT CODEPAGE and SERVER CODEPAGE parameters. If you have data sources for which you want to use different code pages, you can define the data sources in separate configuration files. If you select a bidirectional server and client code page, bidirectional layout transformation is enabled. For bidirectional client and server code pages, you can set the configuration parameters for bidirectional language support:

- SHAPING
- SYMMETRIC SWAPPING
- TEXT ORIENTATION
- TEXT PRESENTATION

5. Create an environment variable CAC_CONFIG and set it to point to the cac.ini configuration file.
6. Create a library environment variable that includes the directories where the shared libraries are installed. Specify one of the following library environment variables:

- For AIX: **LIBPATH**
- For HP-UX: **SHLIB_PATH**
- For Linux: **LD_LIBRARY_PATH**
- For Solaris: **LD_LIBRARY_PATH**

7. Run the CLI application in this environment. For example, in AIX, run the following export statement:

```

export CAC_CONFIG=/opt/IBM/isclassic113/cli/lib/cac.ini
export LIBPATH=/lib:/opt/IBM/isclassic113/cli/lib
program1

```

8. Edit the `odbc.ini` file to add a new data source for the UNIX, Linux and USS client. The `odbc.ini` file is located in the CLI software directory for the application where the driver manager resides. The data source name must correspond to a query processor name defined on the data server and a `DATASOURCE` name defined in the client configuration. For example:

```
[ODBC data sources]
CACSAAMP=InfoSphere Classic Federation client
.
.
.
[CACSAAMP]
client=/opt/IBM/isclassic113/cli/cacsqcli.so
```

You can specify one of the following clients:

- AIX: `cacsqcli`
- HP-UX: `libcacsqcli.sl`
- Linux: `libcacsqcli.so`
- Solaris: `libcacsqcli.so`

You must add a data source definition for each data source that you want to access.

CLI clients

InfoSphere Classic Federation Server for z/OS supports Call Level Interface (CLI) clients for UNIX and Linux, native z/OS, and USS.

You can develop 32-bit and 64-bit CLI client applications.

- 32-bit CLI drivers are available on AIX, Linux, zLinux, and Solaris but not available on HP-UX.
- 64-bit CLI drivers are available on AIX, Linux, zLinux, Solaris, and HP-UX.

You can use the following sample applications:

- `clisamp` for testing connectivity and SQL calls on the Windows, UNIX and Linux operating systems
- `CACSAAMP` for testing connectivity and SQL calls on z/OS and USS

The CLI clients communicate with a Classic data server by connecting to a connection handler service that is defined to the data server. The connection handler service routes client requests to query processor services. The query processor services access the various Classic federation data sources. Each connection handler service instance can service multiple client applications concurrently.

The Classic client drivers are multithreaded. Multithreaded client applications can spawn multiple concurrent connections to the Classic data server. Also, similar to the connection handler, a query processor can serve multiple clients. The connections are handled at the user level and each physical connection is represented by a user in the Classic data server.

A query processor can serve multiple physical connections from one client application or from multiple client applications. You can also configure a query processor to only serve one user to avoid any interference.

The CLI architecture consists of the following components:

- The CLI-compliant application, which performs processing and calls the CLI functions to submit SQL statements and retrieve results.

- CLI function calls, that submit SQL requests to a specific data source and return results to the application.

You define a data source to identify the data that you want to access. The data source name is equivalent to the query processor service name defined to the data server. Defining a data source consists of defining the service name and communication parameters (TCP/IP) to determine the data server with which the client is communicating.

You can also configure ODBC driver managers on the UNIX and Linux operating systems to load the ODBC driver.

CLI client for UNIX, Linux, and Windows

With the CLI clients for UNIX, Linux, and Windows, applications use SQL to access data in both relational and non-relational database management systems.

These CLI clients provide access from a UNIX, Linux, or Windows client application or tool to data servers.

Configuring the UNIX, Linux, and Windows CLI client:

To configure the UNIX, Linux, and Windows CLI clients, you edit and customize client configuration parameters.

About this task

The configuration file resides in the following installation directories:

- For Linux: /opt/ibm/isclassic113/cli/lib/cac.ini.
- For UNIX: /opt/IBM/isclassic113/cli/lib/cac.ini.
- For Windows: C:\Program File\IBM\ISClassic113/ODBC/lib/cac.ini.
- For the 64-bit version on AIX, Linux, zLinux, and Solaris: the configuration file resides in the 64-bit lib subdirectories.

Procedure

1. Open the cac.ini file as shown in the following example.

```
*****
* Sample configuration file *
*****
* messages and codes catalog
NL CAT = /opt/IBM/isclassic113/cli/lib
NL = US English
* default datasource location
DEFLOC = CACSAMP
DATASOURCE = CACSAMP tcp/111.111.111.111/nnnn
* performance and memory parameters
FETCH BUFFER SIZE = 32000
MESSAGE POOL SIZE = 1000000
```

2. Edit the DATASOURCE configuration parameter. If the application communicates with multiple data servers or with multiple data sources within a data server, you must define a DATASOURCE configuration parameter for each data server or data source that you want to access.

Optional: You can enable a DATASOURCE override by specifying the following parameters:

```
DATASOURCE = <data source name> <communication protocol>/<ip address>/
<port number>/<server codepage>/<client codepage>/<text presentation value>/
<text orientation value>/<symmetric swapping value>/<shaping value>/
```

For example:

```
DATASOURCE = CACSAMPB tcp/111.111.111.112/1112 IBM-420/IBM-1256/VISUAL/RTL/ON/ON
```

3. Change the communication string for the DATASOURCE parameter to specify the method to communicate with the data server. You can use TCP/IP from the client to access the data server. For example, specify the following DATASOURCE definition for TCP/IP communication:

```
DATASOURCE = sourcename tcp/hostname/portnumber
```

4. Optional: Define code page information with the CLIENT CODEPAGE and SERVER CODEPAGE parameters. If you did not install into the default directory, you need to update the NL CAT parameter.

If you have data sources for which you want to use different code pages, you can define the data sources in separate configuration files. If you select a bidirectional server and client code page, bidirectional layout transformation is enabled. For bidirectional client and server code pages, you can set the configuration parameters for bidirectional language support:

- SHAPING
- SYMMETRIC SWAPPING
- TEXT ORIENTATION
- TEXT PRESENTATION

5. Create an environment variable CAC_CONFIG and set it to point to the cac.ini configuration file.
6. Create a library environment variable that includes the directories where the shared libraries are installed. Specify one of the following library environment variables:

- For AIX: **LIBPATH**
- For HP-UX: **SHLIB_PATH**
- For Linux: **LD_LIBRARY_PATH**
- For Solaris: **LD_LIBRARY_PATH**
- For Windows: **PATH**

7. Run the CLI application in this environment. For example, in AIX, run the following export statement:

```
export CAC_CONFIG=/opt/IBM/isclassic113/cli/lib/cac.ini
export LIBPATH=/lib:/opt/IBM/isclassic113/cli/lib
program1
```

CLI client for native z/OS

With the CLI client for native z/OS, you can build client applications by using the IBM C runtime environment.

Building native z/OS CLI applications:

For native z/OS, you need to build and link CLI applications for the IBM C runtime environment.

Compiling and linking with IBM C

The sample JCL CACBLDI is provided for compiling and linking a CLI application on native z/OS in the IBM C runtime environment. The CACBLDI sample is located in the *USERHLQ.USERSAMP* data set. The CLI application is named MYCLIAPP in the following example:

```

//*****
//*
//*  CACBLDI
//*  IBM C - COMPILE AND LINK A SAMPLE APPLICATION MYCLIAPP
//*  WHICH USES STANDARD EDCCL JCL PROCEDURE
//*
//*  1) PROVIDE A JOB CARD THAT IS VALID FOR YOUR SITE
//*  2) CHANGE CACINHLQ REFERENCES TO INSTALLED HLQ
//*  3) CHANGE USERHLQ REFERENCES TO USER HLQ
//*  4) CHANGE your REFERENCES TO IDENTIFY THE SOURCE AND          *
//*     TARGET LOCATIONS OF YOUR APPLICATION
//*
//*  NOTE: THE CLI/ODBC HEADER FILE CACCLI WILL BE INCLUDED FROM
//*        USERHLQ.USERSAMP DURING COMPILATION
//*
//*****
//*
//CCIBM   EXEC EDCCL,
//        INFILE='your.source.library(MYCLIAPP)',
//        OUTFILE='your.target.loadlib,DISP=SHR',
//        CPARAM='DLL',
//        LPARM='DYNAM=DLL'
//*
//COMPILE.SYSLIB DD DISP=SHR,DSN=UserHLQ.USERSAMP
//LKED.SYSDEFSDD DD DISP=SHR,DSN=CACINHLQ.SCACSIDE
//LKED.SYSIN   DD *
//          INCLUDE SYSDEFSDD(CECCLI)
//          INCLUDE SYSDEFSDD(CECVHS)
//          INCLUDE SYSDEFSDD(CECSQL)
//          NAME MYCLIAPP(R)
//*

```

Configuring native z/OS CLI clients:

You need to create JCL and create a configuration file to configure the native z/OS CLI client.

Procedure

1. Create JCL that specifies the following DD names:

STEPLIB

Points to the data set where the load modules reside. Includes the Classic federation installation load library to load the necessary CLI DLL load modules.

MSGCAT

Points to the Classic federation message catalog.

VHSCONFIG

Points to a data set that contains the CLI configuration parameters. You can use *USERHLQ.USERSAMP(CACCLNT)* as a JCL sample.

2. Use the sample configuration file CACINIZ. The CACINIZ sample is located in the *USERHLQ.USERCONF* data set. The following example shows a configuration file that the VHSCONFIG DD name refers to:

```

NL CAT = DD:MSGCAT
NL = US English
* default datasource location
DEFLOC = CACSAMP
DATASOURCE = CACSAMP tcp/n.nn.nnn.nn/9087
* performance and memory parameters
FETCH BUFFER SIZE = 32000
MESSAGE POOL SIZE = 4000000

```

```
TRACE LEVEL = 4
* Logger thread:
SERVICE INFO ENTRY = CACLOG LOG 1 1 1 100 1 5M 5M DISPLAY
```

CLI client for USS

With the CLI client for USS, you can build client applications by using the IBM C runtime environment.

Building USS CLI applications:

The sample file CACMAKEU is provided for compiling and linking a USS CLI client application. The CACMAKEU sample is located in the *USERHLQ.USERSAMP* data set.

Example: The application is named `mycliapp` in this example:

1. Replace the CACINHLQ with the high-level qualifier for the Classic federation installation.
2. Copy the CACCLI header file from UserHLQ.USERSAMP into the relative path `../include`, and rename it to `caccli.h`

```
PASSCC = cc
CC = $(PASSCC) $(XCFLAGS) -I include
CFLAGS = -W c,dll -DMVS -DUNIX -DBIGENDIAN -D_POSIX_C_SOURCE
XCFLAGS = $(CFLAGS) -D_POSIX_SOURCE -DCOMPLETE_EXTERNAL_TASK

mycliapp : mycliapp.o
    cc -o $@ -W l,dll mycliapp.o "'/'CACINHLQ.SCACSIDE(CECLI)'"

mycliapp.o : mycliapp.c
    $(CC) -c mycliapp.c
```

Configuring the USS CLI client:

You need to define the necessary environment variables and create a configuration file to configure the native USS CLI client.

Before you begin

The authorization ID needs search authority for the directory that contains the configuration file and read permissions.

Procedure

1. Define the following environments variables:

CAC_CONFIG

Point to the configuration file. For example, the following export statement refers to the file `cac.ini` in the current working directory:

```
export CAC_CONFIG=./cac.ini
```

STEPLIB

Specify the load library where the Classic federation load modules are located. For example:

```
export STEPLIB=CAC.V11R3M00.SCACLOAD
```

2. Define configuration parameters for the CLI client on USS. Use the sample USS configuration file CACINIU to create the configuration file. The CACINIU sample is located in the *USERHLQ.USERSAMP* data set.

```
NL CAT = //CAC.V11R3M00.SCACMSG
NL = US English
* default datasource location
```

```

DEFLOC = CACSAMP
DATASOURCE = CACSAMP tcp/x.x.x.x/9087

* performance and memory parameters
FETCH BUFFER SIZE = 32000
MESSAGE POOL SIZE = 16000000
TRACE LEVEL = 4
TASK PARAMETERS = CACLOG=/u/test/caclog

```

In this example:

- The NL CAT configuration parameter points to the message catalog in the Classic federation installation data set.
- When the TRACE LEVEL configuration parameter is set, the CLI driver starts the logger task. The CACLOG parameter indicates the file where the logger should write the log messages.

Running the cacprtlg utility:

The logger generates a binary log file. You can use the cacprtlg utility to format the binary log file. The cacprtlg utility reads the binary log files and writes the formatted log records to standard output.

Before you begin

The authorization ID needs execute permissions to run the cacprtlg utility.

Procedure

1. Define the following environments variables:

CAC_CONFIG

Points to the configuration file. For example, the following export statement refers to the file cac.ini in the current working directory:

```
export CAC_CONFIG=./cac.ini
```

ddn_CACLOG

Points to the logger output as shown in the following example:

```
export ddn_CACLOG=./caclog
```

STEPLIB

Specify the load library where the Classic federation load modules are located. For example:

```
export STEPLIB=CAC.V11R3M00.SCACLOAD
```

2. Create a file in the path named cacprtlg in the HFS file system. Creating this file informs the runtime loader to look for cacprtlg in the STEPLIB variable. Use the following commands to create the file:

```
touch cacprtlg
chmod 755 cacprtlg
chmod +t cacprtlg
```

3. Optional: Define the ddn_CACFILTER environment variable. This variable points to a text file that contains filter information to limit the output that the cacprtlg utility produces. Use the following command to define the filter file:

```
Export ddn_CACFILTER=./filter.txt
```

Recommendation: Redirect the cacprtlg output to a file. Then download and view the file on a different operating system.

Chapter 4. Administering

After you install and configure your Classic federation environment, you typically perform administration tasks on an ongoing basis.

Administering data servers

Included with IBM InfoSphere Classic Federation Server for z/OS is a z/OS MTO (Master Terminal Operator) interface that you can use to monitor and control data server operations.

Classic data servers are designed to run continuously. With the MTO interface, you can issue commands to display the different services that are active within a data server, the number of users that are currently using a data server or enterprise server, and the amount of memory that is available. You can also issue commands to start services and stop non-critical services.

You can run commands in the following format:

```
F name_of_job,command
```

- F is the abbreviation for the z/OS MODIFY command.
- *name_of_job* is the name of the started task to communicate with.
- *command* is the command to pass to the started task.

In Classic federation, if the started task is a data server that was started for an enterprise server, specify the fully-qualified task name in the following format:

```
name_of_data_server.stepname
```

To send the command to all data servers that are managed by an enterprise server, you can use an asterisk instead of *stepname*.

For example, the first of the following commands is issued for the data server t9396840, and the second command is issued for all of the data servers that were started by the enterprise server:

```
F CACDS.t9396840,display,users  
F CACDS.*,display,config=master
```

You can also update your data server configuration by using the Classic Data Architect or the master terminal operator (MTO) interface. With both methods, you make configuration updates against a running data server.

The commands reference section describes each administration and configuration command that you can use to administer data servers.

System exits

System exits are programs that the system calls at predefined processing points to provide security and accounting functionality for Classic federation.

The data server is a multi-threaded implementation designed to service large numbers of concurrent users. To optimize performance in the data server's multi-tasking environment, all of the system exits are written in Assembler

language, with the exception of the record processing exit. They are assembled as re-entrant. Additionally the exits are link-edited as AMODE 31 RMODE ANY and REF, RENT, RU.

Restriction: If you customize an exit, ensure that your version also meets these criteria.

Security: SAF exit

You can use the SAF exit for security validations. The SAF exit verifies access for user and client connections, to stored procedures, and to specific references in queries.

The SAF exit controls access to system resources based on classes and profiles. The exit authenticates user passwords at connection time. In addition, you can configure the exit to perform the following functions:

- Validate user authorization to access metrics data or run remote console commands.
- Authenticate the TCP/IP address of client connections at connection time
- Verify authority to access a physical file or PSB referenced in an SQL query
- Verify authority to execute a stored procedure program
- Pass user IDs and passwords to CA-Datcom to verify user authorization to access CA-Datcom table references in an SQL query

Recommendation: Use the supplied sample exit module (CAC SX04) in the SAMPLIB data set. If you choose to modify or replace the supplied exit source code, you must assemble and bind the module as described in the overview of the SAF exit API.

In the first implementation of the SAF exit, the SAFID field within the principal data structure presented to the SAF exit (DSECT SAF mapped by the CACSXPL4 macro) is defined as a four-character field that contains the character string "SAF". In all subsequent implementations, SAFID is defined as a three-character field that contains the character string "SAF " followed by a single character that denotes the version (or level) of the structure (the SAF DSECT field). Some functional features of the interface are present only when the SAFVER field is greater than or equal to a particular version value, for example, the presence of the IP address of a connected client. The comments within the "SAF"DSECT explain these dependencies.

User-written code contained within the SAF exit must account for these structure version dependencies before attempting to reference (for example, fetch from or store into) version-dependent fields. Otherwise, unpredictable results can occur, including the potential abnormal termination (abend) of the unit-of-work that activated the SAF exit.

Restriction: The SAF exit invokes services that require APF authorization. Therefore, the SAF exit must reside in an APF-authorized library. The exit routine must not contain the AC(1) attribute.

Activating the SAF exit

To activate the SAF exit, you set up security resources and define configuration parameters during the installation customization process.

Before you begin

Ensure that the STEPLIB DD statement for the data server JCL references the library where the SAF exit load module (CACSX04) is located. The library must be APF-authorized.

About this task

Ensure that only valid z/OS users can access resources by setting SAFEXIT=CACSX04 for the operator service, monitoring service, and query processor services. After authentication, users do not need authorization for each individual resource.

You can configure the supplied SAF exit CACSX04 by using the following configuration parameters. You can supply configuration parameters only when you use the IBM-supplied version of the SAF exit CACSX04.

IMS CLASS=*Class Name*

Specifies the name of the resource class that is checked to determine whether the user has authority to schedule or access the PSBs associated with the tables referenced in a query. The Class Name can be up to eight characters long. This sub-parameter is required when accessing IMS data.

PSB PREFIX=*Prefix*

Specifies a value to prefix the PSB name when forming the resource name. For example, if PSB PREFIX=IMSP is specified and the PSB name is PSB1, the resource name becomes IMSPPSB1.

If you are planning to access IMS data, you might need to modify the IMS CLASS subparameter to define the resource class where IMS PSBs are defined at your site.

To use a PSB, a user ID must have at least CONTROL access to that PSB's corresponding profile within the class.

If you are using RACF, the combination of the length of the PSB name and the length of the prefix must be eight characters or less. If a larger PSB name or prefix combination is encountered, an error message is issued.

ADACLASS=*Facility*

Specifies the name of a class used to check for authorized use of ADABAS view names. The operand can be up to eight characters in length. There is no default value for this parameter. The security administrator must define each ADABAS view name as a resource in the class and grant CONTROL access to each user ID that uses that view name. If the ADABAS view name is not defined, or the user ID is not granted access, the following message is returned on the attempt to pull data from the ADABAS table defined with a view name:

Access Denied

If the ADABAS table is only defined with a file number (no Predict view name), you will receive the same error message as shown above, and the server log contains the following message:

```
CACL010E NO ADABAS VIEW NAME IN USE GRAMMAR
```

SPCLASS=FACILITY

Specifies the name of a class used to check for authorized use of stored procedure names that are defined in the metadata catalogs. These names

are stored in the SYSIBM.SYSROUTINES system table. The operand can be up to eight characters in length. There is no default value for this parameter.

The external security manager (ESM) administrator must define each stored procedure as a resource in this class and grant ALTER access to each user ID that will invoke the stored procedure. If the stored procedure is not defined to ESM or the user ID is not granted access, -5046295 is returned on the attempt to use a stored procedure and message CACL006W is issued.

EXCLUDE=*n*

Indicates the query processor should not provide an ACEE address in commands sent to CA-Datcom. When the SAF exit is active, the address of an ACEE is obtained during SAF exit initialization. This ACEE address is normally passed to CA-Datcom in each database request and CA-Datcom authenticates the request using information within the ACEE.

Whenever the SAF exit is active and you want to avoid database level security checking in CA-Datcom, you must indicate the query processor should exclude the ACEE from the database requests that are sent to CA-Datcom. Set the value of *n* to 2 (heterogeneous query processor CA-Datcom connector). This setting will not provide the ACEE address in the call parameters.

VALIDATE=*Y/N*

Indicates whether the exit should validate that the user ID has authority to access or use a protected resource. Examples of protected resources include, but are not limited to, data sets, IMS PSBs, and databases. Use both SQL security and the SAF exit in conjunction with your site security package to restrict access to your data.

Specify an operand of **Y** to indicate that the exit routine should validate access rights to protected resources. Typically, access rights validation is accomplished by running a RACROUTE macro which invokes a security system, such as IBM RACF, to perform the necessary validation tasks. Specify an operand of **N** to indicate that the exit routine should not perform access rights validation.

The exit routine is invoked for access rights checking regardless of the **Y** or **N** value of the operand. The default value for VALIDATE is **Y**.

This parameter helps you to control access with greater precision:

- Ensure that only valid users can access resources by setting VALIDATE=Y against the operator, monitoring, or query processor services.
VALIDATE=Y authenticates each individual resource.
- Eliminate the overhead of verifying that the user has authority to access a resource by setting VALIDATE=N against the operator, monitoring, or query processor services.

Do this if you have elected to use SQL security to control access to tables and stored procedures. This setting is also useful in a test or development environment if you trust any user with a valid z/OS user ID to access the data. For example, you might not want to use DB2 privileges or use RACF to verify each resource.

NETACCESS=*Y/N*

Indicates whether the exit should validate the IP address of the connected client to authenticate access to the data server.

Set the value to Y when the IP address of the connected client is known and the SERVAUTH parameter of the RACROUTE REQUEST=VERIFY invocation is supplied. The RACROUTE operation is successful when the associated user ID has at least READ-level access rights to the network security zone resource. If the security system indicates that it cannot make a decision in response to the request because a corresponding network security zone resource profile does not exist, the SAF exit regards the response as Access Denied.

A value of N indicates that the SERVAUTH parameter is omitted from the RACROUTE REQUEST=VERIFY invocation. This is the default.

MONCLASS=*monitor-class-name*

Indicates the name of the security class that contains a profile that provides access authentication.

This parameter is valid if VALIDATE=Y on the monitoring service for the SAF exit. If this parameter is not specified, SERVAUTH is the default name of the operator security class.

MONPROF=*monitor-profile-name*

Indicates the name of the resource profile that provides access authentication.

This parameter is valid if VALIDATE=Y on the monitor service for the SAF exit. If this parameter is not specified, CEC.MONITOR is the default profile name.

OPERCLASS=*operator-class-name*

Indicates the name of the security class that contains a profile that provides access authentication.

This parameter is valid if VALIDATE=Y on the operator service for the SAF exit. If this parameter is not specified, SERVAUTH is the default name of the operator security class.

OPERPROF=*operator-profile-name*

Indicates the name of the resource profile that provides access authentication.

This parameter is valid if VALIDATE=Y on the operator service for the SAF exit. If this parameter is not specified, CEC.OPER is the default profile name.

If you want to modify the supplied exit to add capabilities that the existing configuration parameters do not provide, deploy a custom exit as described in the topic about security and the SAF exit.

To configure the SAF exit and verify that it is working:

Procedure

1. Edit the sample SAF exit (SCACSAMP member CACSX04) if you need to customize CACSX04.
2. Add any required SAFEXIT configuration parameters in the service definition for the monitoring, operator, or query processor service classes (OPER, MAA, QP, or QPRR). by using the Classic Data Architect or MTO SET,CONFIG command.
3. After changing the SAF definition for any services that you need to secure, stop and restart the data server.

Example

Ensure that only a valid z/OS user can connect to the operator and monitoring services by using the following command. This command runs the supplied SAF exit with the default values:

```
F <Data-Server-Name>,SET,CONFIG,SERVICE=<Service-Name>,SAFEXIT=CACSX04
```

If you want to take advantage of the flexibility of the exit by using SAF exit parameters against a monitoring service, add name/value pairs to the command, as follows:

```
F <Data-Server-Name>,SET,CONFIG,SERVICE=<Monitor-Service-Name>,  
SAFEXIT="CACSX04,VALIDATE=Y"
```

The first value of the SAFEXIT parameter must be the exit name. You must enclose the SAFEXIT values in double quotes.

SAF exit: API overview

The parameters passed to the SAF exit are defined by the CACSXPL4 member located in the SCACMAC library. The SAF exit is called for one of three functions: initialization, validation, or termination.

The CACSXPL4 macro describes the interface to the SAF exit. The comments provided in the CACSXPL4 macro describe the SAF structure fields and their intended usage. Comments contained within the CACSX04 source code describe the interface to and the intended behavior of the SAF exit.

Assembling and binding the SAF exit

Member CACALX04 in the USERSAMP data set contains a JCL stream that you can use to assemble and bind the sample SAF exit, CACSX04.

Requirements for assembling and binding CACSX04:

- You must direct the assembler to produce Extended Format (GOFF) object code. Specifying the assembler options GOFF,OBJ,NODECK satisfies this requirement and directs the assembler to write the object code to the file referenced by the SYSLIN DD statement.
- The CACSX04 executable module (a program object) must reside in a PDSE that is included in the list of APF-authorized libraries.
- The traditional load module format is not supported.
- You must direct the z/OS Program Management Binder to produce a program object at the z/OS V1R10 level or higher with support for mixed-case external symbol names. The resultant program object must contain the re-entrant attribute and must not contain the AC(1) attribute.

Specifying the binder options RENT,CASE=MIXED,COMPAT=ZOSV1R10 meets these requirements.

For more information about assembler options, see the High Level Assembler documentation in the z/OS product documentation.

For more information about the z/OS binder options see the z/OS MVS™ Program Management documentation in the z/VM product documentation.

SAF exit: initialization:

The initialization function is used to initialize the SAF environment for a user when the user connects to the query processor.

The SAFNAME field contains a pointer to a null-terminated character string that consists of the exit initialization parameters. When no initialization parameters are supplied to the exit, the pointer value itself can be binary zeros or the referenced area can consist of an effectively empty string (zero or more blanks followed by a terminating null byte). The input parameters are specified on the SAF exit configuration parameter used to invoke the SAF exit. On the SAF exit parameter additional sub-parameters can be placed after the name of the exit.

The SAFUSERI and SAFUSERP fields contain the user ID and password of the user that is connecting to the service.

The exit performs initialization processing and allocates storage or other resources that are required for validation processing. A pointer to the anchor for these resources can be placed in the SAFUSER field of the parameter list. This pointer is preserved and passed to the exit on subsequent invocations.

For example, the SAF exit validates that the user ID and password are valid and constructs an ACEE for the user. In the SAF structure field SAFACEE, the SAF exit is required to return the address of the ACEE representing the user, or binary zeros to indicate that an ACEE pointer is not provided.

Subsequently, server processes and the SAF exit use the referenced ACEE when performing activities on behalf of the user when those activities might need to access or operate upon protected resources.

If the optional input subparameter EXCLUDE= is included in the SAF exit configuration parameters, the specified value is returned to the caller in the SAFEXCLD field of the parameter list. This field is used in conjunction with the SAFACEE field when performing database-level security checking within CA-Datcom.

Additionally, the SAF exit inspects the ACEEGRPL field in the ACEE. If the length field is greater than zero, the contents of the ACEEGRPN field is copied to the SAFCGRP for subsequent use in SQL security processing.

If the exit returns a non-zero return code, query processor initialization for the user is halted, the user is disconnected, and the return code issued by the exit is returned to the client application.

SAF exit: validation:

The validation function is used to verify a user's authority against objects such as PSBs, Adabas files, and stored procedures.

The SAFTYPE field identifies the type of validation to be performed.

SAFIMS

For IMS access using a DBB or BMP connector interface the exit is called immediately before PCB selection logic is invoked. The name of the PSB that is referenced by the SAFNAME field is the PSB specified in the data server's JCL. The exit should verify that the user has authority to use the specified PSB name identified in the SAFNAME field.

When you use the DRA interface to access IMS data, the exit can be called at two different points. The exit is always called immediately before a PSB is to be scheduled. The exit should validate that the user has authority to use the PSB referenced by the SAFNAME field.

If the exit returns a non-zero return code, the PSB is not scheduled, processing for the query is terminated, any other PSBs that have been scheduled for the query are unscheduled, and the return code issued by the exit is returned to the client application. The application can still issue other SQL requests.

The exit can also be called when the query contains a join. In this situation, the DRA interface schedules the JOIN PSB specified in the metadata grammar for a table referenced in the query. The exit is invoked as previously described. For subsequent tables in the join, the connector checks to determine whether the PSBs that have already been scheduled contain a PCB that can be used to access the table. If a usable PCB is located, the SAF exit is called with the name of the primary PSB (as specified in the metadata grammar for the table that is referenced in the query). This PSB is not scheduled, however, an authorization check is performed to verify that the user has authority to access the primary PSB (referenced by the SAFNAME field) associated with the table.

If the exit returns a non-zero return code, the other PSBs that have been scheduled for the query are unscheduled and the return code issued by the exit is returned to the client application. The application can still issue other SQL requests.

There is no indication of which of the three processing sequences invoked the exit.

SAFVSAM and SAFSEQ

When a query references a sequential or local VSAM file, the exit is called immediately before the file is opened. The exit validates that the user has authority to access the file name referenced by SAFNAME. When a PDS member is referenced, the name of the member is not passed to the exit.

If the exit returns a non-zero return code, the file is not opened, processing for the query is terminated, and the return code issued by the exit is returned to the client application. The application can still issue other SQL requests.

SAFSP

For stored procedures, the SAF exit is invoked immediately before the application program associated with a stored procedure definition is loaded for execution. The SAF exit validates that the user has authority to execute the program. This validation is performed indirectly based on the stored procedure table name referenced by the SAFNAME.

If the exit returns a non-zero return code, the program is not loaded or executed, processing for the stored procedure request is terminated, and the return code issued by the exit is returned to the client application. The application can still issue other SQL requests.

SAFADAB

When a query references an ADABAS database, the exit is called immediately before the database is accessed. The exit validates that a view name is provided in the SAFNAME field and that the user has authority to access the identified view.

If the exit returns a non-zero return code, the database is not accessed, processing for the query is terminated, and the return code issued by the exit is returned to the client application. The application can still issue other SQL requests.

SAF exit: termination:

The termination function is called when a user disconnects from the query processor. At this time, the exit can perform any termination processing necessary, and must free any resources it has allocated.

SAF exit: System access control - examples

When a user accesses the data server through a TCP/IP connection, the IP address associated with that user is presented to the SAF exit. This information can be used during SAF exit initialization, for example, to restrict access by specific users or groups connecting to the data server from specific IP addresses.

TCP/IP uses the SAF SERVAUTH resource class to control access to a large variety of network resources. The implementation of the sample CACSX04 exit exploits a resource type known as a network security zone as the entity to protect.

You define a network security zone to TCP/IP by using the NETACCESS configuration control statement. Each network security zone that you define must have a SAF SERVAUTH profile for the resource named `EZB.NETACCESS.sysname.tcpname.zonename`.

sysname

The name of the z/OS system. This is the value associated with the `&SYSNAME` system symbol.

tcpname

Name of the TCP/IP stack. This is the name of the JCL procedure used to start the TCP/IP address space which is equivalent to the `TCPIPJOBNAME` parameter in the TCP/IP control statement set.

zonename

Name of the security zone.

In general, network security zone RACF profiles are defined with a Universal Access (UACC) of NONE. Individual users or groups who are to be permitted to access the network security zone are then granted READ access to the corresponding resource profile.

You set the SAF exit NETACCESS parameter to control the behavior of the SAF exit based on the IP address of the connected client.

See the z/OS Security Server documentation for more information about network security zones and RACF commands.

Example: Prevent access to the data server

In this example, you want to prevent inbound access to the data server that is running with user ID SAM by any clients in security ZONEB. The following steps are required to prevent access:

- Define the IP addresses that comprise security zone ZONEB to TCP/IP by using NETACCESS configuration control statements with the INBOUND qualifier.

- Define the SAF resource profile for security ZONEB to RACF with a universal access of READ. The resource profile name is of the form: EZB.NETACCESS.*sysname.tcptime*.ZONEB. For example:
RDEFINE SERVAUTH EZB.NETACCESS.*sysname.tcptime*.ZONEB UACC(READ)
- Deny user SAM access to resource EZB.NETACCESS.*sysname.tcptime*.ZONEB in the SERVAUTH class. For example:
PERMIT SERVAUTH EZB.NETACCESS.*sysname.tcptime*.ZONEB CLASS(SERVAUTH)
ID(SAM) ACCESS(NONE)

As a result, TCP/IP prevents any dispatchable unit running with user ID SAM from receiving any inbound traffic from a client whose IP address is in ZONEB.

Example: Permit access to the data server

In this example, you want to permit access to the data server running with user ID SAM to specific clients with specific user IDs entering the system from security ZONEB. To permit access, the following steps are required:

- Define the IP addresses that comprise security ZONEB to TCP/IP by using NETACCESS configuration control statements with both the INBOUND and OUTBOUND qualifiers.
- Define the SAF resource profile for security ZONEB to RACF with a universal access of NONE. The resource profile name is of the form: EZB.NETACCESS.*sysname.tcptime*.ZONEB. For example:
RDEFINE SERVAUTH EZB.NETACCESS.*sysname.tcptime*.ZONEB UACC(NONE)
- Grant user SAM and the additional user IDs who are allowed to connect to the server READ access to resource EZB.NETACCESS.*sysname.tcptime*.ZONEB in the SERVAUTH class. For example:
PERMIT SERVAUTH EZB.NETACCESS.*sysname.tcptime*.ZONEB CLASS(SERVAUTH)
ID(SAM) ACCESS(READ)
PERMIT SERVAUTH EZB.NETACCESS.*sysname.tcptime*.ZONEB CLASS(SERVAUTH)
ID(other) ACCESS(READ)

Accounting: SMF exit

The SMF exit is used to report wall-clock time and CPU time for an individual user session with a query processor task. Additionally, if SQL security is active and an authorization violation is detected by the query processor, the exit is called to log the violation.

The SMF exit writes out its data as user SMF records. SMF requires that an application that writes SMF records be run from an APF-authorized library.

Activating the SMF exit

When you activate the SMF exit, you can maintain elapsed time values used by SQL statements and log authorization violations.

Before you begin

1. Install the data server, perform initial configuration customization, and verify the installation configuration using the sample application and data.
2. Include the SMF exit load module (CAC SX02) in an APF-authorized library (SCACLOAD).
3. Ensure that the data server JCL references the APF-authorized library in the STEPLIB DD statement where the SMF exit is located (SCACLOAD).
4. Ensure that any other data sets referenced in the STEPLIB DD statement are also APF-authorized.

Procedure

1. Edit the sample SMF exit (SCACCONF member CACSX02).
2. You can supply the following values on the SMFEXIT configuration parameter when using the IBM-supplied version of the SMF exit.

RECTYPE=*nnn*

This is a required parameter that defines the SMF user record type. This parameter contains a numeric value between 128 and 255.

SYSID=*xxxx*

This is a required parameter that contains the primary JES subsystem ID. SYSID can be a maximum of four characters.

3. Start the data server.
 - If your data server is already running, restart it.
 - This operation can also be performed using the MTO Operator Interface.
4. In the SCACSKEL member CACSQL, uncomment the SELECT statements that reflect the data source you are using.
 - a. Ensure that the user ID is authorized to access the table that is referenced in the query.
 - b. Save the changes after you have completed editing the member.
5. Edit SCACSAMP member CACCLNT to set the SQLIN parameter to reference the name of the SCACSAMP member that you edited in the previous step. Save your changes.
6. Submit CACCLNT and review the output.

The query should function normally and return the expected result set.
7. Ensure that the SMF record file exists, for example, SYS1.MAN1.
8. Dump the SMF records related to Classic federation into a data set. Sample JCL is shown in the following example.

```
//*INSERT VALID JOB CARD HERE
//STEP1 EXEC PGM=IFASMFDP
//INDD1 DD DISP=SHR,DSN=SYS1.MAN1
//OUTDD1 DD DISP=(NEW,CATLG),DSN=CAC.SMFDUMP,
// UNIT=SYSDA,VOL=SER=XXXXXX,SPACE=(TRK,(5,5),RLSE),
// DCB=(LRECL=32760,BLKSIZE=27998,RECFM=VBS,DSORG=PS)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
INDD(INDD1,OPTIONS(DUMP))
OUTDD(OUTDD1,TYPE(xxx))
```

Where Type (*xxx*) is the record type as specified in the RECTYPE= parameter.

9. Run SAS, IDCAMS, or any other tool that processes SMF records. If you run IDCAMS, specify the following SYSIN:

```
//STEP2 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
PRINT INDATASET(CAC.SMFDUMP)
```

Where *CAC.SMFDUMP* is the name of the dump file.

10. Verify the output.

To verify the output, examine the DSECT fields that map the SMF accounting routine output data from the SMF exit.

Verifying SMF exit output:

DSECT fields map the output data from the SMF exit accounting routine. Understanding the field definitions enables you to verify this output data.

The following tables describe the DSECT fields associated with the SMF exit accounting routine, and related authorization violation type codes.

Table 22. SMF accounting file DSECT field definitions

Field	Definition	Length (in bytes)
CACSXSMF DSECT	Data structure for the SMF record.	N/A
RDWLEN DS H	Length of the record.	2
RDWSPAN DS H	Reserved for system use.	2
FLG DS X	Reserved for SMF use.	1
RECTYPE DS AL1	Value specified in the RECTYPE= parameter. This is the SMF record type.	1
ENTIME DS BL4	Time in BIN format from TIME macro. This is the time the event ended.	4
ENDATE DS PL4	Date in BIN format from the TIME macro. This is the date the event ended.	4
SYSID DS CL4	JES subsystem ID from the SYSID= * Parameter.	4
USRTYPE DS BL2	Zero for CPU time and elapsed time. Type code for authorization violation.	2
USERID DS CL8	SQL ID from AXPLSQID. Padded with blanks.	8
STTIME DS BL4	Time in BIN format from the TIME macro. This is the time the event started.	4
STDATE DS PL4	Date in BIN format from the TIME macro. This is the date the event started.	4
AGCPU DS BL4	Total CPU time used since the event started. The value is represented in milliseconds.	4
RECLN EQU *-CACSXSMF	Length of the standard SMF reporting record.	
ORG STTIME	Alternate record definition for authorization violation reporting.	
OBJNAME DS CL27	Name of the object for which the user is not authorized to access, define, grant, or revoke authority for.	27
TOTLEN EQU *-CACSXSMF	Length of the authorization violation SMF record.	

Table 23. Authorization violation type codes

Type Code	Authorization violation type
101	User is not authorized to issue a DROP TABLE for the requested table.
102	User is not authorized to issue a DROP INDEX for the requested table.
103	User is not authorized to issue a DROP VIEW for the requested view.
104	User is not authorized to issue a DROP PROCEDURE for the requested stored procedure.
200	User is not authorized to create the requested table.
201	User is not authorized to create the requested index
202	User is not authorized to issue the CREATE VIEW statement.

Table 23. Authorization violation type codes (continued)

Type Code	Authorization violation type
203	User is not authorized to issue the CREATE PROCEDURE statement
300	User is not authorized to issue a SELECT statement for the requested table or view.
301	User is not authorized to issue an UPDATE statement for the requested table.
302	User is not authorized to issue an INSERT statement for the requested table.
303	User is not authorized to CALL the requested stored procedure.
403	User is not authorized to issue a DELETE statement against the requested table.
500	User is not authorized to issue the requested GRANT statement.
501	User is not authorized to issue the requested REVOKE statement.

SMF exit: API overview

The SMF exit parameter list uses an AVA (Access Validation and Accounting) interface. The macro CACSXPL that is found in the SCACMAC library maps the parameters.

The AXPLFUNC field indicates the function you are calling:

- Initialization
- Accounting
- Authorization violations
- Termination

Table 24. SMF exit field definitions

Field	Definition
&DSECT=YES	Used to control whether a DSECT definition is generated or whether the fields are generated in the exit's local storage area.
AXPL DSECT	DSECT definition that is generated if &DSECT=YES is specified.
AXPL DS 0H	Label that is generated when &DSECT=YES is not specified.
AXPLID DC CL4'AXPL'	Identifier that should be checked to determine whether the internal storage area of the AXPL parameter list has been corrupted.
AXPLUSER DS 3A	Available for SMF exit use.
AXPLETYP DS F	Event type.
AXPLESUB DS F	Event sub-type.
AXPLESEQ DS F	Event sequence.
AXPLEBEF EQU 1	Before event execution.
AXPLEAFT EQU 2	After event execution.
AXPLSQID DS CL8	User ID left-justified, blank-filled.
AXPLPENV DS A	Pointer to event specific information.
AXPLTEXT DS A	Pointer to a text buffer (this is usually an SQL statement or the name of the object for which authorization failed).
AXPLTXL DS F	The decimal length of the text buffer.
AXPLSQLC DS F	The SQLCODE from the SQLCA after processing for the SQL event is completed.
AXPLFUNC DS H	Function identifier flag.
AXFNINIT EQU 0	Initialization.
AXFNVALI EQU 4	Validation and accounting.

Table 24. SMF exit field definitions (continued)

Field	Definition
AXFNTERM EQU 8	Termination.

SMF exit: initialization:

The SMF exit initialization function is called immediately after the exit is loaded during initialization of the query processor task when a user connects to the service.

The AXPLPENV parameter points to an (optional) input parameter string that can be passed to the exit. The input parameters are specified on the SMFEXIT configuration parameter used to invoke the SMF exit. Additional sub-parameters can be placed on the SMFEXIT parameter after the name of the exit.

The exit performs initialization processing and allocates any storage or other resources that are required for validation and accounting processing. A pointer to the anchor for these resources can be placed in the AXPLUSER field of the parameter list. This pointer is preserved and passed to the exit on subsequent invocations. Upon initialization, the AXPLSQID parameter contains the user ID of the user connecting to the query processor. The contents of the other fields in the parameter list are indeterminate.

If the exit returns a non-zero return code, query processor task initialization is stopped, the user is disconnected, and the return code issued by the exit is returned to the client application.

SMF exit: validation and accounting:

This section describes SQL statement types client applications can use, correlations between SQL statement types and SMF parameters, and validation and accounting activities the SMF function performs.

This SMF exit function is called at predetermined processing points. The AXPLETYP, AXPLESUB, and AXPLESEQ fields uniquely identify each processing point. The SMF exit is called to process SQL events that are identified by an AXPLETYP value of 3. The AXPLESUB field identifies the type of SQL statement that the client application has issued. The client application can issue the following types of SQL statements:

Table 25. SQL statement types

AXPLESUB value	Equate value	Type of SQL statement
1	DYNEXEC	Dynamic execute. When called at this point, the AXPLTEXT and AXPLTXTL fields will be zero.
2	CLOSE	Close cursor. When called at this point, the AXPLTEXT and AXPLTXTL fields will be zero.
3	DESCRIBE	Describe. When called at this point, the AXPLTEXT and AXPLTXTL fields will be zero.
4	EXECIMED	Execute immediate. When called at this point, the AXPLTEXT field will reference the statement being executed, and the AXPLTXTL field will identify the statement length.

Table 25. SQL statement types (continued)

AXPLESUB value	Equate value	Type of SQL statement
5	EXECUTE	Execute. When called at this point, the AXPLTEXT field will reference the statement being executed, and the AXPLTXTL field will identify the statement length.
6	FETCH	Fetch cursor. When called at this point, the AXPLTEXT and AXPLTXTL fields will be zero.
7	OPEN	Open cursor. When called at this point, the AXPLTEXT field will reference the statement being executed, and the AXPLTXTL field will identify the statement length if the client application is using static SQL. If the application is using dynamic SQL, the AXPLTEXT and AXPLTXTL fields will be zero.
8	PREPARE	Prepare statement. When called at this point, the AXPLTEXT field will reference the statement being executed and the AXPLTXTL field will identify the statement length.
9	SLCTINTO	Select into. When called at this point, the AXPLTEXT field will reference the statement being executed and the AXPLTXTL field will identify the statement length.

The AXPLESEQ field identifies whether the exit is being called before (1) (AXPLEBEF) or after (2) (AXPLEAFT) the SQL statement has been processed. The exit is called for each SQL statement issued by the client application. The user has control of the query processor service for the duration of the SQL statements execution.

Therefore, when the exit is called before the SQL statement is processed, it should obtain the TCB time for the current TCB. When called after the SQL statement has been processed, the exit should obtain the current TCB and compute the difference between the after and before SQL statement processing times. This value must be added to an aggregate value that the exits need to maintain for all SQL statements issued by the client application.

Example: The following sequence of SQL statements are issued for a dynamic SQL SELECT query:

- PREPARE
- OPEN
- DESCRIBE
- FETCH (until a SQLCODE of 100 is returned)
- CLOSE

To obtain the correct CPU time for the query, the exit needs to compute the CPU time used for each of these statements and add them together. Depending on the type of client application, you can issue different types of SQL statements. In the case of a client application that has more than one cursor open at a time, the individual SQL statements that are issued by the client application will be interleaved.

For this exit, the type of SQL statement being issued is not important unless the exit captures the text of the SQL statement being issued by the client application.

If the exit returns a non-zero return code, query processor task processing of the query is stopped and the return code issued by the exit is returned to the client application.

SMF exit: authorization violations:

This function processes authorization violations when SQL security is active and the query processor detects an authorization violation.

The AXPLETYP contains a value of 5. The AXPLESUB field identifies the type of authorization exception. The AXPLTEXT field identifies the name of the object for which authorization failed, and the AXPLTXTL field contains the length of the name in AXPLTEXT.

The exit reports the violation exception, and generates the alternate form of the SMF record shown in Table 22 on page 168.

SMF exit: termination:

This exit function performs any termination processing necessary and frees any resources it has allocated when the user disconnects from the service task. The termination function is called during query processor task termination processing.

For example, the SMF exit generates the SMF user record to report the CPU time used in milliseconds. The SMF record also contains the time and date when the user connected to and disconnected from the query processor task.

CPU resource governor

The CPU resource governor exit restricts the amount of CPU time that a user consumes for a particular unit of work.

Execution governor limits are based on the number of data rows examined (which is the number of calls issued to the connector interface) and the number of rows returned in a result set after all query post-processing is complete. The governors IBM delivers are fairly coarse. Depending on the query, a large amount of CPU time can be expended before one of these limits is reached.

On activation, the system passes the CPU resource governor exit the allowed CPU time for the user. Periodically, the CPU resource governor exit is called to check to see how much CPU time has been used. After exceeding the allotted time, the exit returns a return code that stops the query. The system controls the frequency at which it calls the exit.

The CPU resource governor exit is responsible for determining the unit of work based upon the series of SQL statements that the client application issues.

For example, the CPU GOVERNOR exit observes the following rules:

- The unit of work is from the first received PREPARE until all cursors have been closed. For a typical client application that only issues one query at a time, the unit of work is the duration of a single query. If multiple cursors are opened the unit of work persists until all cursors have been closed. Any additional SQL statements issued by the client while a query is active (for example, a SELECT INTO) are treated as part of the unit of work.
- If no queries are active, the exit treats any other SQL request as a single unit of work. For example, if no cursors are open, the exit treats requests such as SELECT INTO and EXECUTE IMMEDIATE as a single unit of work.

Activating the CPU resource governor exit

By configuring, activating, and validating the CPU resource governor exit, you can restrict the amount of CPU time that a user can consume for a unit of work.

Before you begin

- Install the product, perform initial configuration customization, and verify the installation configuration using the sample application and data.
- Include the CPU resource governor exit load module (CACSX03) in the load library that the data server is using.

Although the CPU resource governor exit does not require APF authorization, the SAF, SMF, and Workload Manager exits do. If you are running any of the latter exits, the CPU resource governor exit must also be placed in an APF-authorized load library.

About this task

The steps in the following procedure describe how to configure, activate, and validate the CPU resource governor exit:

Procedure

1. Edit the sample CPU governor exit (SCACCONF member CACSX03).
2. You can supply the following values on the CPUGOVERNOR configuration parameter.
 - a. Specify the CPU resource governor exit name.
 - b. Specify **Maximum CPU Time**.

This field specifies the maximum amount of CPU time that a single query can take (or a group of queries can take) in a multiple cursor situation. The following are valid values:

<i>nS</i>	The number of seconds, where <i>n</i> = number is a value between 1 and 6000.
<i>nM</i>	The number of minutes, where <i>n</i> = number is a value between 1 and 6000.
 - c. Uncomment the CPUGOVERNOR parameter.
 - d. Save your changes.
3. Start the data server.
 - If the data server is already running, stop and restart the query processor service for data source CACSAMP. This can be done using the MTO Interface.
4. Uncomment any SELECT statements in SCACSKEL member CACSQ1 that reflect the data source you are using.
 - a. Ensure that the user ID is authorized to access the table that is referenced in the query.
 - b. Save your changes.
5. Edit SCACSAMP member CACCLNT by setting the SQLIN parameter to reference CACSQ1, and save the member.
6. Submit CACCLNT and review the output.

The query should function normally and return the expected result set. You are ready to validate the CPU resource governor exit.

7. Set the CPU time limit specified in the CPUGOVERNOR parameter artificially low, to induce a timeout.
 - a. Save your changes.
8. Stop and restart the data server, then submit CACCLNT.
 - Alternatively, you can restart the query processor service for data source CACSAMP using the MTO interface.

The query should not return a result set, and the following error message should appear:
CPU time exceeded.

If the time limit is not exceeded you will see the normal result set.
9. Increase the CPU limit specified on the CPUGOVERNOR parameter or comment out the CPU GOVERNOR parameter, so you can run normal queries.
10. Stop the data server and restart it.
 - Alternatively, you can stop and restart the query processor service for data source CACSAMP using the MTO interface.

CPU resource governor exit: API overview

This API passes field values in a parameter list to the CPU resource governor exit. The table lists and describes the fields.

The parameter list uses an AVA (Access Validation and Accounting) interface and is mapped by the macro CACSXPL, which is found in the SCACMAC library. The AXPLFUNC field indicates the function you are calling:

- Initialization
- Accounting
- Termination

Table 26. CPU resource governor exit fields and descriptions

Field	Description
&DSECT=YES	Used to control whether a DSECT definition is generated or whether the fields are generated in the exit's local storage area.
AXPL DSECT	DSECT definition that is generated when &DSECT=YES is specified.
AXPL DS 0H	Label that is generated when &DSECT=YES is not specified.
AXPLID DC CL4'AXPL'	Identifier that should be checked to determine whether the internal storage area of the AXPL parameter list has been corrupted.
AXPLUSER DS 3A	Words available for CPU resource governor exit use.
AXPLETYP DS F	Event type.
AXPLESUB DS F	Event sub type.
AXPLESEQ DS F	Event sequence.
AXPLEBEF EQU 1	Before event execution.
AXPLEAFT EQU 2	After event execution.
AXPLSQID DS CL8	User ID left-justified, blank-filled.
AXPLPENV DS A	Pointer to event specific information.
AXPLTEXT DS A	Pointer to a text buffer (this is usually an SQL statement).
AXPLTXTL DS F	The decimal length of the text buffer.
AXPLSQLC DS F	The SQLCODE from the SQLCA after processing for the SQL event is completed.
AXPLFUNC DS H	Function identifier flag.

Table 26. CPU resource governor exit fields and descriptions (continued)

Field	Description
AXFNINIT EQU 0	Initialization.
AXFNVALI EQU 4	Validation and accounting.
AXFNTERM EQU 8	Termination.

CPU resource governor exit: initialization:

This function performs initialization processing and allocates any storage or other resources that are required for validation processing. The initialization function is called when a user connects to the service, immediately after the exit is loaded during initialization of the query processor task.

The AXPLPENV parameter points to an optional input parameter string that can be passed to the exit. The input parameters are specified on the CPUGOVERNOR configuration parameter that is used to invoke the CPU resource governor exit. On the CPUGOVERNOR parameter additional sub-parameters can be placed after the name of the exit.

A pointer to the anchor for required resources can be placed in the AXPLUSER field of the parameter list. This pointer is preserved and passed to the exit on subsequent invocations. Upon initialization, the AXPLSQID field contains the user ID connecting to the query processor. The contents of the other fields in the parameter list are indeterminate.

If the exit returns a non-zero return code, query processor task initialization is halted, the user is disconnected, and the return code issued by the exit is returned to the client application.

CPU resource governor exit: validation and accounting:

The validation and accounting function captures and computes aggregated CPU time and stops query processing when queries have exceeded the time limit for the prescribed unit of work.

The function is called at predetermined processing points. The AXPLETYP, AXPLESUB, and AXPLESEQ fields uniquely identify each processing point. The exit is called for each SQL statement issued by the client application. These events are identified by an AXPLETYP of 3. The different types of SQL statements are identified in the AXPLESUB field. The AXPLESEQ field identifies whether the exit is called before (1) or after (2) the SQL statement has been processed.

Exits that are called for SQL events must determine (based on the SQL statement type) whether the client application is beginning a unit of work, ending a unit of work, or is in the middle of a unit of work.

The exit performs the following actions:

- Start unit of work: Reset aggregate CPU time, capture current CPU time, and enter the middle of the unit of work.
- End unit of work: Reset aggregate CPU time and prepare to start the next unit of work.
- Middle of unit of work: After the exit determines that it is in the middle of a unit of work, the current CPU time should be obtained before the SQL statement

is issued . After the SQL statement has been executed, the exit is called and captures the current CPU time and computes the amount of time taken by that SQL statement. The exit maintains an aggregate CPU time for the unit of work, which represents the sum of the individual SQL statement execution times. The exit checks this aggregate time after an SQL statement has been processed to determine whether the CPU time limit has been exceeded.

When called to process SQL statements, the exit behaves in the following ways:

- If the exit has already generated a CPU time exceeded error when it was called during machine execution (described below), it does not generate another error. The exit will still be called to perform after SQL statement processing and sets a flag that indicates an error has been reported.
- If, after an SQL OPEN statement has been processed, the AXPLSQLC field contains a non-zero value indicating that an error was detected, the system does not call the exit to close the cursor associated with the statement that failed. Typically this will be a -204 or -206 error, or might be an error generated by any system exit. In these situations the exit must check to see what state the unit of work is in to determine whether it should reset its unit-of-work state. If there are no other cursors open, the exit needs to perform end unit-of-work processing.

The exit will also be called while the SQL program (that was generated for the query) is executing. The AXPLETYP field will contain a 4 in these situations. Currently, the exit is called at the two points where the MAX ROWS EXAMINED and MAX ROWS RETURNED governor checks are performed. The AXPLESUB field contains the machine instruction number that identifies the instruction being executed, and the AXPLESEQ contains a 1.

The exit does not consider the instruction type being executed. When called with an AXPLETYP value of 4, the exit should capture the current CPU time and compute how much time has elapsed since processing of the SQL statement started. This value should be (temporarily) added to the aggregate CPU time for the current unit of work. If the time limit is exceeded, the exit issues a return code to halt query processing.

The CPU resource governor exit uses an AVA interface, which is more complex than a custom interface like the SAF interface. An AVA interface accommodates additional call points.

If the exit returns a non-zero return code, the query processor task terminates execution of the current query, and the return code issued by the exit is returned to the client application.

CPU resource governor exit: termination:

The termination function performs any termination processing necessary, and frees any resources it has allocated. The termination function is called during query processor task termination processing when the user disconnects from the service task.

Workload Manager exit

The WLM exit tracks units of work and manages that work in goal or compatibility mode.

The Workload Manager (WLM) exit is used to interface with the z/OS Workload Manager.

A unit of work is the execution of an SQL statement that a client application issues. Table 25 on page 170 identifies the different types of SQL statements that a client application can issue. When the client issues one of these SQL statements, that user has control of the query processor service thread for the period that the query processor takes to service that SQL statement.

The WLM exit supports most of the parameters accepted by the IWMCLSFY macro in order to identify the service class that will be used to manage and/or report on the individual units of work. The WLM exit classifies these units of work based on the query processor service definition during query processor TCB initialization processing. Therefore, all users being serviced for a data source are managed in the same service class. Exit points are available at the user connection level that enable a customized exit to manage individual users of a query processor service, but the WLM exit performs no processing at these exit points.

Required: The WLM macros require that you execute the WLM exit from an APF-authorized load library. Therefore, the data server and all associated load modules must also reside in an APF-authorized load library.

Activating the WLM exit

The Workload Manager exit reports unit-of-work activities and manages queries in WLM goal mode.

Before you begin

- Install the product, perform initial configuration customization, and verify the installation and configuration using the sample application and data.
- Include the WLM exit load module (CACSX06) in the load library that the data server is using. Based on the previous examples, the WLM exit must be in an APF-authorized library (SCACLOAD).
- Ensure that the data server JCL references the APF-authorized library in the STEPLIB DD statement where the WLM exit is located (SCACLOAD). Ensure that any other data sets referenced in the STEPLIB DD statement are also APF-authorized.

About this task

The WLM exit is activated by defining the WLM service. The WLM exit must be able to handle all of the users accessing the data server concurrently.

Procedure

1. Define the WLM service during the installation customization process by specifying the parameters in the WLM section of the customization parameters file.

Otherwise, you can define the service by using the Classic Data Architect to add the service or with the MTO ADD command as shown in the following example:

```
ADD,CONFIG,SERVICE=WLMSEV,SERVICECLASS=WLM
```

2. The following guidelines apply to defining values for some of the configuration parameters for the WLM service.

Exit name

Specifies the name of the WLM exit to be invoked. The WLM exit name is CACSX06. Any parameters that the exit needs can be specified in the task data field (after the exit name). The exit name and its input parameters must be separated by a space or comma.

SUBSYSTYPE

It is preferable to define a subsystem type to WLM. SUBSYSTYPE can be up to four bytes in length and must conform to WLM subsystem types. Modify the subsystem type to a type that is valid on your system. Examples are: IMS, CICS, JES, and STC. If the address space of the query processor is a started task, STC might be a good choice.

SUBSYSNAME

This name and the subsystem type are used to connect to WLM and classify work received. Using IMS as an example:

```
SUBSYSTYPE=IMS SUBSYSNAME=IMSA
```

The WLM service cannot be stopped if any query processor services are active. When the WLM service is stopped, the address space-level control block is freed. During normal shutdown, the data server stops tasks in LIFO (last-in, first-out) sequence. The WLM service must be defined before any query processor services to allow the WLM service to terminate after all query processor tasks have completed termination processing.

- a. Uncomment the WLM service definition and supply the following sub parameters in the task data field. Parameters are specified using a *keyword=value* format and are comma delimited.

Exit name

Specifies the name of the WLM exit to be invoked. The WLM exit name is CACSX06. Any parameters that the exit needs can be specified in the task data field (after the exit name) on the WLM service definition. The exit name and its input parameters must be separated by a space or comma.

SUBSYS=xxxx

Specifies the generic subsystem type under which the unit of work is reported in WLM. It is preferable to define a subsystem type to WLM. The SUBSYS type can be up to four bytes in length and must conform to WLM subsystem types. Modify the subsystem type to a type that is valid on your system. Examples are: IMS, CICS, JES, and STC. If the address space of the query processor is a started task, STC might be a good choice.

SUBSYSNM=xxxxxxxx

Specifies the name of the subsystem that the unit of work is reported under in WLM. The SUBSYSNM name can be up to eight bytes in length. This name and SUBSYS (subsystem type) are used to connect to WLM and classify work received. Using IMS as an example:

```
SUBSYS=IMS SUBSYSNM=IMSA
```

The WLM service definition must come before any query processor service definitions. Failure to do so will result in a S0C4 abend when the data server is stopped. Additionally, the WLM service definition cannot be stopped if any query processor services are active. If the WLM service is stopped and a query processor service (task) is active, a S0C4 abend occurs when the query processor task is stopped. These abends occur when the WLM exit is called during WLM service initialization processing, which allocates and references an address space-level control block when the WLM exit is called from a query processor task. When the WLM service is terminated, the address space-level control block is

freed. During normal shutdown, the data server terminates tasks in LIFO (last-in, first-out) sequence. The WLM service definition must be defined before any query processor service definitions to allow the WLM service to terminate after all query processor tasks have completed termination processing.

- b. Save your changes.
3. Edit the service configuration member SCACCONF member CACSWLM.
- a. Uncomment the WLM UOW parameter and specify any desired subparameters. Subparameters are specified using a *keyword = value* format and are comma delimited. The following optional subparameters are supported by the WLM exit:

ACCTINFO=xxx...

Specifies accounting information. A maximum of 143 characters of accounting information can be supplied. The default is NO_ACCTINFO.

COLLECTION=xxx...

Specifies a customer-defined name for a group of associated packages. The maximum collection name that is supported is 64 characters long. The default is NO_COLLECTION.

CORRELATION=xxx...

Specifies the name associated with the user or program creating the work request, which resides within the network. The maximum correlation name that is supported is 64 characters long. The default is NO_CORRELATION.

LUNAME=xxxxxxxx

Specifies the name of the local LU name associated with the requestor. The maximum LU name is 8 characters long. The default is NO_LUNAME.

NETID=xxxxxxxx

Specifies the network identifier associated with the requestor. The maximum identifier is 8 characters long. The default is NO_NETID.

PACKAGE=xxxxxxxx

Specifies the package name for a set of associated SQL statements. The maximum package name is 8 characters long. The default is NO_PACKAGE.

PERFORM=xxxxxxxx

Specifies the performance group number (PGN) associated with the work request. If specified, the performance group number value must be within the range of 1-999, represented as character data, left-justified, and padded with blanks. The default is NO_PERFORM.

PLAN=xxxxxxxx

Specifies the name of an access plan for a set of associated SQL statements. The maximum plan name is 8 characters long. The default is NO_PLAN.

PRCNAME=xxxxxxxxxxxxxxxx

Specifies the name of a DB2 stored procedure associated with the work request. The maximum name that can be supplied is 18 characters long. The default is NO_PRCNAME.

PRIORITY=nnnnnnnnnn

Specifies the priority associated with the work request. The priority is specified as a decimal number. The maximum permitted value is 2147483647. The default is NO_PRIORITY (x80000000).

SUBSYSPM=xxx...

Specifies character data related to the work request. This information is passed to the workload manager for use in classification. A maximum of 64 characters of information can be supplied. The default is NO_SUBSYSPM.

TRXCLASS=xxxxxxxx

Specifies a class name within the subsystem that the workload manager recognizes. The maximum transaction class name that can be supplied is 8 characters long. The default is NO_TRXCLASS.

TRXNAME=xxxxxxxx

Specifies a transaction name for the work request that the workload manager recognizes. The maximum transaction name that can be supplied is 8 characters long. The default is NO_TRXNAME.

- b. Save the service configuration member.

Restriction: In WLM goal mode, a maximum of 254 characters of input parameters can be specified on the WLMUOW configuration parameter. The priority for units of work should be less than VTAM and IMS. The discretionary goal might result in very slow response times. Performance periods allow you to define a high number of service units for short transactions and a smaller number for long-running transactions. See the z/OS documentation about workload management for information about how to define service classes and classification rules.

4. Start the data server.
 - If the data server is already running start the WLM service, restart the query processor service for data source CACSAMP using the MTO interface.
5. Uncomment the SELECT statements in SCACSKEL member CACSQL that reflect the data source you are using.
 - a. Ensure that the user ID is authorized to access the table that is referenced in the query.
 - b. Save your changes.
6. Edit SCACSAMP member CACCLNT to set the SQLIN parameter to reference the SCACSAMP member CACSQL, and save your changes.
7. Start RMF™ data gathering if it is not already active.
 - a. Enter START RMF from the system console.

See z/OS documentation for information on using RMF.
 - b. Start data gathering with Monitor III ISPF panel from TSO.
8. Start the RMF Monitor III report before you start the sample query. This report processes data written to VSAM data sets by RMF.
 - a. Access RMF Monitor III interactively from an ISPF panel.

The SYSRTD report option reports response time distribution by service class and period. You should see the query response time in this report by the service class you selected.
9. Submit CACCLNT and review the output.

The query should function normally and return the expected result set. A snapshot of how WLM is managing this workload can be obtained by running the RMF postprocessor workload activity report during execution while the sample is running.

10. Run the RMF Monitor II interactive report.

- a. Enter RMF from a TSO session.
- b. Select option 2 for RMF Monitor II from the menu.

This report displays the activity only as it occurs, so the query must be a long one to see its activity.

11. Optional: Run the SYSSUM Sysplex summary report.

This report shows goals versus actual for service class periods when the system is in goal mode.

WLM exit: API overview

This API passes fields in a parameter list to the Workload Manager exit. The table lists and describes the fields.

The parameter list passed to the WLM exit uses an AVA (Access Validation and Accounting) interface and is mapped by the macro CACSXPL, which is found in the SCACMAC library. The AXPLFUNC field indicates the function you are calling:

- Initialization
- Management and reporting
- Termination

Table 27. WLM exit fields and descriptions

Field	Description
&DSECT=YES	Controls whether a DSECT definition is generated or whether the fields are generated in the exit's local storage area.
AXPL DSECT	DSECT definition that is generated when &DSECT=YES is specified.
AXPL DS 0H	Label that is generated when &DSECT=YES is not specified.
AXPLID DC CL4'AXPL'	Identifier that should be checked to determine whether the internal storage area of the AXPL parameter list has been corrupted.
AXPLUSER DS 3A	Words available for CPU Resource Governor exit use.
AXPLETYP DS F	Event type.
AXPLESUB DS F	Event sub type.
AXPLESEQ DS F	Event sequence.
AXPLEBEF EQU 1	Before event execution.
AXPLEAFT EQU 2	After event execution.
AXPLSQID DS CL8	User ID left-justified, blank-filled.
AXPLPENV DS A	Pointer to event specific information.
AXPLTEXT DS A	Pointer to a text buffer (this is usually an SQL statement).
AXPLTXTL DS F	The decimal length of the text buffer.
AXPLSQLC DS F	The SQLCODE from the SQLCA after processing for the SQL event is completed.
AXPLFUNC DS H	Function identifier flag.
AXFNINIT EQU 0	Initialization.

Table 27. WLM exit fields and descriptions (continued)

AXFNVALI EQU 4	Management and Reporting.
AXFNTERM EQU 8	Termination.

WLM exit: initialization

This function performs initialization processing and allocates any storage or other resources that are required for subsequent processing. The initialization function is called by the WLM initialization service during data server address space initialization.

The AXPLTEXT parameter points to any additional parameters that were specified in the TASKPARM configuration parameter. The AXPLTXTL parameter identifies the length of the input parameters. If no parameters are supplied, the AXPLTEXT and AXPLTXTL field values are zero.

The WLM exit might be called multiple times to perform initialization and termination processing, if the WLM initialization service is stopped and then restarted using the MTO Interface.

The exit also registers itself with WLM and receives a WLM token that the exit passes on subsequent calls. You can place a pointer to the anchor for these resources in the AXPLUSER field of the parameter list. This pointer is preserved and passed to the exit on subsequent invocations. Upon initialization, the contents of the other fields in the parameter list are indeterminate.

The AXPLUSER field consists of three fullwords that the exit can store anchor blocks in. The intent is to allow the exit to store an anchor for address-space-level storage acquired during initialization processing. The second fullword should be used to store an anchor block for storage that was acquired during query processor service initialization. This storage is *TCB-level* storage and should be allocated for each instance of a query processor service. The third fullword is available if the exit needs to allocate additional storage to manage an individual user.

When the exit is called to perform TCB initialization the AXPL storage area passed for initialization processing is cloned and the new copy is passed to the exit. If the exit stores an anchor block in one of the AXPLUSER fullwords, that address is local to the TCB being serviced. Similarly, when the exit is called to service a connection request, a copy of the TCB-level AXPL storage area is cloned and the new copy is passed to the exit. This copy is passed on subsequent calls to the exit to service individual SQL statements issued by the client application.

If the exit returns a non-zero return code, the WLM initialization service is terminated and a message written to the log. No additional call will be made to the WLM exit.

WLM exit: management and reporting

This overview lists the processing points where WLM exit management and reporting functions can be called and the type of processing they perform.

The WLM exit is called at predetermined AXPLETYP, AXPLESUB and AXPLESEQ processing points.

TCB initialization and termination:

The WLM exit initialization and termination functions perform TCB-level initialization and termination processing.

The WLM exit is called with an AXPLETYP value of 1 (AVAETCB) during query processor initialization and termination for each query processor TCB.

The AXPLESUB field identifies whether the exit is being called for:

1. Initialization processing (TCBINIT) or
2. Termination (TCBTERM).

The AXPLTEXT field contains a pointer to the WLM UOW configuration parameter value. The AXPLTXL field contains the length of the configuration parameter input. The AXPLPENV field contains a point to the data source name of the query processor being activated. The AXPLSQID is blank.

For example, on TCB initialization the WLM exit runs the WLM classify macro (IWMCLSFY) using the Workload Manager unit of work (WLM UOW) parameters supplied (if any). The WLM exit then runs the macro IWMECREA to create an enclave TCB environment. On the IWMECREA macro the FUNCTION_NAME parameter is set to the first eight characters of the data source name. At TCB termination the exit issues an IWMEDELE to delete the enclave.

If the exit allocates a TCB-level control block and stores it in the AXPLUSER area on initialization processing, then this storage must be freed when the exit is called to perform TCB termination processing.

On initialization processing, if the exit issues a non-zero return code, service initialization is stopped and the service is not started. On termination processing, the return code is ignored and normal termination process is continued. In either case, the return code value is logged.

The TCB initialization call is actually deferred until the first user connects to a query processor task. This allows the exit to perform the IWMCLSFY and IWMECREA calls and not violate the WLM recommendation that these two calls should be issued in rapid succession followed by an IWMEJOIN call to associate a unit of work with the enclave. Because these calls are not issued when a query processor task is physically started (only when the first user connects), the user typically issues an SQL request immediately, which causes an IWMEJOIN call to be issued.

The exit might defer creating an enclave environment until a user connect call is issued (discussed next). However, if the query processor is running in multi-user mode (Max. Users greater than 1) the exit has to manage context switches between users when the client application issues SQL statements.

User connect and disconnect:

The WLM exit functions described in this document perform user-level processing when you connect or disconnect from a query processor service.

The WLM exit is called with an AXPLETYP value of 2 (AVAEUSR) when a user connects or disconnects from a query processor service. The AXPLESUB field identifies whether the exit is being called for connect processing (CONNECT) or disconnect processing (DISC).

The AXPLTEXT field contains a pointer to the WLMUOW configuration parameter value. The AXPLXTL field contains the length of the configuration parameter input. The AXPLPENV field has a value of zero. The AXPLSQID contains the user ID of the user connecting to or disconnecting from the service.

The sample exit does not perform any processing for these call points.

If the exit issues a non-zero return code on initialization processing, the user is disconnected from the service and the return code is returned to the client application. On disconnect, the return code is ignored and all user related resources are freed, however, the return code is returned to the client application. In either situation, the WLM-generated return code is logged.

SQL statement processing:

The WLM exit performs any required actions to manage the unit of work that the SQL statement represents. For example, the WLM exit treats each SQL statement as a single unit of work and joins the enclave before the SQL statement is processed, then leaves the enclave after the statement has completed processing.

The AXPLESEQ field identifies whether the exit was called before (AXPLEBEF) or after (AXPLEAFT) the query processor has processed the SQL statement. The user has control of the query processor TCB (thread) for the duration of the SQL request. If running in multi-user mode, another user can be serviced on the next SQL statement received by the query processor. The WLM exit is called with an AXPLETYP value of 3 (AVAESQL) when a client application issues an SQL statement. The AXPLSQID field contains the user ID of the client issuing the SQL statement.

The duration of an SQL statement varies based on the type of SQL statement being issued and other configuration parameter values. The configuration parameter that has the most impact is PDQ. When PDQ is not active or when a query cannot be processed in PDQ mode, then the OPEN SQL statement will execute the longest because the entire result set is staged for fetch processing. In these instances any describe, fetch, and close cursor requests will execute very quickly. When running in PDQ mode the work is more evenly distributed between the open and the fetches because the result set is incrementally built based on the number of rows that can fit into the result set buffer.

If the exit issues a non-zero return code, processing for the query is stopped and the return code is returned to the client application. The application can still issue another SQL request. There are situations where the exit-generated return code will not be reported to the client. This occurs when the exit is called to perform **after** processing, but an error code has been reported by another exit or generated by the system. In these cases the **original** error code takes precedence. In either case, the WLM-generated return code is logged.

WLM exit: termination

The WLM exit termination function performs any required termination processing and frees allocated resources. The function is called during WLM termination processing, before the data server address space is shut down.

If the exit allocated an address space level control block and stored it in the AXPLUSER area, that storage must be freed.

The WLM exit can be called multiple times to perform initialization and termination processing, if the WLM initialization service is stopped and then restarted using the MTO Interface.

DB2 thread management exit

The DB2 thread management exit enables you to validate clients before you establish connections to DB2 using CAF and control the duration of these CAF connections.

The DB2 thread management and security exit modifies the default behavior of connecting to and disconnecting from DB2. In addition, the thread management exit performs SAF calls to validate the user ID of the client and establishes the correct primary authorization ID for the client in DB2.

The DB2 thread management exit issues RACROUTE calls and must be run from an APF-authorized library. Additionally, when issuing the RACROUTE calls, the exit enters key zero supervisor state. The exit reverts back to user key problem state immediately after each RACROUTE call returns.

The DB2 thread management exit runs under the DB2 Call Attachment Facility (CAF) service in the data server. The CAF service runs as a z/OS subtask under the data server, whose sole responsibility is to create and manage CAF connections to DB2. One instance of the subtask is required for each concurrent DB2 user.

Required: A minor modification to the DB2-supplied authorization exit DSN3SATH is required to establish the correct primary authorization ID in DB2.

By default, connections to DB2 itself using CAF are created when a client connects to DB2 and remains active until the data server is shut down. While this maximizes re-usability of the DB2 connections, the DB2 primary authorization ID for all connections is based on the data server's started task or job name. In many cases, this level of security checking will not be adequate for your particular installation.

After activation, the DB2 thread management and security exit is invoked to perform the following functions:

- Initialization is called once at initial subtask start up.
- Client Connection is called each time a new client has acquired the CAF subtask.
- Another Connection to DB2 is called after each attempt is made to connect to DB2.
- Client Disconnection is called each time a client has released its connection to the CAF subtask.
- Termination is called once at subtask termination.

The exit CACSX07 performs the following functions at each of these invocation points:

- Initialization requires no processing. The exit returns a successful return code.
- Client Connection validates that the user ID and password for the incoming client are valid using a SAF call and establishes an ACEE control block for the TCB so the DB2 identified authorization exit can establish the correct primary authorization ID prior to requesting a connection to DB2.
- After Connection to DB2, the exit deletes the ACEE that was established when the client connection request was issued.
- Termination requires no processing. The exit returns a successful return code.

Activating the DB2 thread management exit

When you configure the DB2 thread management exit you can validate clients prior to connecting to DB2 using CAF and control the duration of the connections.

Before you begin

- Install the data server, perform initial configuration customization, and verify the installation and configuration.
- Ensure that the DB2 thread management exit load module CACSX07 is in an APF-authorized library (SCACLOAD).
- Ensure the data server JCL references the APF-authorized library in the STEPLIB DD statement where the DB2 thread management exit is located (SCACLOAD). Also ensure that any other data sets referenced in the STEPLIB DD statement are also APF-authorized.

Procedure

1. Edit the sample exit in SCACCONF member CACSX07.
2. Define values for the THREADMGMTEXIT configuration parameter on the CAF service.
3. Update the DB2-supplied sample identify authorization exit DSN3SATH and insert the assembler logic, found immediately before the label SATH019 in the SCACSAMP member CACSXDSN.
4. Reassemble the identity authorization exit DSN3SATH.

```
//*DSN3ADD PROVIDE VALID JOB CARD
/*
//DSN3ADD PROC CAC='CAC', CAC HIGH-LEVEL QUAL
// DB2='DB2' DB2 HIGH-LEVEL QUAL
//ASSEMBLE EXEC PGM=ASMA90,PARM='LIST,NODECK,RENT'
//SYSLIB DD DISP=SHR,DSN=&DB2..MACLIB
// DD DISP=SHR,DSN=&DB2..ADSNMACS
// DD DISP=SHR,DSN=&DB2..AMODGEN
// DD DISP=SHR,DSN=&CAC..SCACMAC
//SYSLIN DD DISP=SHR,DSN=&&TEMP
//SYSUT1 DD DSN=&&SYSUT1,UNIT=VIO,SPACE=(1700,(2000),,,ROUND)
//SYSPRINT DD SYSOUT=*
//SYSIN DD DISP=SHR,DSN=&DB2..ASM(DSN3SATH)
/*
//LINK EXEC PGM=IEWL,COND=(4,LT),
// PARM='LIST,OL,RENT,REUS,AMODE=31,RMODE=ANY'
//SYSLIN DD DISP=SHR,DSN=&&TEMP
// DD *
// NAME DSN@SATH(R)
//SYSLMOD DD DISP=SHR,DSN=&DB2..SDSNEXIT(DSN@SATH)
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(120,120),,,ROUNT),DCB=BUFNO=1
//SYSPRINT DD SYSOUT=*
//
```

5. If your DB2 SDSNEXIT library is in the z/OS linklist, refresh the linklist with the z/OS **F LLA, REFRESH** command.
6. Start the data server.
 - If the data server is already running, restart it.
7. Verify the exit is working correctly by running a client that issues DB2 queries.
 - While the client is active, you can view the client's connection to DB2 by issuing the DB2 command **-DIS THD (*)** from a z/OS console or SDSF.

The z/OS LOG output from the command should display the data server as the ID of the thread owner and the client-supplied user ID as the primary authorization ID.

Customizing the DB2 thread management exit

Customizing the DB2 thread management exit enables you to validate clients and control the duration of CAF connections.

If your installation has special processing requirements that can be addressed by a DB2 thread management exit, you can update the exit IBM delivers or create a custom exit of your own. The following table describes the DB2 thread management parameter list and processing options for developing your own custom DB2 thread management exit.

The DB2 thread management exit is called to perform the following functions:

- **Initialization:** Each time a new DB2 CAF service is started from the region controller, an initialization call is made to perform any one-time initialization processing for the started task. If your exit must allocate any storage for use throughout the life of the task, your exit can allocate the storage in this call and place the address of that storage in the user field DB2TUSRW supplied in the exit parameter structure.
- **Client Connection:** Called each time a new client has acquired a CAF subtask. Typically, this call is made immediately before connecting to DB2 on behalf of the client. The one exception is when the exit is set to leave connections to DB2 active even when clients disconnect.
- **After Connection to DB2:** Called after each attempt is made to connect to DB2. This call is made regardless of whether the connect was successful. The parameters field DB2STAT indicates whether the connection attempt was successful.
- **Client Disconnection:** Called each time a client has released its connection to the CAF subtask.
- **Termination:** Called once at subtask termination. If you allocated any exit memory at initialization, this is when it needs to be freed.

The parameters listed in the following table are located in the PDS member CACSXPL7 of the SCACMAC library:

Table 28. DB2 thread management exit parameters

Field	Description
DB2TUSRW	Initialized to binary zeros prior to the initialization call and left unchanged after that point in processing. Use of this field is determined by the user exit.
DB2TUSRP	Points to a NULL terminated user parameter as defined on the service definition for the DB2 CAF task. This parameter includes any text included after the exit name itself in the task data field. For example, to pass the string USERPARM to the exit from the service definition, the task data for the exit is: CACPLAN,CACSX07 USERPARM

Table 28. DB2 thread management exit parameters (continued)

Field	Description
DB2TSSN	Points to the four-character subsystem name as defined in the Service Name field of the service definition the task. In most cases, this field is for informational purposes only. However, the exit can change this field on client connection calls to designate a new subsystem to connect to, if necessary. If this field is updated, it will remain updated until the thread disconnects from DB2. At that point, it will be changed back to its original service information entry value.
DB2TPLAN	Points to the 8-character DB2 PLAN name as defined in the Task Data field of the service definition for the task. In most cases, this field is for informational purposes only. However, the exit can change this field on client connection calls to designate a new DB2 PLAN to open a DB2 connection. If this field is updated, it remains updated until the thread disconnects from DB2. At that point, it is changed back to its original service definition value.
DB2TUID	Points to the user ID provided by the client when the client connected to the data server. This field is binary zeros on initialization and termination calls, as no user is available when these calls are issued. This field is for reference purposes only and must not be changed by the exit.
DB2UPWD	The DB2UPWD field points to the user password provided by the client when it connected to the data server. This field is binary zeros on initialization and termination calls, as no user is available when these calls are issued. This field is used for reference purposes only and must not be changed by the exit.
DB2TSTAT	The DB2TSTAT field identifies whether a current CAF connection exists to DB2. This field is used for reference purposes only.
DB2TFUNC	The DB2TFUNC field identifies the function of the call as described previously. Defined values for this field are: <ul style="list-style-type: none"> • DB2TINIT: Initialization function • DB2TCCON: Client Connection function • DB2TDB2C: DB2 Post Connection/Plan Open function • DB2TCDIS: Client Disconnect function • DB2TTERM: Termination function

Table 28. DB2 thread management exit parameters (continued)

Field	Description
DB2TRFNC	<p>The DB2TRFNC field can be used by the exit to:</p> <ul style="list-style-type: none"> • Explicitly request connection or reconnection to the DB2 subsystem. • Explicitly request disconnection from the DB2 subsystem. • Notify the system that a user ID or password validation error has occurred. <p>The field is used to alter the default DB2 subsystem connection and disconnection behavior in the DB2 CAF service. The default behavior is to connect to DB2 when the first user requests DB2 access and disconnect from DB2 at data server shutdown. The exit should set this field on each call to one of the following values:</p> <ul style="list-style-type: none"> • DB2TRDFL: Do default connection processing • DB2TRCON: Connect to DB2. If a connection already exists, terminate that connection and create a new connection using the subsystem and plan name in the fields DB2TSSN and DB2TPLAN. This value is valid for Initialization and Client Connection functions. • DB2TRDIS: Disconnect from DB2 if a connection exists. This value is valid for the Client Disconnect function. • DB2TRUER: User or password information is invalid. This value can be returned on the Client Connection function to return an Access Denied error to the requesting client.

The return code (register 15 value) for successful completion of the user exit should always be set to 0. Any other value causes an error message to be returned to the requesting client.

Activating a customized exit

You can follow the instructions for activating the DB2 thread management exit to activate your customized exit with the following modifications:

- Replace the system default exit name CACSX07 with the name of your customized exit.
- If your exit does not create a TCB level ACEE for the DB2 primary authorization ID setting, skip the instructions for updating and reassembling DSN3SATH.

Record processing exit

Use the record processing exit to modify the characteristics of a record to meet specific processing requirements. These requirements might include modifying record data or calling external programs.

Purpose

The record processing supports the following activities:

- Modifying VSAM records before publishing
- Compressing, decompressing, encrypting, and decrypting VSAM records
- Query filtering
- Calling external programs
- Processing sequential files

While it is possible to filter VSAM records with the record processing exit, it is best to define views on your tables for this purpose. See the topics “Creating native VSAM tables and views for change capture” and “Creating CICS VSAM tables and views for change capture”.

Programming considerations

A sample exit, CACSX08, is supplied in the sample library. The exit must be re-entrant, AMODE(31), RMODE(ANY). Registers must be saved on entry and restored on exit. The exit performs initialization, processing, and termination functions. Errors in the exit routine might affect the operation of the product as a whole.

The exit can be written in any language, so consider the performance of the chosen language because the data server calls the exit for each record.

Register contents at entry to the exit routine are as follows:

- R1 contains a pointer to a parameter list
- R13 points to a register save area in standard format
- R14 contains the return address
- R15 contains the entry point of the routine

Restore all other registers upon return from the exit. For information about input parameters, see the topic “Input parameters for the record processing exit”.

If you make changes to the exit, you must stop and restart the data server before the changes take effect.

Input parameters for the record processing exit

Programs that call the record processing exit pass address pointers to the parameters listed in the table.

Table 29. Record processing exit input parameters

Field	Description	Length
Function	INIT, UPDATE, PROCESS, or TERM. These values are padded on the right with spaces.	7 bytes
Table description	Name of the table being processed.	18 bytes
Input record	Record that was read.	
Input record length	Length of the input record.	Binary fullword
Output record	Record that the application is to process.	
Output record length	Length of the output area.	Binary fullword

Table 29. Record processing exit input parameters (continued)

Field	Description	Length
Return code	Completion code of the exit call. A return code set to zero indicates processing completed normally. A return code less than zero terminates the query. A return code greater than zero skips processing of the current record and reads the next record.	Binary fullword
User word	A word passed to the exit that can be used to anchor additional information.	Binary fullword

Record processing exit: initialization

Initialization processing can acquire any required system resources, such as allocating persistent memory or calling and initializing a decompression routine.

When the data server opens a table, the initialization function acquires the resources that your environment needs for later processing. Initialization processing occurs once for each query.

Record processing exit: process

The process function carries out the operations that you want to perform for each record.

In a Classic federation environment, the record processing exit calls the process function after the query processor reads a record. The query processor passes the record to the exit, which then modifies the record and rebuilds it in the output record area.

The output format of the record must match the metadata grammar that maps the table.

Record processing exit: termination

The data server calls the termination function when it closes a table, before it unloads the exit from memory.

The exit releases acquired resources and performs any required clean up tasks.

SQL updates to application data

When you update application data in a secure transaction environment, changes to data can be aggregated and committed as a single logical unit of work and can also be rolled back under application control or in the event of unexpected application errors.

You can use a set of SQL statements to update data and manage transactions. How each data source processes transactions varies based on the capabilities of the data source.

Transaction processing

A transaction is a sequence of one or more SQL statements that together form a logical unit of work. A transaction automatically begins with the first SQL statement that is run within a client connection to the data server.

Transactions continue through subsequent SQL statements until one of the following conditions occur:

- A client COMMIT statement ends the transaction and makes any database changes permanent.
- A client ROLLBACK statement aborts the transaction and backs out any database changes.
- A client disconnect occurs that results in a transaction commit.
- An unexpected client disconnect occurs that results in a transaction rollback. In addition to committing or rolling back changes, all open cursors that are associated with the transaction are closed and any prepared statements are freed.

After a transaction is completed by a commit or a rollback, a new transaction starts with the next SQL statement that the SQL client issues.

Some clients support an autocommit mode of processing that makes each SQL statement its own transaction by automatically issuing an explicit COMMIT statement after each SQL statement. In applications that require bundling multiple updates in the same transaction, this feature must be deactivated.

Transaction processing depends on the capabilities of the underlying connectors. In some cases, the connector does not allow an update because the data source does not support a method of committing or rolling back database changes.

The CA-Datcom, DB2 for z/OS, and IMS data sources support two-phase commit capabilities.

In general, it is good practice to issue a ROLLBACK statement when the data server returns a negative SQLCODE.

Attention: Database transactions automatically lock database resources and prevent concurrent access to other database users. Applications that issue updates must keep update transactions as short as possible to avoid contention for database resources.

SQL update statements


This topic describes the SQL statements used to update application data and manage transactions in the data server. Each of these statements can be dynamically prepared and executed or executed immediately by client applications.

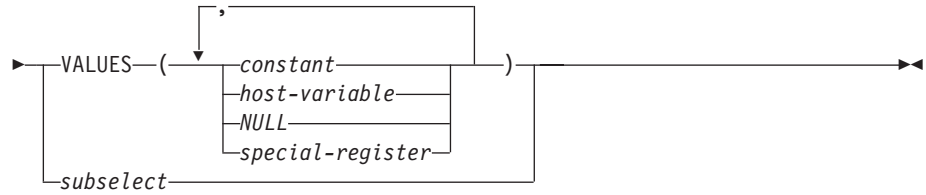
With the exception of COMMIT and ROLLBACK, you can run prepared update statements multiple times within the same transaction.

INSERT

The INSERT statement inserts one or more rows in an application database.

INSERT statement syntax

►►—INSERT INTO—*table-name* 

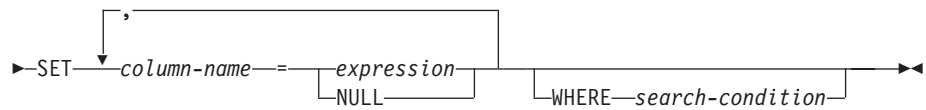
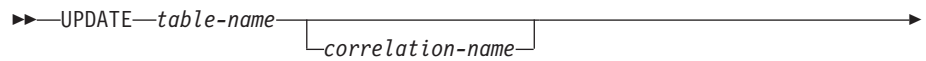


If you omit the column names from the statement, the values or subselect list must include data for all columns of the inserted table.

UPDATE

The UPDATE statement updates one or more rows in an application database.

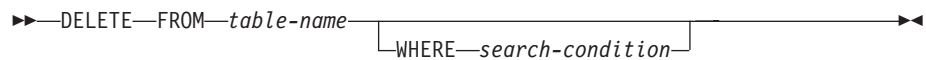
UPDATE statement syntax



DELETE

The DELETE statement deletes one or more rows in an application database.

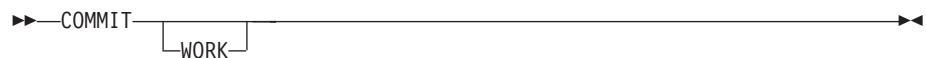
DELETE statement syntax



COMMIT

The COMMIT statement commits a transaction and makes any pending database changes permanent. In addition, all open cursors associated with the transaction are closed and all prepared statements are freed.

COMMIT statement syntax



ROLLBACK

The ROLLBACK statement rolls back a transaction and backs out any pending database changes. In addition, all open cursors associated with the transaction are closed and all prepared statements are freed.

ROLLBACK statement syntax



SQL updates and mapped tables

Tables that are mapped from nonrelational and relational databases to the metadata catalog contain some mapping constructs with special meaning that you need to be aware of when you update mapped tables.

Mappings that contain multiple records

With some connectors, such as IMS and CA-IDMS, you can map multiple records or segments in a database path to a single logical table in the metadata catalog. When you update these mappings with SQL update statements, updates are applied to only the last record that is mapped in the path.

Example: This mapping involves two different record types: EMPLOYEE and PAYCHECK.

```
EMPLOYEE RECORD:
    SSN             CHAR(9)
    LAST_NAME      CHAR(20)
    FIRST_NAME     CHAR(20)
PAYCHECK RECORD:
    PAY_DATE       DECIMAL(8,0)
    GROSS_PAY      DECIMAL(15,2)
    NET_PAY        DECIMAL(15,2)
    FED_TAX        DECIMAL(7,2)
    STATE_TAX     DECIMAL(7,2)
    FICA           DECIMAL(7,2)
```

In this example, an EMPLOYEE_PAY table maps the EMPLOYEE database path to PAYCHECK. The table also returns each employee record combined with each of the paycheck records for each employee in a standard SQL query. In the case of an update call, the updates apply to only the PAYCHECK record.

The following SQL statement inserts a new paycheck record:

```
INSERT INTO EMPLOYEE_PAY (SSN, PAY_DATE, GROSS_PAY, NET_PAY,
    FED_TAX, STATE_TAX, FICA)
VALUES( '012339920', 10311999, 4200.00, 3300.00,
    800.00, 75.00, 25.00 );
```

In this case, a value is provided for SSN even though the EMPLOYEE record itself is not updated. For INSERT statements, values that are provided for any records other than the last record in the mapping are used to qualify the parent or owner record under which the values are inserted.

These values are treated like foreign keys in a relational database. For example, if the employee with SSN 012339920 does not exist in a record in the database, SQLCODE -530 is returned because a nonvalid foreign (positioning) key is supplied.

Restriction: Inserts with subselects are not supported for multiple-record mapping.

The same concepts apply to UPDATE and DELETE statements. However, because both UPDATE and DELETE support WHERE clauses, foreign key qualification is placed in the WHERE clause itself.

For updates, SET statements must not include columns from any record other than the last record in the mapping. Otherwise, SQLCODE -4903 is returned.

Positions of inserted records

The position of inserted records is determined by any values that are supplied for records other than the last record in a mapped database path.

In single record mappings and multiple record mappings where no path information is supplied, the position of new records is determined by the underlying database system. Before you permit insert access on a mapped table, consider the underlying insert behavior to ensure that unqualified insert positioning by the database produces the desired results.

Record inserts with full and partial mapping

Database record mappings can include columns for all or only part of the underlying database record. When you insert records that contain partial mappings, the areas of the database record that are not mapped are initialized to binary zeros.

In cases where mapped columns from a target database record are omitted from an INSERT statement, the underlying data in the record is initialized as follows:

- NULL IS specification (if supplied)
- One of the following data type values:
 - SPACES for underlying character types
 - ZERO for numeric data types
 - Binary zeroes for VARCHAR data types

Updates and deletions of database records

The UPDATE and DELETE statements apply changes to all database records that meet the WHERE clause criteria specified. You need to specify WHERE criteria correctly, because unqualified updates and deletes change all instances of the target record in the database.

Updates and NULL records

UPDATE statements only update existing records in the database.

When you retrieve rows for tables that are mapped to multiple database records, SQL rows are returned even when the last record that is mapped in the table does not exist in the database within the mapped path. UPDATE statements are not applied to rows in which the last record is returned as all NULLs due to the lack of a database record in the database.

Mappings that contain record arrays

SQL updates are not supported on tables that contain record array mappings of OCCURS clauses. If you need to update records that contain multiple occurrences of data items, each occurrence must be mapped as a separate column to allow updates.

Group items and overlapping fields

Do not issue INSERT statements on table mappings that contain group items or overlapping fields. Inserts on tables that map group items or overlapping fields can produce unpredictable results due to initialization of unspecified columns in the underlying database record.

A *group item* refers to a single data item that is broken down into sub-items. For example, the group item ZIP-CODE can be subdivided into ZIP-FIRST-5 and ZIP-LAST-4 with the following COBOL definition:

```
05 ZIP-CODE.  
10 ZIP-FIRST-5 PIC X(5).  
10 ZIP-LAST-4 PIC X(4).
```

Unpredictable results are likely to occur when overlapped columns are defined with different SQL data types.

Update processing recommendations

When you set up update processing, consider using separate table mappings to define security and using the NULL IS parameter to define the string length for a column.

Usually, separate table mappings for update purposes are better than general mappings for both query and update. You should secure non-update mappings to prevent accidental use of those mappings for update.

When specifying NULL IS on columns in an update table, the NULL IS value should be the same length as the underlying database field to ensure proper initialization on INSERT.

Adabas updates

When you make SQL updates to Adabas data sources, use NISNHQ for large transactions.

If a single query updates a large number of records, ensure that the Adabas parameter NISNHQ is large enough to handle the number of records.

DB2 for z/OS updates

When you make SQL updates to DB2 for z/OS data sources, ensure you set up the appropriate authorization.

If you run the DB2 thread management exit, you must grant DB2 table update authority to each user ID that connects to the data server. Otherwise, you need to grant update authority to the data server task.

If an authorization problem causes DB2 to return an error, SQLCODE-9999 is returned with a message that the DB2 error is in the SQLEXT field. The SQLEXT field contains the value E551 that indicates a -551 error.

CA-Datcom updates

For CA-Datcom to support the rollback of transactions that update your source data, you need to configure your CA-Datcom data source.

About this task

You can specify INSERT, DELETE, or UPDATE statements to update CA-Datcom tables. To modify a table, the User Requirements Table in use when the update occurs must allow modifications to the specified table. You need to include the parameter UPDATE=YES on the DBURTBL macro for the table being modified.

SQL modification subsequently requires either a commit or rollback to delineate a unit of recovery. The commit or rollback can be explicit or implicit. The program or script that is run can issue an explicit COMMIT or ROLLBACK. The database generates an implicit COMMIT at the normal end of the process. The database generates an implicit ROLLBACK at the abnormal end of the process.

Enabling the rollback requires several additional steps.

To enable rollback in CA-Datcom, perform the steps in the following procedure.

Procedure

1. Enable the CA-Datcom logging system.
2. Identify the tables that are candidates for transaction backout. Transaction backout requires logging of all update (INSERT, DELETE, and UPDATE) transactions that affect a specific table.
3. Specify LOGGING=YES in the CA-Datcom Data Dictionary for all tables identified in the previous step, and catalog the definitions to the CA-Datcom Directory (CXX).
4. Specify TXNUNDO=YES in the DBURSTR macro for all User Requirements Tables used by programs that require transaction backout.
5. Assemble and link-edit all affected User Requirements Tables.

What to do next

To verify that the rollback is successful, you must issue an SQL statement to modify (INSERT, DELETE, or UPDATE) a CA-Datcom table, and then issue an explicit ROLLBACK statement. The ROLLBACK process issues message DB00103I that contains a return code value on the system log. Return code RC=Y indicates that the rollback completed successfully after backing out modified data.

Another way to verify the rollback is to query (SELECT) the row before modifications and after the ROLLBACK and compare the results. The rows should be identical.

CA-IDMS updates

With the CA-IDMS data connector, you can update CA-IDMS data by issuing standard SQL update statements to the data server.

Updates that are issued to CA-IDMS mapped tables apply to only the last record that is mapped in the database path.

Navigating CA-IDMS records requires the data server to issue the CA-IDMS @READY command for all database areas needed to access the CA-IDMS records mapped in the table. The CA-IDMS records mapped in the table and the system indexes defined for the first and last CA-IDMS records in the mapping determine the areas readied for processing a mapped table.

Updates of CA-IDMS data

An update to a CA-IDMS table results in an @MODIFY of the last record type mapped in the table. CA-IDMS processes a modify by implicitly updating any set connections for the modified record.

Inserts of CA-IDMS data

An insert to a CA-IDMS table results in an @STORE of the last record type that is mapped in the table. CA-IDMS processes a store operation by implicitly connecting the stored record to all sets in which the record is defined as an automatic member.

To automatically connect stored records, a record position for each owner record type must be established for each automatic set to which the stored record belongs.

If more than one record is mapped in the path, and if the last record is a member of multiple automatic set relationships (common in junction records), every OWNER record of the last record must be mapped in the defined path. In addition, define the set names among all records in the path as `_NONE_` so that positioning occurs for each OWNER record independent of whether the record belongs to any particular set.

Owner records are positioned by searching for the first owner record that matches insert values provided for columns defined in the owner record. You must be careful about providing enough qualification information to uniquely identify the correct owner of the record to be inserted. Otherwise, the resulting owner assigned might not be the owner intended.

When mapping owner records, elements that make up the CALC key of each owner record must be mapped for positioning purposes. Specify CALC KEY mapped elements in the INSERT clause to prevent area scans from occurring when you position owner records. Other elements can also be mapped for positioning qualification purposes, if necessary.

The order of owner records in a table mapped for insert purposes is not relevant. However, if an owner record lacks a CALC key and has a usable system index for positioning purposes, that owner should be the first record in the mapping. The use of system indexes is supported in the first record of a mapping only.

Attempts to insert records without establishing required automatic owner positions results in SQLCODE 0x005700F7.

Before you attempt to issue an INSERT statement to insert CA-IDMS records, review the CA-IDMS schema definition of the candidate record to determine the sets for which the record is an automatic member.

Important: You should use tables mapped with a set relationship of `_NONE_` for SQL INSERT only. Querying these tables produces a Cartesian product of every instance of the owner record type with every instance of the member record type. You can use SQL security to prevent accidental use of these tables for purposes other than inserts.

Delete considerations

A delete from an CA-IDMS table results in an @ERASE PERMANENT of the last record type that is mapped in the table. CA-IDMS automatically cascades the ERASE statement to members of mandatory sets and disconnects members of optional sets when CA-IDMS issues this type of ERASE.

You might not be able to delete some records in the schema due to the number of set relationships they participate in and the amount of cascading that is needed to accomplish the delete.

IMS updates

The IMS connector supports updates to IMS data by issuing standard SQL update statements to the data server. Updates that are issued to IMS-mapped tables apply to only the leaf segment that is mapped in the database path.

IMS updates are supported in both the DRA (DBCTL) and DBB/BMP access modes. However, in DBB/BMP mode, you must manage updates. DBB/BMP mode requires update management because:

- Client connections attempting an update enqueue the whole PSB for update purposes.
- The same client connections lock out IMS access for all other users.

Users that are locked out of the PSB receive SQLCODE -9999 and the error message: ALL PCBS are in use. To avoid lock outs, use DRA mode for updating IMS data.

If you plan to update IMS databases in DBB mode, the data server requires these configurations:

- An IMS database log (IEFRDER) that is allocated to disk storage
- The IMS BKO parameter set to Y in the data server JCL

These configurations allow database changes to be backed out in the event of a client rollback request. If a DBB is started without database backout support, attempts to update the database fail with SQLCODE -5701659 and the message: Update not supported by connector.

IMS can change the JOBSTEP name for a data server when the data server runs IMS access with an IMS log file. In this case, operator console commands to the data server must be issued to the JOBSTEP name that is created by IMS. Because the DBB data server allocates the databases, the user ID that is associated with the data server requires CONTROL authority in RACF.

Attention: Do not update an IMS database that accesses the DBD through a secondary index, especially in the case of an inverted hierarchy.

IMS and PSB

To create PSBs for use with data servers, you must complete planning that is similar to PSB definitions for any IMS batch or online applications.

Unlike batch or online programs in which database and segment sensitivity and access options are based on a specific application program, the following factors determine PSB requirements for a data server:

- DBB versus DRA access
- Table mappings (segment sensitivity)
- Join requirements for one or more databases
- Query and update requirements within a data server transaction (PCB PROCOPT)

If you plan to access IMS data by using DBB or BMP access, the access PSB must have enough PCBs to support all users of the data server who access the database at a single point-in-time. In general, the number of PCBs required for each database must minimally equal the maximum number of query processor services that run in the data server at any point in time.

Restriction: An IMS PSB defined with the PSBGEN option LANG=PL/I prevents access to an IMS table. The valid options are LANG=ASSEM or LANG=COBOL.

IMS single transactions and PSBs

If your application needs to issue both query and update requests to IMS databases in a single transaction, all mapped tables must be accessible through a single PSB.

Queries that occur after an update request automatically use the scheduled update PSB to access the IMS data. If a PCB that is needed to satisfy the query request is not found, SQLCODE -9999 is returned with the message: Cannot access a PCB for the database requested.

PCB processing options for IMS data

When you map tables for IMS updates, verify that all PSBs that are defined in the mapping include the correct PCB processing options to insert, update, and delete IMS segments. Failure to do so results in -9999 errors and the message: Unexpected IMS status code received.

VSAM updates

The same database functions are performed to access and modify CICS VSAM and native VSAM data. Like direct access to VSAM data sources, you can select, insert, update, and delete from a CICS VSAM data source.

Classic federation supports the following VSAM file types:

- Entry-sequenced data set (ESDS)
- Key-sequenced data set (KSDS)
- Relative record data set (RRDS)

Support for these file types does not include VSAM linear data sets and relative record data sets in variable-length record format.

Beginning in Version 10.1 PTF rollup 2, Classic federation supports extended-format VSAM data sets. You can define VSAM data sets for all of the supported VSAM files types in basic and extended formats.

Extended-format VSAM data sets can include any combination of the following characteristics:

- Data striping, referred to as a striped data set.
- Data compression, referred to as a DFSMS compressed-format data set.
- Extended addressability, which applies to data sets greater than 4GB in size.

You can select, insert, update, and delete data in a table that references any of the supported VSAM data set types.

Native VSAM provides transactional capabilities using the Data Facility Storage Management Subsystem Transactional VSAM (DFSMSStvs). DFSMSStvs supports the same VSAM file types and supports alternate indexes with path access. VSAM limitations also apply to VSAM files opened using DFSMSStvs.

Native VSAM also supports update access without the transactional capabilities provided by commit and rollback.

Restriction: Because DFSMSStvs and CICS use different sync point managers, a single transaction must not combine both CICS and DFSMSStvs operations. Commits and backouts cannot be coordinated between the two environments.

CICS provides logging, which provides support for transactional capabilities. CICS VSAM files need to be recoverable to participate in logging. With CICS support for transactions, you can perform commit and rollback operations.

Inserts performed against native VSAM and CICS VSAM RRDS files result in appending new records to the end of the data set.

VSAM limitations apply to CICS VSAM:

- Deletes are not supported on ESDS VSAM files due to the flat file format of ESDS.
- Rollbacks are not supported on the ESDS file, because deletes are performed on insert.

CICS also uses alternate indexes that are defined against a VSAM ESDS or KSDS data set. You can issue SELECT statements to the alternate index for both ESDS and KSDS data sets. You can issue UPDATE and DELETE statements to the alternate index when it is defined for a KSDS data set. UPDATE and DELETE statements that reference an alternate index on an ESDS data set are not supported.

Transport protocol for CICS VSAM transactions

Classic federation uses LU6.2 to transmit and return data between the data connector and the provided CICS transaction, EXV1.

Starting CICS transactions

The data server starts a CICS EXV1 transaction and sends the transaction a file name. The EXV1 CICS transaction issues an EXEC CICS INQUIRE FILE. If the file is not opened, it issues an EXEC CICS SET FILE OPEN and reissues the EXEC CICS INQUIRE FILE. Information such as MaxRecLen, KeyLen, KeyOff, and file type is returned.

The data server also starts an EXV1 CICS transaction for each table open. The data server sends an Open command to the EXV1 transaction, which gets information about the file. Depending on the query, commands such as Seek, Search, Read Next, Insert, Update, Delete, Commit, and Rollback are sent to the EXV1 transaction and the EXV1 transaction issues the EXEC CICS native calls.

Monitoring federated queries

You can monitor queries that are run by Classic federation users. This monitoring can be useful when queries are taking a long time to run or using a large amount of system resources.

Before you begin

To monitor queries using this Classic Federation Metrics view, you must be connected to a version 10.1 or greater Classic Federation server with both a query processor and monitoring service running.

About this task

When you update the metrics, they are displayed in two categories and listed by user. The first category of metrics is a summary of all queries for a given user broken out by statement type. The second category of metrics shows information about statements currently being run (equivalent to the DISPLAY,QUERIES MTO command). The metrics that are collected are for all the query processor service classes (QP, QPRR and QPLD).

Procedure

1. To open the view, select **Window > Show View > Classic Federation Metrics**.
2. Collect metrics from the connected servers.
 - a. Select the connected server that you want to poll. If the desired server is not shown, connect to the server from the Console Explorer view.
 - b. Click **Update metrics**. The metrics are updated only when you click **Update metrics** to allow you time to study the results and determine yourself when you want to update the display again with the latest information.

Two-phase commit

Two-phase commit enables you to update multiple, disparate databases within a single transaction, and commit or roll back changes as a single unit-of-work.

As a transaction manager, InfoSphere Classic Federation Server for z/OS prohibits updates to more than one database system in a single unit-of-work. There is no guarantee that changes to all databases will be committed or rolled back together. Without two-phase commit, a successful commit to one database followed by a failed commit (and thus a rollback) to another database would create a partial update, and therefore cause the databases to be out-of-sync.

In general, all database systems that support transactions act as a syncpoint manager for changes made to application data. As a syncpoint manager, these databases maintain a log of changes and can roll back one or more changes to data if the application requests a rollback. In two-phase commit processing, the role of syncpoint manager is moved to an external agent, and each database manager acts as a participant in the two-phase commit process. In two-phase commit processing, applications request commit or rollback processing from the external syncpoint manager instead of the database system itself.

To ensure database integrity, external syncpoint managers must keep a separate log of database activity so that database transactions can be committed or rolled back in the event of a system failure. Because failures can occur at any point during two-phase commit processing, the syncpoint manager must be able to restart at any future point-in-time and know exactly where the two-phase process ended at the time of the failure.

In some cases, the state of a transaction might be classified as *in-doubt* and require a manual decision to determine if the transaction should be committed or rolled back. This is particularly true if the failure occurred somewhere in the middle of phase one processing. In this case, the transaction is in a state where a commit might still be able to proceed, but the application has the option of rolling back.

Recoverable Resource Manager Services (RRS) support

Recoverable Resource Manager Services (RRS) is a syncpoint manager for two-phase commit processing across multiple database management systems. Using RRS, you can ensure that changes to all databases will commit or roll back together.

Classic federation supports the Recoverable Resource Manager Services (RRS) in the z/OS environment. RRS support is implemented for the following databases:

- DB2 for z/OS
- IMS
- CA-Datcom

- VSAM DFSMStvs

As a syncpoint manager, RRS maintains a log of all transactions. Application commit and rollback requests are issued to RRS instead of the native database systems. RRS, in turn, issues commit requests in two phases to all database participants in each transaction.

RRS-enabled query processors

A query processor enabled for RRS functions differently than a standard query processor.

Each query processor task running in a data server processes requests for one or more user connections. When a client or user connects to the server, a specific query processor task is assigned to the connection until the client disconnects from the server. For each client connection, the query processor also maintains a separate transaction. A new transaction is implicitly started whenever the client issues a data access call after either connecting to the server, or ending a previous transaction with a syncpoint (commit or rollback) request.

Tip: Consider the following factors when determining if an RRS-enabled query processor is needed at your site:

- When you use an RRS-enabled query processor, all transactions are sent through RRS, not just updates.
- RRS-enabled query processors are not as fast as non-RRS-enabled query processors.

Configuring query processors for RRS

To enable a query processor for two-phase commit processing in an RRS environment, you need to configure the query processor for RRS.

About this task

In the RRS environment, each transaction is assigned to an RRS context for two-phase commit processing. The query processor creates and maintains an RRS context for each transaction associated with a client connection.

At startup, the CACQPRRS module automatically registers itself with RRS, using the name XDI.RRSWMN.CAC.jjjjjjjj,' where jjjjjjj is the 1-8 character job or started task name (such as XDIPROD).

The RRS query processor dynamically loads the RRS interface modules.

Recommendation: Within a given data server, use RRS-enabled query processors only or use non-RRS-enabled query processors only. Avoid combining both types of query processors within the same data server which might lead to unpredictable results.

Procedure

1. Specify CACQPRRS as the load module on the service definition for the query processor.
2. Ensure that you can access the library that contains the RRS interface modules (usually SYS1.CSSLIB) from either the link-list or from the STEPLIB JCL statement for the data server.

What to do next

For more information about the RRS environment, see the z/OS MVS documentation about programming resource recovery.

Performance impact of RRS-enabled query processors

RRS-enabled query processors can slow performance, because the transactions are single-threaded.

To prevent deadlocks in a query processor task, all transactions are single-threaded when running with RRS. Performance with an RRS-enabled query processor will be slower than with a non-RRS-enabled query processor. The difference in performance depends on the type and volume of transactions. If possible, you should run a query processor without RRS support.

Single-threaded transactions do not prevent additional user connections to the query processor. However, single threaded transactions automatically cause data requests to be queued on the internal pending queue until an in-flight transaction for another client is committed. The performance impact is significant, particularly in applications that mix SQL requests that return medium to large result sets with update transactions.

DB2 for z/OS two-phase commit considerations

DB2 for z/OS uses the RRSAF interface.

When you use an RRS-enabled query processor, you do not use the DB2 Call Attach Facility (CAF), as you do with a standard query processor.

Important: Do not combine DB2 access from an RRS-enabled query processor and a non-RRS-enabled query processor within the same data server.

The CAF provided two pieces of information to the standard query processor that you need to provide to the RRS-enabled query processor:

- The plan name to open when connecting to DB2 for z/OS
- The DB2 thread management exit

For more information about RRSAF, see the DB2 for z/OS documentation about application programming.

Creating the plan name

An RRS-enabled query processor uses the plan name stored in the system catalog.

The plan name is set based on the plan used to import DB2 tables for a particular subsystem. You can create a plan named CACPLAN in DB2. CACPLAN is the default plan name used for pre-existing tables in the system catalog that do not have a plan name.

RRS and system exits

The RRS interface works with the security exit to establish DB2 connections.

The RRS DB2 connector automatically connects with information from CACSX04 if CACSX04 is active in the security environment. The connector uses the authorization ID of the data server job name or the started task name.

IMS two-phase commit considerations

To set up two-phase commit processing for IMS, you need to initialize the IMS RRS environment, define IMS tables for the RRS environment, and define IMS subsystems.

Initializing the RRS environment

The CACRRSI service initializes the IMS RRS environment.

About this task

Only one IMS environment service is allowed when you configure the data server for IMS access. Combining RRS and non-RRS access in a single data server is not supported.

Procedure

- Provide IMS configuration information in the IMS section of the customization parameters file.
 - Default PSB name: Modify `IMSDFPSB=DEFPSB` to specify the name of a PSB. This PSB name is used when an IMS table is referenced by a `CREATE TABLE` statement that contains no PSB name.
 - IMS subsystem ID: Modify `IMSSSID=SSID` to specify the IMS subsystem to access.
- Import the `SCACCONF(CACSVIMO)` member to define the IMS RRS service. Issue the following `MTO IMPORT` command:
`IMPORT,CONFIG,FILENAME=DSN:SCACCONF(CACSVIMO)`
- Define the IMS RRS service before any query processors in the configuration file to ensure that the RRS service is started before any query processor tasks, and stopped after all query processor tasks.
- Ensure that the `AERTDLI` is available from either the link-list or from the `STEPLIB JCL` statement for the data server. Both the IMS data connector and the IMS RRS initialization service dynamically load `AERTDLI`.

Defining tables for the RRS environment

The IMS connector uses the Open Database Access (ODBA) callable interface to IMS. ODBA uses the IMS AIB interface. All tables mapped for the RRS environment must include a `PCBPREFIX` or `PCBNAME` definition for named PCB lookup.

Ensure that all of your IMS mapped tables define a `PCBPREFIX` before using an RRS-enabled query processor. Accessing a table through an RRS-enabled query processor that does not have a `PCBPREFIX` defined results in the error `IMS_ERR_STATUS (0x00570047)` and AIB return and reason codes `0x0104` and `0x0108` which are written to the data server log.

Specifying IMS subsystems for communication with multiple IMS subsystems

The IMS RRS interface supports communication with more than one IMS subsystem. To support this capability, the `CREATE TABLE` and `CREATE INDEX` statements include an optional `SUBSYSTEM` parameter for identifying the name of the subsystem to connect to.

If the DDL used to create the IMS table definition does not contain a `SUBSYSTEM` clause, the value specified for the `SSNPARM` configuration parameter is used when scheduling the PSB and identifying the IMS subsystem to use.

CA-Datcom two-phase commit considerations

Support for two-phase commit using RRS facilities is implemented in CA-Datcom Version 10.0 SP02 (Service Pack 2) and later.

When deciding whether to use RRS for two-phase commit processing and determining how to enable RRS, consider the following factors:

- Existing CA-Datcom facilities fully protect CA-Datcom database-only jobs. Therefore, two-phase commit protection is not needed for these jobs.
- Within a single data server, you must use either an RRS-enabled query processor or a non-RRS-enabled query processor. Combining both types of query processors within the same data server might lead to unpredictable results.

For additional information about the two-phase commit functionality that CA-Datcom supports, see the CA-Datcom documentation about database and system administration.

Enabling CA-Datcom for two-phase commit

The startup parameters supplied at the time the Multi-User Facility (MUF) is started enable CA-Datcom for two-phase commit processing.

Before you begin

User requirements tables must link to CA-Datcom macros in Version 10 or later. Attempting to link a user requirements table to an earlier CA-Datcom Version results in error message 00570115.

Procedure

1. Specify the required parameters and data set definition when you start the MUF. For information about setup requirements, see the CA-Datcom documentation about two-phase commit processing.
2. Assemble and link-edit all user requirements tables used in two-phase commit processing with the appropriate CA-Datcom macro library. The CA-Datcom macros generate a new version identifier and new table definitions.

VSAM DFSMStvs two-phase commit considerations

VSAM DFSMStvs uses RRS as the syncpoint manager for two-phase commit processing for VSAM files accessed through DFSMStvs.

VSAM DFSMStvs access runs in the RRS environment which requires an RRS query processor.

For DFSMStvs access, use of the VSAM service is not valid.

A single transaction should not combine both CICS operations and DFSMStvs operations. DFSMStvs and CICS use different syncpoint managers. As a result, commits and rollbacks cannot be coordinated between the two environments.

Stored procedures

A stored procedure is an application program that performs work that SQL SELECT, INSERT, UPDATE, and DELETE operations cannot perform. A client application invokes a stored procedure application by issuing an SQL CALL statement.

Stored procedures are written in C, COBOL, or Assembler. The main entry point name for the stored procedure module must be the same as its load module name. By default, the stored procedures are expected to be compliant with Language Environment (LE). If the stored procedure does not follow LE conventions, the CREATE PROCEDURE definition for the stored procedure must specify RUN OPTIONS 'NO_LE'.

This section provides an overview of stored procedure processing, the environments in which a stored procedure runs, and the interfaces and support routines that the stored procedure can call to perform specific tasks during execution.

Overview of stored procedure processing

A stored procedure is a form of a remote procedure call that operates in a client-server environment. The application program associated with the stored procedure that is referenced on the CALL statement runs in the address space of the server.

When a CALL statement is issued, the query processor verifies that the following requirements are met:

- The stored procedure identified in the CALL statement exists in the metadata catalog.
- The correct number of parameters and all required parameters are supplied.
- For each parameter for which data was supplied, the data type supplied by the client application is compatible with its defined data type.

The query processor forwards the CALL request to the stored procedure connector for processing. When the stored procedure is defined, you specify the name of a z/OS load module that the stored procedure connector executes when a CALL statement for that stored procedure is received.

The stored procedure connector determines if an executable copy of the stored procedure is already loaded in memory. If not, the connector loads the requested program in the server address space. The stored procedure connector then enters the stored procedure application program, passing a standard parameter list that contains the data values that were sent from the client application. The stored procedure performs any requested processing, and on completion, returns control to the stored procedure connector.

If the stored procedure updates any databases, the application should explicitly issue a commit to apply any changes made by the application before control is returned to the stored procedure connector. Alternately, if the stored procedure detects an error during processing, it should issue a rollback to ensure that any changes that have been made are backed out. If the stored procedure accesses or updates a file (for example, a VSAM file) it must open the file upon entry and ensure that it is closed when control is returned to the stored procedure connector.

When defining the parameters that are passed to the stored procedure application program, you identify the SQL data type for each parameter. You also define if the parameter will be used as an input parameter, an output parameter, or both.

These parameters are passed to the stored procedure in a standard SQLDA format. The stored procedure extracts these parameters from the SQLDA. Based on the parameter values, it performs the processing that the stored procedure application is designed to perform.

While executing, the stored procedure can update the contents of output and input-output parameters that were passed to the application. After control has returned from the stored procedure, the query processor accesses the parameter list passed to the stored procedure application program, extracts the contents of any non-null output and input-output parameters. All of the original input parameters and the original or updated output and input-output parameters are then returned to the client application that issued the CALL statement.

In addition to passing the parameters supplied by the client application, the stored procedure is also passed an SQLCA structure that the stored procedure can update with an error SQLCODE that the query processor will return to the client application. Alternatively, if the stored procedure returns a non-zero return code to the stored procedure connector, the return code will be returned as the SQLCODE of the CALL statement to the client application. The SQLCODE from the SQLCA takes precedence over the stored procedure return code.

Stored procedures can also return an SQL result set in addition to returning input and output parameters. A callable interface supports creation of a result set and the insertion of data rows into a result set. The columns in a result set are defined in the processing logic of the stored procedure and can vary from one client invocation to the next if necessary.

Stored procedure execution environment

A stored procedure runs in the address space of the data server and competes with the data server and other stored procedure applications for resources in the address space.

At the time that the stored procedure runs, the data server has already allocated the memory it needs for the message pool. Any memory that the stored procedure allocates is therefore memory that the data server does not manage. Any allocated memory needs to be freed before the stored procedure returns control to the stored procedure connector.

The load module for the stored procedure also uses memory that the data server does not manage. Because it is likely that you execute multiple copies of the stored procedure simultaneously, these applications must be written as re-entrant and should be link-edited as re-entrant, reusable, and refreshable (RENT,REUS,REFR).

Resident and non-resident stored procedure programs:

Before running the stored procedure, the stored procedure connector determines if a copy of the stored procedure application program needs to be loaded. You can use the STAY RESIDENT parameter to control this processing behavior when you define the stored procedure. The behavior is referred to as *residency*.

You can specify the following values for the STAY RESIDENT parameter.

- STAY RESIDENT = NO: The stored procedure is *non-resident*. Each time a CALL statement is issued, the stored procedure connector loads a copy of the stored procedure and when control is returned an unload is issued.
- STAY RESIDENT = YES: The stored procedure is *resident*.

For resident-stored procedures, the stored procedure connector maintains a list of currently-loaded stored procedure application programs. When a CALL statement is issued, the stored procedure connector checks the list of currently loaded applications. A stored procedure application program that is not on the list is loaded and added to the list before it is called.

Resident stored procedures remain loaded until the query processor is terminated. Each active query processor instance maintains its own list of resident stored procedures. If the stored procedure application program is re-entrant, one copy of the stored procedure application program can be loaded. However, its use count can be greater than one (if multiple query processor instances are executing). In these situations, the stored procedure is not physically unloaded from memory until all query processor instances that issued a load for the stored procedure application program terminate processing.

In addition, the stored procedure connector loads multiple copies of the stored procedure if the stored procedure (for example, the load module name) is associated with multiple stored procedure definitions. Currently, there is no method to determine how many stored procedure applications programs are loaded within the server or how many have been loaded by a particular query processor or stored procedure connector.

When you develop a stored procedure you need to define the stored procedure as non-resident. After testing is completed, change the stored procedure to be resident for performance purposes.

Recommendation: While testing a stored procedure application program defined as resident, if you run the stored procedure once and then modify it, you need to shut down the query processor instance to re-test it. By stopping the query processor instance that ran the stored procedure application program the first time, you can load the updated copy of the stored procedure application program. Because this situation occurs frequently, display a version identifier during initial development so that you can easily ascertain which version of your stored procedure the stored procedure connector is running.

LE execution environment:

Generally, a stored procedure is developed using a high-level language, such as COBOL. The IBM-supplied high-level languages use the z/OS Language Environment (LE). Language Environment provides common memory allocation, error reporting, and other services the high-level languages or Assembler Language programs can use.

Stored procedures that are LE-compliant run within the LE environment of the Classic data server.

Important: You cannot use LE-compliant stored procedures with the NO_LE option, otherwise abends can occur. The NO_LE option was supported in previous releases of Classic federation. This option is not supported in Version 10.1 or higher. If you had LE-compliant stored procedure definitions with the NO_LE option in a previous release, you must manually redefine those stored procedures without specifying the NO_LE option.

Overview of CICS interface for stored procedures

With the CICS interface, stored procedures can open CICS conversations, execute a CICS application program, and update VSAM files.

If you need to create a stored procedure that updates a VSAM file, you might find that CICS has exclusive control of the file and that it cannot be updated from the data server address space. For these situations, and potentially others, facilities are provided that enable you to invoke a CICS application program.

The client application that issued the CALL statement passes a copy of the data to the CICS application program. Like a stored procedure application running within the address space of the data server, the CICS application program can update the values for output and input-output parameters for return to the client application.

Unlike stored procedures running in the address space of the data server, the CICS application program does not have accessibility to an SQLCA structure for error reporting. Instead, it has addressability to an application return code that is returned to the stored procedure application program running in the data server address space and is automatically returned to the client application. If a CICS abend is detected it is sent back to the stored procedure, and by default, to the client application.

Overview of IMS interface for stored procedures

With the IMS interface, stored procedure programs can access IMS data, schedule and unschedule PSBs, and issue a series of standard DL/I calls.

The CACTDRA interface module allows a stored procedure to access IMS data locally (within the data server address space) using the DRA interface. The CACTDRA interface allows a stored procedure application program to schedule a PSB, issue a series of standard DL/I calls, and then unschedule the PSB. The CACTDRA interface allows access and updates to full function IMS databases such as HDAM or HIDAM, and Fast Path DEDB databases.

Using the CACTDRA interface allows the stored procedure to safely access and update IMS data and supports transaction isolation from other stored procedures that might also be accessing IMS data. Using the CACTDRA interface, the stored procedure can perform in a similar way to an IMS/DC online transaction program.

Overview of CA-Datcom interface for stored procedures

With the CA-Datcom interface, stored procedure programs can use native CA-Datcom commands and control blocks to access CA-Datcom data within the data server address space.

If your stored procedure needs to access or update CA-Datcom data, you can use the CACTDCOM interface load module. The CACTDCOM interface module allows a stored procedure to access CA-Datcom data within the server address space. Using the CACTDCOM interface allows the stored procedure to safely access and update CA-Datcom data, and supports transaction isolation from other stored procedure application programs that might also be accessing CA-Datcom data.

The CACTDCOM interface requires a stored procedure to open a User Requirements Table (URT). After the URT is open, any series of CA-Datcom commands can be issued. When the stored procedure application program completes processing, the URT must be closed.

Support routines

In addition to the CICS, CA-Datcom, and IMS interfaces, your stored procedure application program can call the support routines CACSPGRO, CACSPGPW, and CACSPGUI to copy parameters and values to the application storage area. You can retrieve RUN OPTIONS values and the user ID and password of the logged-on user.

The following table identifies the name and the purpose of each subroutine.

Table 30. Support routines

Routine name	Purpose
CACSPGRO	Copies the value of the RUN OPTIONS parameter into an application storage area.
CACSPGPW	Copies the value of the user password into an application storage area. The user password was captured when the client application connected to the server.
CACSPGUI	Copies the value of the user ID into an application storage area. The user ID was captured when the client application connected to the server.

Unlike the CICS, CA-Datacom, and IMS interfaces that are distributed as load modules, these support routines are supplied in object-module form for direct inclusion in your stored procedure. These routines are written in Assembler Language and use standard OS linkage conventions. All routines are passed two parameters. The first parameter must be the address of the SQLDA parameter passed to the stored procedure. The second parameter is an address where the value is copied.

Stored procedure samples

You can use sample stored procedures to help you develop your own process.

The stored procedure samples are located in the SCACSAMP library. Most of the samples are written in COBOL. For each sample, the following table identifies the member name and provides a short description.

Table 31. Stored procedure samples

Member name	Description
CACSPCCC	Sample compile and link deck for stored procedure applications calling CACSPBR.
CACSPCCD	Sample compile and link deck for the sample stored procedure using the CACTDCOM interface.
CACSPCCL	Sample compile and link deck for sample local stored procedure.
CACSPCCR	Sample compile and link deck for the sample CICS stored procedure.
CACSPCOM	Generic stored procedure to invoke a CICS application program.
CACSPCP	Sample stored procedure definitions containing a parameter definition for each supported data type.
CACSPCPY	COBOL definitions for the argument data passed to the stored procedure. This should be included in your stored procedure application. Sample local stored procedure application using the IMS DRA interface.
CACSPDC1	Sample local stored procedure using the interface module CACTDCOM to access CA-Datacom. Statically link module CACTDCOM with CACSPDC1.
CACSPCLL	Sample local stored procedure application.
CACSPREM	Sample remote stored procedure application executing in CICS.
CACSPDFH	COBOL version of the CICS communications area passed to a CICS application invoked by CACSP62.
CACSPGRO	Get RUN OPTIONS support routine object module.
CACSPGPW	Get Password support routine object module.
CACSPGUI	Get User ID support routine object module.
CACSPSCA	COBOL SQLCA structure for inclusion in your stored procedure application program.
CACSPSDA	COBOL SQLDA structure for inclusion in your stored procedure application program and/or CICS application program.
CACSPVTM	COBOL APPC function and data structures for interfacing with CACSPBR.

Table 31. Stored procedure samples (continued)

Member name	Description
CACSPRSS	Sample COBOL stored procedure that creates a result set.

Defining stored procedures

You can use the Classic Data Architect to define stored procedures. You specify the stored procedure definition by using the CREATE PROCEDURE statement.

General stored procedure information is stored in the SYSIBM.SYSROUTINES system table. The parameters that are supplied by the client application and passed to the stored procedure are stored in the SYSIBM.SYSPARMS system table.

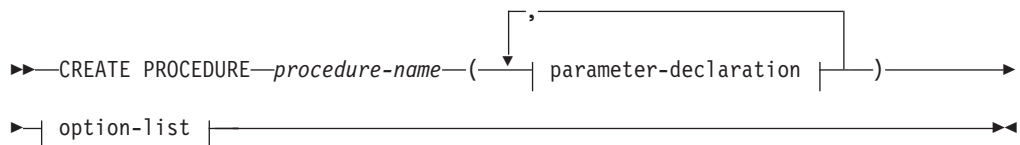
When the stored procedure is run in an LE environment, you can use the RUN OPTIONS parameter on the CREATE PROCEDURE statement to supply custom LE runtime options during environment initialization. In addition, extended use of the RUN OPTIONS parameter enables you to deactivate the LE environment for a particular stored procedure. The RUN OPTIONS parameter also allows you to specify CICS transaction scheduling information for CICS interfacing or CA-Datacom resource information for CA-Datacom interfacing.

CREATE PROCEDURE

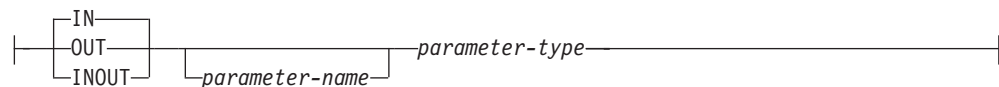
Using the Classic Data Architect, you can create a stored procedure definition with the CREATE PROCEDURE statement.

After the stored procedure is defined, you can generate and run the DDL for creating the stored procedure in a metadata catalog.

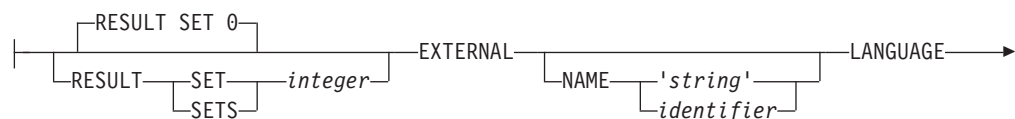
CREATE PROCEDURE

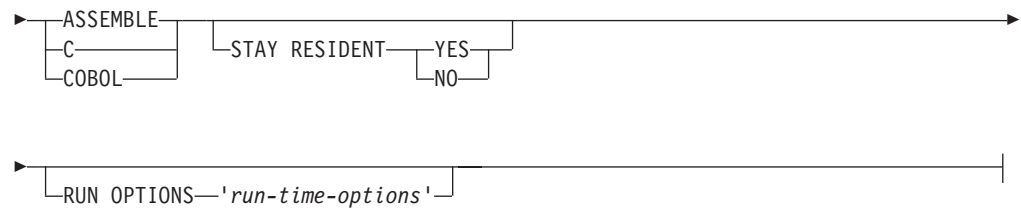


parameter-declaration:



option-list:





The table that follows describes each of the parameters you can specify using the CREATE PROCEDURE statement.

Table 32. CREATE PROCEDURE parameters and descriptions

Parameter	Description
CREATE PROCEDURE <i>procedure-name</i>	A required keyword phrase that defines a stored procedure. <i>procedure-name</i> names the stored procedure. The name is implicitly or explicitly qualified by an owner. The name, including the implicit or explicit qualifier, must not identify an existing stored procedure at the current server. To explicitly specify an owner, use the syntax <i>owner-name.procedure-name</i> . The <i>owner-name</i> can be 1-to-8 characters long and the <i>procedure-name</i> can be 1-to-18 characters long. If an <i>owner-name</i> is not specified, the implicit <i>owner-name</i> is the TSO user ID of the person that runs the metadata utility to define the stored procedure.
parameter-declaration	Specifies the parameters that are passed to the stored procedure application program and that must be supplied by the client application when the stored procedure is invoked. At least one parameter must be defined for a stored procedure. There is no fixed upper limit on the number of parameters that can be defined for a stored procedure. The maximum number of parameters that can be defined is dependent on the size of the resultant SQLDA. The maximum data size for all data and indicator variables is 32767-bytes. You can use a stored procedure parameter for input, output or both input and output. The options are: <ul style="list-style-type: none"> • IN: The default. It identifies the parameter as an input parameter to the stored procedure. The parameter contains its original value when the stored procedure returns control to the client application. • OUT: Identifies the parameter as an output parameter. The stored procedure application program returns a value to the client application or a null-indicator that indicates no value is being returned. • INOUT: Identifies the parameter as both an input and an output parameter. The client application must supply a value for an INOUT parameter, because upon return the stored procedure might have changed this value.
parameter-name	The <i>parameter-name</i> specifies the name of the parameter in the parameter-declaration. The name can be up to 30 characters long, must be unique within the stored procedure definition and cannot be IN, OUT, or INOUT. Parameter names are optional, but specifying a <i>parameter-name</i> is highly recommended. When initially testing your stored procedure your client application might receive various SQL codes indicating that an incompatible data type was passed, a NULL parameter is not allowed, or other similar codes. To resolve these problems you can activate tracing in the data server. When activated, a log message is generated for the parameter in error and the information identifies the <i>parameter-name</i> , its data type, and length. Naming your parameters, therefore, makes problem resolution easier in these situations.
parameter-type	Required keyword that identifies the SQL data type of the parameter.
option-list	List of options to be used for the stored procedure.

Table 32. CREATE PROCEDURE parameters and descriptions (continued)

Parameter	Description
RESULT SET(S) <i>integer</i>	The value of <i>integer</i> represents the number of result sets that can be returned by the stored procedure. Currently, this value can be set to 0 or 1.
EXTERNAL NAME ' <i>string</i> ' or <i>identifier</i>	Specifies the z/OS load module of the application program that the Server should load to satisfy a call for the stored procedure. If you do not specify the NAME clause, NAME procedure-name is implicit. The procedure-name is limited to 8 characters. When an explicit name is specified it can be from 1-to-8 characters long and can be supplied either as a quoted string or as an identifier. A quoted string is required if the application program name matches any of the keywords supplied on the CREATE PROCEDURE statement.
LANGUAGE	Required parameter that identifies the programming language in which the stored procedure was written. All programs should be designed to run in IBM's Language Environment (LE). Valid values are: <ul style="list-style-type: none"> • ASSEMBLE: Assembler Language • COBOL: IBM COBOL • C: IBM C Language
STAY RESIDENT	Identifies whether the stored procedure connector should unload the stored procedure after it has been executed. Options are: <ul style="list-style-type: none"> • NO: Unload the program after each execution. When initially testing your stored procedure you should use this option. This allows you to modify your stored procedure and re-test it without having to shutdown the query processor that received the client application request to execute your stored procedure. • YES: Do not unload the stored procedure after it has completed execution. For performance purposes, YES should be specified after your stored procedure has been tested.
RUN OPTIONS <i>run-time-options</i>	Specifies the Language Environment run-time options to be used for the stored procedure. You must specify run-time-options as a character string no longer than 254 bytes. If you do not specify RUN OPTIONS or pass an empty string, the stored procedure connector passes a value of ALL31(OFF). Additionally, the product has extended the use of the RUN OPTIONS parameter to disable executing a stored procedure in an LE environment and to supply CICS transaction scheduling information or CA-Datcom resource information. RUN OPTIONS information must be supplied on a single input line. To specify a RUN OPTIONS string that exceeds 80 characters, use a variable length file.

A stored procedure parameter can be defined as one of the types listed in the following table.

Table 33. Stored procedure supported data types

Supported data types	Description (<i>n</i> is always a decimal integer)
INTEGER	Fullword signed hexadecimal, 32-bits, no decimal point.
SMALLINT	Halfword signed hexadecimal, 16-bits, no decimal point.
DECIMAL(<i>p</i> [, <i>s</i>])	Packed decimal $1 \leq p \leq 31$ and $0 \leq s < p$ where: <ul style="list-style-type: none"> • <i>p</i> is the precision (total number of digits) and • <i>s</i> is the total number of digits to the right of the decimal point.

Table 33. Stored procedure supported data types (continued)

Supported data types	Description (<i>n</i> is always a decimal integer)
FLOAT	4-byte single precision floating point number.
DOUBLE	8-byte double precision floating point number.
CHAR (<i>n</i>)	Fixed-length character string of length <i>n</i> where $1 \leq n \leq 254$.
VARCHAR (<i>n</i>)	Variable-length character string where $1 \leq n \leq 32704$.
GRAPHIC (<i>n</i>)	Fixed-length, double-byte character set (DBCS) string where $1 \leq n \leq 127$. The value of <i>n</i> specifies the number of DBCS characters. For example, GRAPHIC(10) specifies a parameter that occupies 20 bytes of storage.
VARGRAPHIC (<i>n</i>)	Variable-length, DBCS string where $1 \leq n \leq 16351$. The value of <i>n</i> specifies the number of DBCS characters. For example, VARGRAPHIC(10) specifies a parameter that occupies 20 bytes of storage.

DROP PROCEDURE

Use the DROP PROCEDURE statement to remove an existing stored procedure from a metadata catalog. The DROP PROCEDURE statement is also required when you replace an existing stored procedure with a stored procedure of the same name.

DROP PROCEDURE

►►—DROP PROCEDURE—*procedure-name*—◄◄

The following table describes the supplied parameters of the DROP PROCEDURE statement.

Table 34. DROP PROCEDURE parameters and descriptions

Parameter	Description
DROP PROCEDURE	Statement that removes an existing stored procedure from the metadata catalogs.
<i>procedure-name</i>	Identifies the stored procedure to be dropped. The name must identify a stored procedure that has been defined with the CREATE PROCEDURE statement at the current server. When a procedure is dropped, all privileges on the procedure are also dropped.

Deactivating the Language Environment

If your stored procedure cannot run in a Language Environment (LE), deactivate it.

For example, if a stored procedure is written in Assembler, and not using LE services, specify NO_LE on the RUN OPTIONS parameter.

Important: The following rules apply to stored procedures and the Language Environment:

- If your stored procedure runs in a Language Environment, do not specify the keyword NO_LE. Use of the keyword NO_LE for an LE-compliant stored procedure can result in an abend.
- You can use stored procedures that are not LE-compliant only if the stored procedure entry point can be directly accessed from an LE program. For stored

procedures that are not LE-compliant, you must specify the NO_LE option on the RUN OPTIONS parameter, otherwise abends can occur.

Writing stored procedures

When you create your own stored procedures, consider programming language, passed parameters, and SQL data types. Passed parameters enable you to specify SQLCA structures and SQLDA structures, including SQLVAR structures associated with the SQLDA.

A stored procedure is invoked using standard Assembler Language linkage conventions. You can write stored procedures in C, COBOL or Assembler Language. The stored procedure is always passed two parameters: an SQLDA structure and an SQLCA structure. The following figure shows how these parameters are passed. All parameters and the data and indicator values contained in the SQLDA are 31-bit addresses.

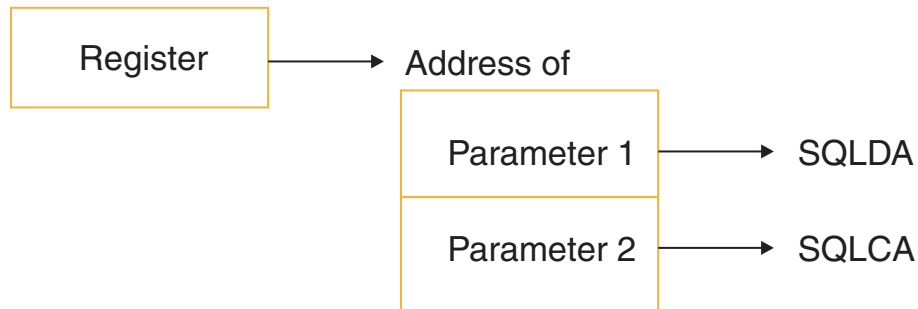


Figure 7. Parameters passed to the stored procedure

Upon return from the stored procedure, Register 15 is assumed to contain a return code value. If a non-zero value is found in Register 15, it will be returned to the client application unless there is a non-zero value in the SQLCODE field in the SQLCA.

Required: Your stored procedure must be coded as a subroutine and must not issue a function call that causes the run-time environment to be terminated. For example, a COBOL stored procedure must not issue STOP RUN and must instead return control using the GOBACK statement. If you terminate the run-time environment (for example, using STOP RUN in COBOL), this causes the query processor task to also be terminated.

The SQLDA structure consists of a 16-byte header followed by a variably-occurring array of SQLVAR structures. Each SQLVAR represents a parameter-declaration from the CREATE PROCEDURE statement. The SQLVARs are passed in the sequence defined on the CREATE PROCEDURE statement. Each SQLVAR structure is 44-bytes long.

The sample member CACSPSDA is a COBOL copybook that shows the structure and contents of the SQLDA structure. The following figure shows the contents of the SQLDA header.

The SQLDA structure is as follows:

```
01 ARG-DATA.
   05 ARG-SQLDAID          PIC X(8).
   05 ARG-SQLDABC          PIC 9(8) COMP.
   05 ARG-SQLN             PIC 9(4) COMP.
```

```

05 ARG-SQLD PIC 9(4) COMP.
05 ARG-SQLVAR OCCURS 1 TO 2000 TIMES
    DEPENDING ON ARG-SQLN.
10 ARG-SQLTYPE PIC 9(4) COMP.
88 ARG-SQL-VARCHAR VALUE 449.
88 ARG-SQL-CHAR VALUE 453.
88 ARG-SQL-VARGRAPHIC VALUE 465.
88 ARG-SQL-GRAPHIC VALUE 469.
88 ARG-SQL-FLOAT VALUE 481.
88 ARG-SQL-DECIMAL VALUE 485.
88 ARG-SQL-INTEGER VALUE 497.
88 ARG-SQL-SMALLINT VALUE 501.

10 ARG-SQLLEN PIC 9(4) COMP.
10 ARG-SQLDATA POINTER.
10 ARG-SQLIND POINTER.
10 ARG-SQLNAME.
20 ARG-NAME-LEN PIC 9(4) COMP.
20 ARG-NAME-LABEL PIC X(30).

```

The following table describes the format and contents of the SQLDA header.

Table 35. SQLDA header contents

COBOL name	SQL data type	Description
ARG-SQLDAID	CHAR(8)	Signature that identifies the structure as an SQLDA. Always contains the value "SQLDA" followed by three spaces.
ARG-SQLDABC	INTEGER	Identifies the length of the SQLDA structure and is computed as 16 + (SQLN * 44).
ARG-SQLN	SMALLINT	Identifies the number of SQLVAR entries contained in the SQLDA.
ARG-SQLD	SMALLINT	Identifies the number of SQLVAR entries contained in the SQLDA. Same as ARG-SQLN.

The following table describes the format and contents of the SQLVAR structure.

Table 36. SQLVAR contents

COBOL name	SQL data type	Description
ARG-SQLTYPE	SMALLINT	Identifies the type of data that is referenced by field ARG-SQLDATA.
ARG-SQLLEN	SMALLINT	Identifies the length of the data that is referenced by field ARG-SQLDATA.
ARG-SQLDATA	INTEGER	Pointer to the argument data. Before using this pointer value you must check the 2-byte data value referenced by ARG-SQLIND to determine whether the argument data is null (for example, a value was not supplied by the client application). If the argument data is null, the data referenced is low-values and should not be referenced. If the data is not null, the size of the data referenced is identified by ARG-SQLLEN.
ARG-SQLIND	INTEGER	Pointer to a 2-byte (half-word - SMALLINT) indicator field that identifies whether ARG-SQLDATA is null or not. If the indicator field contains zeros, the data referenced by ARG-SQLDATA is not null and contains valid data matching the SQL data type identified in ARG-SQLTYPE. If the indicator field contains -1 (x'ffff') then the data referenced by ARG-SQLDATA is binary zeros and should not be referenced.

Table 36. SQLVAR contents (continued)

COBOL name	SQL data type	Description
ARG-SQLNAME	VARCHAR(30)	The <i>parameter-name</i> specified in the CREATE PROCEDURE statement for this parameter. If no <i>parameter-name</i> was specified, ARG-NAME-LEN is zeros and ARG-NAME-LABEL is low-values. If a <i>parameter-name</i> was specified, ARG-NAME-LEN identifies how long <i>parameter-name</i> is and ARG-NAME-LABEL contains the <i>parameter-name</i> left justified and padded with blanks.

The following table describes the SQL data types that can be passed to the stored procedure and the length of the data referenced by ARG-SQLDATA for each data type.

Table 37. SQL data type descriptions

COBOL name	Value	Value in ARG-SQLLEN and corresponding length of data referenced by ARG-SQLDATA
ARG-SQL-VARCHAR	449	Identifies that ARG-SQLDATA references a variable length character field. ARG-SQLLEN identifies the maximum length of the variable length character field, excluding the 2-byte length field. The actual size of the data is identified in the 2-byte length field.
ARG-SQL-CHAR	453	Identifies that ARG-SQLDATA references a fixed length character field. ARG-SQLLEN identifies the length of the character field.
ARG-SQL-VARGRAPHIC	465	Identifies that ARG-SQLDATA references a variable length graphic field. ARG-SQLLEN identifies the maximum length (in DBCS characters) that the variable length graphic field can be, excluding the 2-byte length field. The actual size of the data (in DBCS characters) is identified in the 2-byte length field.
ARG-SQL-GRAPHIC	469	Identifies that ARG-SQLDATA references a fixed length graphic field. ARG-SQLLEN identifies the length of the graphic field in DBCS characters.
ARG-SQL-FLOAT	481	Identifies that ARG-SQLDATA references a floating point number. The field is a single precision floating point number that is 4-bytes long if FLOAT was specified on the parameter-declaration on the CREATE PROCEDURE statement. If DOUBLE was specified, then the field is a double-precision floating point number that is 8-bytes long.
ARG-SQL-DECIMAL	485	Identifies that ARG-SQL-DATA references a signed packed decimal field. The first byte of ARG-SQL-LEN identifies the scale (number of digits) in the decimal field. The second byte of ARG-SQL-LEN identifies the precision (implied decimal point) of the decimal data. If the scale is an even number, the physical length of the decimal data is scale / 2. If the scale is an odd number, the physical length of the data is (scale +1)/2.
ARG-SQL-INTEGER	497	Identifies that ARG-SQL-DATA references a signed full-word field that is 4-bytes long.
ARG-SQL-SMALLINT	501	Identifies that ARG-SQL-DATA references a signed half-word field that is 2-bytes long.

The SQLCA structure is much simpler than the SQLDA. Sample member CACSPSCA is a COBOL copybook that shows the structure and contents of the SQLCA structure. The following sample code shows the contents of this copy book.

The SQLCA structure is as follows:

```

01 ARG-SQLCA
05 ARG-SQLCAID                PIC X(8).
05 ARG-SQLCABC                PIC 9(8) COMP.
05 ARG-SQLCODE                PIC 9(8) COMP.

```

Table 38. SQLCA contents

COBOL name	SQL data type	
ARG-SQLCAID	CHAR(8)	Signature that identifies the structure as an SQLCA. Always contains the value "SQLCA" followed by three spaces.
ARG-SQLCABC	INTEGER	Identifies the length of the SQLCA structure.
ARG-SQLCODE	INTEGER	Full-word 32-bit signed integer that is returned to the client application when SQLCODE contains a non-zero value.

The ARG-SQLCABC field reports a length that is longer than the structure's contents identified above. A full DB2-style SQLCA is passed to your stored procedure. These additional fields are not being documented, as they are not inspected by the stored procedure connector or the query processor and are not returned to the client application.

Sample member CACSPCP provides a sample stored procedure definition that contains parameter definitions for the different SQL data types that can be passed to a stored procedure. Its contents are shown in the following sample DROP and CREATE PROCEDURE definition:

```

DROP PROCEDURE CAC.DATA_TYPES;
CREATE PROCEDURE CAC.DATA_TYPES
(IN DT_SMALLINT SMALLINT,
 IN DT_INTEGER INTEGER,
 IN DT_CHAR CHAR(10),
 IN DT_VARCHAR VARCHAR(20),
 IN DT_FLOAT FLOAT,
 IN DT_DOUBLE DOUBLE,
 IN DT_DECIMAL DECIMAL(9,2),
 IN DT_GRAPHIC GRAPHIC(10),
 IN DT_VARGRAPHIC VARGRAPHIC(20))
EXTERNAL NAME CACSPDAT
LANGUAGE COBOL
STAY RESIDENT NO;

```

This sample shows that the stored procedure name is CACSPDAT and it is written in COBOL. For a COBOL program, the field definitions for the actual data values and their associated indicator variables must be defined in the LINKAGE SECTION.

While the COBOL field names do not have to match the parameter names specified on the stored procedure CREATE PROCEDURE definition statement, it is a good practice. Establish a standard for naming null indicator variables.

In COBOL, addressability to the data associated with a parameter and its null indicator field is established by using the SET ADDRESS OF statement.

You should also establish addressability to a parameter's null indicator and test to see whether the value is not null before establishing addressability to its data value. Likewise, after addressability is established before referencing a data value, check its associated null indicator before attempting to manipulate a data value. This practice should be done even if you know that a value does not contain a null value. This practice prevents abends at a later date, if another team performs maintenance on the client application and does not follow the rules.

The following section contains sample COBOL data and indicator definitions:

```
LINKAGE SECTION.  
COPY CACSPSDA.  
COPY CACSPSCA.
```

```
01 DT-SMALLINT          PIC S9(4) COMP.  
01 DT-=SMALLINT-IND    PIC S9(4) COMP.  
  
01 DT-INTEGGER         PIC S9(9) COMP.  
01 DT-INTEGGER-IND    PIC S9(4) COMP.  
  
01 DT-CHAR             PIC X(10).  
01 DT-CHAR-IND        PIC S9(4) COMP.  
  
01 DT-VARCHAR.  
  05 DT-VARCHAR-LEN    PIC 9(4) COMP.  
  05 DT-VARCHAR-DATA   PIC X(1)  
                      OCCURS 20 TIMES  
                      DEPENDING ON DT-VARCHAR-LEN.  
01 DT-VARCHAR-IND     PIC S9(4) COMP.  
  
01 DT-FLOAT            COMP-1.  
01 DT-FLOAT-IND       PIC S9(4) COMP.  
  
01 DT-DOUBLE           COMP-2.  
01 DT-DOUBLE-IND      PIC S9(4) COMP.  
  
01 DT-DECIMAL          PIC S9(7)V99 COMP-3  
01 DT-DECIMAL-IND     PIC S9(4) COMP.  
  
01 DT-GRAPHIC          PIC G(10)  
                      USAGE DISPLAY-1.  
01 DT-GRAPHIC-IND     PIC S9(4) COMP.  
01 DT-VARGRAPHIC.  
  05 DT-VARGRAPHIC-LEN PIC 9(4) COMP.  
  05 DT-VARGRAPHIC-DATA PIC G(1)  
                      USAGE DISPLAY-1  
                      OCCURS 20 TIMES  
                      DEPENDING ON DT-VARGRAPHIC-LEN.  
01 DT-VARGRAPHIC-IND PIC S9(4) COMP.
```

Establishing addressability

The following shows how to establish addressability:

PROCEDURE DIVISION USING ARG-DATA, ARG-SQLCA.

```
SET ADDRESS OF DT-SMALLINT-IND TO ARG-SQLIND(1).  
IF DT-SMALLINT-IND = ZEROS  
  SET ADDRESS OF DT-SMALLINT TO ARG-SQLDATA(1).  
  
SET ADDRESS OF DT-INTEGGER-IND TO ARG-SQLIND(2).  
IF DT-INTEGGER-IND = ZEROS  
  SET ADDRESS OF DT-INTEGGER TO ARG-SQLDATA(2).  
  
SET ADDRESS OF DT-CHAR-IND TO ARG-SQLIND(3).  
IF DT-CHAR-IND = ZEROS  
  SET ADDRESS OF DT-CHAR TO ARG-SQLDATA(3).  
  
SET ADDRESS OF DT-VARCHAR-IND TO ARG-SQLIND(4).  
IF DT-VARCHAR-IND = ZEROS  
  SET ADDRESS OF DT-VARCHAR TO ARG-SQLDATA(4).  
  
SET ADDRESS OF DT-FLOAT-IND TO ARG-SQLIND(5).  
IF DT-FLOAT-IND = ZEROS
```

```

        SET ADDRESS OF DT-FLOAT TO ARG-SQLDATA(5).

SET ADDRESS OF DT-DOUBLE-IND TO ARG-SQLIND(6).
IF DT-DOUBLE-IND = ZEROS
    SET ADDRESS OF DT-DOUBLE TO ARG-SQLDATA(6).

SET ADDRESS OF DT-DECIMAL-IND TO ARG-SQLIND(7).
IF DT-DECIMAL-IND = ZEROS
    SET ADDRESS OF DT-DECIMAL TO ARG-SQLDATA(7).

SET ADDRESS OF DT-GRAPHIC-IND TO ARG-SQLIND(8).
IF DT-GRAPHIC-IND = ZEROS
    SET ADDRESS OF DT-GRAPHIC TO ARG-SQLDATA(8).

SET ADDRESS OF DT-VARGRAPHIC-IND TO ARG-SQLIND(9).
IF DT-VARGRAPHIC-IND = ZEROS
    SET ADDRESS OF DT-VARGRAPHIC TO ARG-SQLDATA(9).

```

Your stored procedure must not modify the contents of the SQLDA structure. Your program can modify the contents of the data referenced by ARG-SQLDATA and ARG-SQLIND fields. Data that is modified for output or input-output fields is returned to the client application.

When your application modifies an SQLIND indicator field to indicate that the corresponding data is null, then any modifications to the corresponding data field are not returned to the client application. Additionally, your application should not modify an SQLIND indicator field that identifies the corresponding data field as being null to indicate that the data field (upon return from your stored procedure) now contains valid data. Such a modification is likely to cause problems in the client application because the client application is not expecting to receive data for a parameter that it knows is null.

Your stored procedure can call other stored procedure application programs or one or more of your existing application programs. However, if you call existing applications ensure that these are written as subroutines.

If you are not using the supplied CICS, CA-Datcom, IMS, or result set creation interfaces, compile and link your stored procedure application program using the standard procedures for the language that the application program is written in. If you are using the CICS, CA-Datcom, IMS, and/or result set creation interfaces, the procedures are slightly modified so that the interface load modules are included in the link step.

After you have linked an executable copy of your stored procedure, either copy it into one of the load libraries referenced in the data server JCL or update the data server JCL to include the library that your application resides in on the STEPLIB DD statement. Also, add any additional DD statements to the Server JCL that your stored procedure references.

Start the data server after you have performed these modifications. You are now ready to start testing your stored procedure. You will need to do this from a client application. Descriptions of the techniques that you can use to invoke a stored procedure are provided in related topics.

Invoking stored procedures

The clients invoke your stored procedure application program. All clients support the SQL CALL statement.

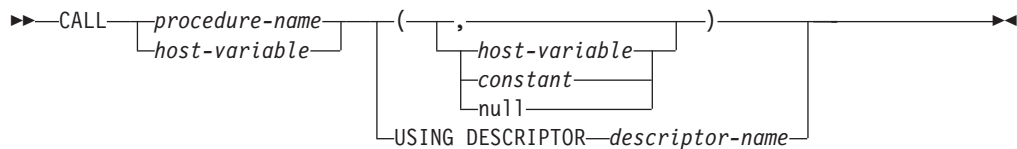
This section describes the CALL statement syntax that you use to invoke a stored procedure, how to obtain metadata about stored procedures, and the connector-specific APIs that you can use to issue the CALL statement.

CALL statement

This topic provides information about the syntax and usage of the CALL statement.

Important: Client applications using ODBC or JDBC clients cannot use the USING DESCRIPTOR form of this statement and cannot use indicator host-variables. They use parameter markers instead.

CALL statement



The following table describes the CALL statement parameters.

Table 39. CALL statement parameters

Parameter	Description
<i>procedure-name</i> or <i>host-variable</i>	Identifies the name of the stored procedure to be executed. The name should generally be specified in the form <i>owner.name</i> . If an owner is not specified, the connected user's user ID is used for the owner. If a user ID is not supplied on connect the owner is PUBLIC. You can either specify the name of the stored procedure explicitly as an identifier <i>procedure-name</i> , or you can use a host-variable reference, where <i>host-variable</i> contains the name of the stored procedure to be executed.
<i>host-variable</i>	The name of a <i>host-variable</i> that, for input or input-output parameters contains the data that is passed to the stored procedure application program. For output parameters, the <i>host-variable</i> identifies a storage location that will be updated with the value supplied by the stored procedure application program upon return from the CALL statement. The host variables data type must be compatible with the data type defined for the parameter. If a null indicator variable is specified it can only contain a -1 for an output parameter.
constant	A numeric or string constant that is passed to the stored procedure application program. Constants can only be specified for input parameters. Additionally, the constant must match the data-type for the parameter. That is, a numeric constant of the appropriate scale and precision must be supplied for INTEGER, SMALLINT, DECIMAL, FLOAT and DOUBLE parameter types. For CHAR or VARCHAR parameters, the constant must be in single quotes. Constants cannot be specified for GRAPHIC or VARGRAPHIC data types.
NULL	Identifies that no value is being supplied for the parameter. NULL can only be specified for input parameters. This causes the parameter's associated null indicator to be set to -1 when the stored procedure application program is called.

Table 39. CALL statement parameters (continued)

Parameter	Description
USING DESCRIPTOR <i>descriptor-name</i>	Identifies that the parameters to be passed to the stored procedure application program are contained in the SQLDA structure referenced by <i>descriptor-name</i> . The correct number and types of parameters must be contained in the SQLDA. Additionally, SQLN must identify the number of parameters being passed and SQLDABC must contain the correct value - (SQLN * 44) + 16.

ODBC stored procedure support

The ODBC client implements the standard ODBC API interfaces used to invoke and obtain metadata information about the stored procedures defined for the data source that the ODBC client is connected to.

Some differences exist between the implementation of the ODBC client and the ODBC standard.

Using the SQLProcedures call, you can obtain the list of stored procedures that are defined for a data source in the metadata catalogs. To retrieve parameter information about one or more stored procedure definitions, you use the SQLColumnProcedures call. The ODBC client implementation supports the following syntax and options for these calls:

- Qualifier names are not supported. szProcQualifier should always be null and cbProcQualifier should always be zero.
- When retrieving the result sets from these calls, TABLE_QUALIFIER is always null.
- The REMARKS field is always spaces.
- For SQLProcedure calls, PROCEDURE_TYPE is always SQL_PT_UNKNOWN.
- When retrieving the result set from the SQLProcedureColumns call, the COLUMN_TYPE will only be one of the following values:
 - SQL_PARAM_INPUT
 - SQL_PARAM_INPUT_OUTPUT
 - SQL_PARAM_OUTPUT
- RADIX is always 10.
- NULLABLE is always set to SQL_NULLABLE_UNKNOWN.

To invoke a stored procedure use the SQLExecDirect call. The ODBC client supports the following two formats of the CALL statement that can be supplied on the SQLExecDirect call:

- The shorthand syntax: "{[?]=}call *procedure-name*(*parameter*,[*parameter*],...)"
- "CALL *procedure-name*(*parameter*,[*parameter*],...)"

Before issuing the SQLExecDirect call you might have to issue one or more SQLBindParameter calls for any parameter markers that are contained in the CALL statement.

The following rules apply to parameter markers:

- For output or input-output parameters, *parameter* must be a parameter marker.
- For input parameters, *parameter* can be a literal, a parameter marker or NULL.
- For the short format, the return value must be a parameter marker.
- For *procedure-name*, you can supply an identifier or a parameter marker.

Unlike the ODBC standard, all parameters defined for the stored procedure must be supplied on the CALL statement.

If you use the second format of the CALL statement or the shorthand format without a return value and your stored procedure application returns a non-zero return code (or the stored procedure sets a non-zero value for SQLCODE in the SQLCA) then:

- The RETCODE will either be set to SQL_ERROR or SQL_SUCCESS_WITH_INFO, depending on the return code (or SQLCODE) value, or you will need to
- Issue the SQLError call and inspect the pfNativeError parameter to obtain the return code (or SQLCODE) value.

If you want to use the shorthand version of the CALL and specify a return code value, follow these rules when you define the stored procedure:

- The first *parameter-declaration* must be OUT.
- The first *parameter-declaration parameter-name* must be RC.
- The first *parameter-declaration parameter-type* must be INTEGER.

Required: If your stored procedure application program uses non-zero return code values to report warning conditions that are not errors, you must use the shorthand format specifying an RC value if you want to inspect any data returned from the stored procedure application program. When a non-zero return code is encountered, the query processor will not update the SQLDA returned to the client with any updated output or input-output parameters that the stored procedure modified. If you specify an RC parameter and a non-zero return code is encountered, the stored procedure connector updates the RC parameter with the return code value and returns zeros to the query processor. This enables the stored procedure to return any updated values to the client and the SQLExecDirect call to report SQL_SUCCESS.

For more information about the SQLExecDirect and SQLBindParameter calls, see the Microsoft information about ODBC APIs..

Creating result sets in stored procedures

Result sets extend the capability of output parameters by allowing multiple rows of data to be returned in a single invocation of a stored procedure.

The creation of result sets does not preclude the use of output parameters. You can build procedures that return both output parameters and results sets. You can only use result sets from stored procedures that are compliant with Language Environment (LE).

When a stored procedure returns a result set, the result set is automatically in an open state after a successful call to the procedure. This means the application can immediately begin fetching the result set rows, just as it does after opening a prepared cursor. The main difference between opening a cursor and executing a stored procedure that returns a result set is that the description of the result set cannot be determined until the procedure is successfully called.

Stored procedures can return only one result set per application call. The number, names, and SQL types of the columns in a result set are determined by the stored procedure itself, and can vary from call to call if there is an application reason to do so.

To create a result set, the stored procedure must be defined in the system catalog as capable of returning a result set. To do this, the procedure must be specified with `RESULT SETS 1` in the `CREATE PROCEDURE` statement that defines it.

The interface routines `CACRSCR`, `CACRSIN`, and `CACSADDR` enable your stored procedure to create result sets. The routines are callable by any language used to implement a stored procedure.

CACRSCR interface routine

The `CACRSCR` interface routine enables your stored procedure to create result sets. The routine creates an empty result set using an `SQLDA` that describes the columns in the result set.

With this routine, the developer of the stored procedure can optionally designate ascending or descending columns to order by. The routine is callable by any language used to implement a stored procedure.

Parameters

The parameters for the routine `CACRSCR` are:

- The `SQLDA` passed to the stored procedure on invocation. This `SQLDA` describes the original parameters passed to the procedure.

Important: Failure to pass this parameter first can result in an abend in the result set processing logic.

- An `SQLDA` that describes the columns in the result set. Read this topic and related topics for more information on defining a result set `SQLDA`.
- A list of sorting criteria. This list is an array of halfword subscripts for `SQLVAR` entries in the `SQLDA` to be sorted. The list is terminated with an entry containing the value 0.

Example: To sort on the first column in ascending sequence, the array contains the entries 1 and 0. Descending sorting is specified with a negative subscript value. To sort on the third column in descending sequence, the array contains the values -3 and 0.

Return codes

`CACRSCR` returns the following return codes:

- | | |
|---|--|
| 0 | An empty result set was successfully created. |
| 1 | The <code>SQLDAID</code> field does not contain the literal string 'SQLDA.' |
| 2 | The count field <code>SQLD</code> is invalid. It is either less than or equal to 0 or does not match the value in the <code>SQLN</code> field. |
| 3 | The stored procedure is not defined in the System Catalog as returning a result set. The grammar defining the stored procedure must specify <code>RESULT SETS 1</code> . |
| 4 | A result set has already been created in the current invocation of the stored procedure. Only one result set is allowed per invocation. |
| 5 | The stored procedure was called by a client <code>EXEC IMMEDIATE</code> request. A result set cannot be returned when <code>EXEC IMMEDIATE</code> is used. |
| 6 | The sorting list passed is invalid. One of the values in the list exceeds the number of columns in the result set. |

101-199 An invalid name was passed for sqlvar rc-100.

201-299 An invalid sqltype was specified for sqlvar rc-200.

301-399 An invalid sqllen was specified for sqlvar rc-300

>1000 Internal error. Contact IBM Technical Support

Sample

Sample COBOL usage of CACRSCR.

WORKING-STORAGE SECTION.

```
* DEFINE THE RESULT SET SQLDA
01 RESULT-SET-SQLDA.
   05 RS-SQLDAID PIC X(08) VALUE 'SQLDA'.
* SQLDABC IS CALCULATED AS 16 + (NBR COLUMNS * 44)
   05 RS-SQLDABC PIC 9(08) COMP VALUE 60.
* SQLN AND SQLD ARE THE NUMBER OF COLUMNS
   05 RS-SQLN PIC 9(04) COMP VALUE 1.
   05 RS-SQLD PIC 9(04) COMP VALUE 1.
   05 RS-SQLVAR1.
      10 RS-SQLTYPE1 PIC 9(04) COMP VALUE 453.
      10 RS-SQLLEN1 PIC 9(04) COMP VALUE 8.
      10 RS-SQLDATA1 PIC S9(8) COMP.
      10 RS-SQLIND1 PIC S9(8) COMP.
      10 RS-SQLNAME1.
         20 RS-SQLNAME1-LEN PIC 9(4) COMP VALUE 5.
         20 RS-SQLNAME1-NAME PIC X(30) VALUE 'CHAR8'.

* DO AN ASCENDING SORT ON THE CHAR8 COLUMN
01 SORTING-LIST.
   05 SORT-FIRST-ASCENDING PIC S9(4) COMP VALUE 1.
   05 END-OF-SORTING-LIST PIC S9(4) COMP VALUE 0.
```

LINKAGE SECTION.

01 ARG-SQLDA.

.

PROCEDURE DIVISION USING ARG-SQLDA, ARG-SQLCA

.

```
* CREATE THE RETURN RESULT SET
CALL 'CACRSCR' USING ARG-SQLDA, RESULT-SQLDA, SORTING-LIST.
```

.

CACRSIN interface routine

The CACRSIN interface routine Inserts a single row into a previously-created result set. Like CACRSCR, this routine receives the column information in the form of an SQLDA structure.

Parameters

The parameters for the routine CACRSIN are:

- The SQLDA passed to the stored procedure on invocation. This SQLDA describes the parameters passed to the procedure.

Tip: Failure to pass this parameter first can result in an abend in the result set processing logic.

- An SQLDA describing the columns and data to be inserted into the result set. When inserting rows in the result set, the sqldata and sqlind variables for each column must be set to the address of the column data to insert. This SQLDA must be the same SQLDA structure used to create the result set.

Return codes

CACRSIN returns the following return codes:

- 0 A new row was successfully inserted into the result set.
- 1 The SQLDAID field does not contain the literal string SQLDA.
- 2 The count field SQLD is invalid. It is either less than or equal to 0 or does not match the value in the SQLN field.
- 3 The SQLDA address passed is not the same as the SQLDA used to create the result set.
- 4 The number of sqlvars in the SQLDA passed is not the same as the number passed when the result set was created.

201-299

An invalid sqltype was specified for sqlvar rc-200.

301-399

An invalid sqllen was specified for sqlvar rc-300.

401-499

The data passed in sqldata is invalid for sqlvar rc-400.

501-599

The indicator passed in sqlind is invalid for sqlvar rc-500.

>1000 Internal error. Contact IBM Technical Support.

Sample

Sample COBOL usage of CACRSIN.

```

WORKING-STORAGE SECTION.
* DEFINE THE RESULT SET SQLDA
01 RESULT-SET-SQLDA.
   05 RS-SQLDAID PIC X(08) VALUE 'SQLDA'.
* SQLDABC IS CALCULATED AS 16 + (NBR COLUMNS * 44)
   05 RS-SQLDABC PIC 9(08) COMP VALUE 60.
* SQLN AND SQLD ARE THE NUMBER OF COLUMNS
   05 RS-SQLN PIC 9(04) COMP VALUE 1.
   05 RS-SQLD PIC 9(04) COMP VALUE 1.
   05 RS-SQLVAR1.
      10 RS-SQLTYPE1 PIC 9(04) COMP VALUE 453.
      10 RS-SQLLEN1 PIC 9(04) COMP VALUE 8.
      10 RS-SQLDATA1 PIC S9(8) COMP.
      10 RS-SQLIND1 PIC S9(8) COMP.
      10 RS-SQLNAME1.
         20 RS-SQLNAME1-LEN PIC 9(4) COMP VALUE 5.
         20 RS-SQLNAME1-NAME PIC X(30) VALUE 'CHAR8'.

* DO AN ASCENDING SORT ON THE CHAR8 COLUMN
01 SORTING-LIST.

```

```

        05 SORT-FIRST-ASCENDING          PIC S9(4) COMP VALUE 1.
        05 END-OF-SORTING-LIST          PIC S9(4) COMP VALUE 0.

01  CHAR8-COLUMN-DATA                  PIC X(8).

01  CHAR8-NULL-IND                     PIC S9(4) COMP VALUE 0.

LINKAGE SECTION.
.
.
.

PROCEDURE DIVISION USING ARG-SQLDA, ARG-SQLCA
.
.
.
*  CREATE THE RETURN RESULT SET
CALL 'CACRSCR' USING ARG-SQLDA, RESULT-SQLDA, SORTING-LIST.

*  SET THE SQLDATA AND SQLIND POINTERS IN THE RESULT SET
*  SQLDA

CALL 'CACSADDR' USING RS-SQLDATA1, CHAR8-COLUMN-DATA,
                    RS-NULL-IND.

*  INSERT 2 ROWS INTO THE RESULT SET
MOVE 'ROW 1' TO CHAR8-COLUMN-DATA.
CALL 'CACRSIN' USING ARG-SQLDA, RESULT-SQLDA.
MOVE 'ROW 2' TO CHAR8-COLUMN-DATA.
CALL 'CACRSIN' USING ARG-SQLDA, RESULT-SQLDA.
.
.
.

```

CACSADDR interface routine

The CACSADDR interface routine is used by COBOL programs to set the addresses of sqldata and sqlind data items in an SQLVAR.

Parameters

The parameters for the routine CACSADDR are:

- The address of an sqldata variable to be set.
- The address of the column data to be placed in sqldata.
- The address of and indicator variable to be placed in the sqlind variable that immediately follows sqldata. This parameter is optional if an sqlind variable is not needed.

Sample

Sample COBOL usage of CACSADDR:

```

* SET RS-SQLDATA1 TO THE ADDRESS OF CHAR8-COLUMN-DATA AND
*   RS-SQLIND1 TO THE ADDRESS OF RS-NULL-IND
CALL 'CACSADDR' USING RS-SQLDATA1, CHAR8-COLUMN-DATA,
                    RS-NULL-IND.

* SET RS-SQLDATA2 TO THE ADDRESS OF INTEGER-DATA
CALL 'CACSADDR' USING RS-SQLDATA2, INTEGER-DATA.

```

Modifying a COBOL stored procedure to return a result set

To create a result set in a COBOL stored procedure, define an SQLDA and pass the result set that the SQLDA creates to the CACRSCR routine.

The application logic that is necessary to create a result set is relatively straightforward. The stored procedure program first defines an SQLDA that describes the result set and passes the result set to the CACRSCR routine. A sample SQLDA for a three column result set is shown in the following code sample:

Table 5x Sample COBOL SQLDA.

```

01 RESULT-SET-SQLDA.
   05 RS-SQLDAID                PIC X(08) VALUE 'SQLDA'.
*  SQLDABC IS CALCULATED AS 16 + (NBR COLUMNS * 44)
   05 RS-SQLDABC                PIC 9(08) COMP VALUE 148.
*  SQLN AND SQLD ARE THE NUMBER OF COLUMNS
   05 RS-SQLN                   PIC 9(04) COMP VALUE 3.
   05 RS-SQLD                   PIC 9(04) COMP VALUE 3.
   05 RS-SQLVAR1.
      10 RS-SQLTYPE1            PIC 9(04) COMP VALUE 453.
      10 RS-SQLLEN1            PIC 9(04) COMP VALUE 8.
      10 RS-SQLDATA1          PIC S9(8) COMP.
      10 RS-SQLIND1           PIC S9(8) COMP.
      10 RS-SQLNAME1.
         20 RS-SQLNAME1-LEN    PIC 9(4) COMP VALUE 5.
         20 RS-SQLNAME1-NAME  PIC X(30) VALUE 'CHAR8'.
   05 RS-SQLVAR2.
      10 RS-SQLTYPE2            PIC 9(04) COMP VALUE 497.
      10 RS-SQLLEN2            PIC 9(04) COMP VALUE 4.
      10 RS-SQLDATA2          PIC S9(8) COMP.
      10 RS-SQLIND2           PIC S9(8) COMP.
      10 RS-SQLNAME2.
         20 RS-SQLNAME2-LEN    PIC 9(4) COMP VALUE 8.
         20 RS-SQLNAME2-NAME  PIC X(30) VALUE 'LARGEINT'.
   05 RS-SQLVAR3.
      10 RS-SQLTYPE3            PIC 9(04) COMP VALUE 501.
      10 RS-SQLLEN3            PIC 9(04) COMP VALUE 2.
      10 RS-SQLDATA3          PIC S9(8) COMP.
      10 RS-SQLIND3           PIC S9(8) COMP.
      10 RS-SQLNAME3.
         20 RS-SQLNAME3-LEN    PIC 9(4) COMP VALUE 8.
         20 RS-SQLNAME3-NAME  PIC X(30) VALUE 'SMALLINT'.

```

In addition to defining the result set SQLDA, the stored procedure must define a sorting list to pass to the result set creation entry CACRSCR. If no sorting is necessary, the list needs only one variable as shown below:

```

01 NO-SORTING.
   05 END-OF-LIST    PIC S9(4) COMP VALUE 0.

```

After you define the result set and sorting list, the stored procedure calls CACRSCR to create the result set, and makes one call to CACRSIN for each row of data to be added to the result set.

Linking the result set interface into a z/OS load module

When you dynamically call or statically link interface modules that define result sets into your stored procedure load module, you can then return a result set to your calling application. You can also use static calls.

If you use static calls, you must include the load module CACSPRS in the link step of a stored procedure compile and link job. This module is delivered in the SCACLOAD distribution library.

Defining the stored procedure to the metadata catalog

To return a result set to your calling application, you must include a RESULT SETS specification in the CREATE PROCEDURE statement for your stored procedure that enables your stored procedure to return a result set.

Stored procedures defined with a RESULT SETS specification other than 0 are not required to return result sets, but have the option of creating a result set when necessary. A sample definition of a stored procedure that is enabled for result sets follows:

```
CREATE PROCEDURE CAC.SP001
  ( IN   INPUT_PARM  CHAR(8),
    OUT  OUTPUT_PARM CHAR(8) )
  RESULT SETS 1
  EXTERNAL NAME 'SP001'
  LANGUAGE COBOL
  STAY RESIDENT YES;
```

Pass the CREATE PROCEDURE statement as input to the metadata utility to add the definition to the metadata catalog.

Client application and result set interaction

The ODBC, CLI, and JDBC clients can check for result sets. The client program is responsible for closing any result set that is opened by a stored procedure call.

Client applications that call stored procedures must be aware of the procedures that return result sets. ODBC and CLI clients can check for the existence of a result set by calling the ODBC function SQLNumResultCols after executing a stored procedure. If SQLNumResultCols returns any number other than 0, then a result set is open and ready for fetching.

JDBC clients can invoke the CallableStatement method ExecuteQuery when the stored procedure always returns one and only one result set. If there is any doubt, clients must call the Execute method. The Execute method returns true if a procedure returns a result set, and the client retrieves the result set by calling the getResultSet method.

Support routines for stored procedures

You can use support routines in your stored procedures to fetch user IDs, passwords, and run option values.

The Classic federation implementation passes more information to your stored procedure than the SQLDA structure. A hidden area precedes the SQLDA that the product uses to establish and maintain communications with other Classic federation components.

The support routines enable you to get a copy of the connected user ID and password, and a copy of the RUN OPTIONS parameter in the CREATE PROCEDURE statement. Unlike the CICS and IMS DRA interface, these support routines are in object form for direct inclusion in your stored procedure. These support routines are written in assembler language and accept the following parameters:

- The SQLDA structure
- The address of an output field where the requested information is moved

Unlike the CICS, CA-Datacom, and IMS interfaces that are distributed as load modules, these support routines are supplied in object-module form for direct

inclusion in your stored procedure. These routines are written in Assembler Language and use standard linkage conventions. All routines are passed two parameters.

- The first parameter must be the address of the SQLDA parameter passed to the stored procedure.
- The second parameter is an address where the value is copied.

Exception: Like the CICS and IMS DRA interface, these support routines are written to accept 31-bit addresses. However, unlike the CICS and IMS DRA interfaces, these support routines do not perform validation checks to confirm that the first parameter is a pointer to the SQLDA. Failure to pass the SQLDA as the first parameter generally results in some form of addressing exception abend.

Get RUN OPTIONS (CACSPGRO) calling conventions

The support routine CACSPGRO copies the RUN OPTIONS parameter specified on the CREATE PROCEDURE statement into a 254-byte work area that your stored procedure application program supplies.

The following figure shows the calling conventions used to invoke CACSPGRO.

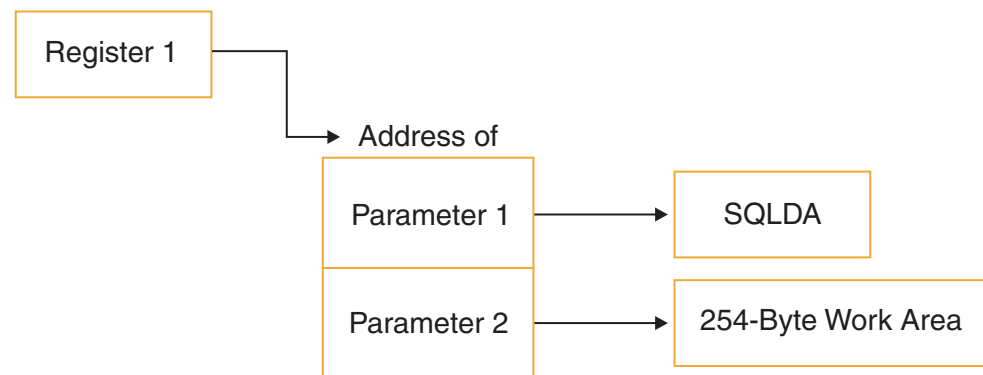


Figure 8. CACSPGRO calling conventions

CACSPGRO always issues a return code of zeros. Upon return, the area that parameter 2 references contains a copy of the RUN OPTIONS parameter padded with blanks.

To link-edit CACSPGRO into your stored procedure application, concatenate the SCACSAMP library into the SYSLIB DD statement on the link step and include the following in the SYSIN DD statement on the link step:

```
INCLUDE SYSLIB(CACSPGRO)
```

Get user ID (CACSPGUI) calling conventions

The CACSPGUI support routine copies the user ID of the connected user into an 8-byte work area that your stored procedure supplies.

The following figure shows the calling conventions that are required to invoke CACSPGUI.

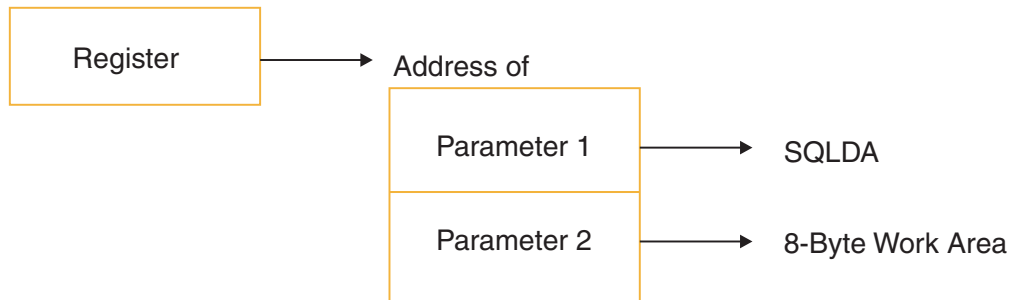


Figure 9. CACSPGUI calling conventions

CACSPGUI always issues a return code of zeros. Upon return, the area that parameter 2 references contains a copy of the user ID padded with blanks. If no user ID is supplied, the output area contains spaces.

To link-edit CACSPGUI into your stored procedure application, concatenate library SCACSAMP into the SYSLIB DD statement on the link step and include the following in the SYSIN DD statement on the link step:

```
INCLUDE SYSLIB(CACSPGUI)
```

Get user password (CACSPGPW) calling conventions

The CACSPGPW support routine copies the connected user password into an 8-byte work area that your stored procedure supplies.

The following figure shows the calling conventions that are required to invoke CACSPGPW.

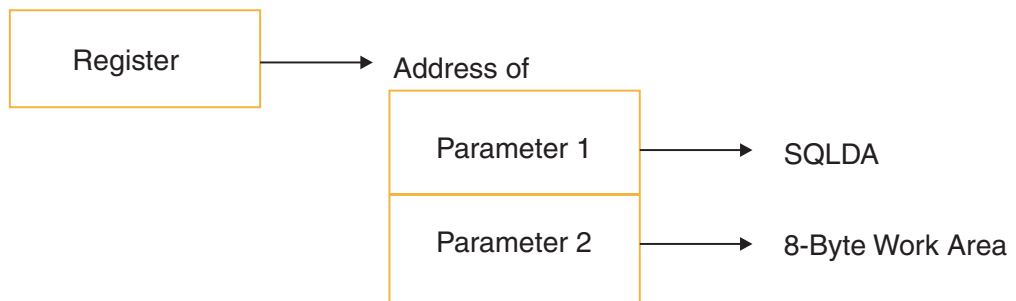


Figure 10. CACSPGPW calling conventions

CACSPGPW always issues a return code of zeros. Upon return, the area referenced by parameter 2 contains a copy of the user password padded with blanks. If no password is supplied, the output area contains spaces.

To link-edit CACSPGPW into your stored procedure application, concatenate library SCACSAMP into the SYSLIB DD statement on the link step and include the following in the SYSIN DD statement on the link step:

```
INCLUDE SYSLIB(CACSPGPW)
```

CICS interface for stored procedures

The CICS interface enables stored procedures that run in the data server address space to communicate with CICS and execute a CICS application program.

The stored procedure CACSPVCM is supplied for basic operations. CACSPVCM uses the CICS interface bridge CACSPBR to initiate an APPC conversation with CICS. The RUN OPTIONS parameter for CACSPVCM links to a user-written CICS application that performs the actual processing. The following operations take place:

- CACSPVCM sends the SQLDA that is supplied by the client application to the CICS application.
- The CICS application sends CACSPVCM an updated version of the SQLDA.
- CACSPVCM deallocates the APPC conversation and returns the SQLDA to the client application.

In most situations, CACSPVCM is a sufficient stored procedure application program. If CACSPVCM does not meet your needs, you can use the CACSPVTM copybook which is provided for writing your own stored procedure application program. CACSPVCM, uses this interface to communicate with CACSPBR. Writing your own stored procedure to interface with CICS is required in the following situations:

- You need to invoke two or more CICS applications.
- You need to run programs in the data server address space and then run a CICS application.

CACSPVCM is written in assembler, and source code is not available. To help you understand how to interface with CACSPBR, a COBOL version of CACSPVCM is provided in the sample library SCACSAMP (CACSPCOM).

Specifying CICS transaction scheduling information

For the CACSPVCM stored procedure to communicate with CICS, you need to use the RUN OPTIONS parameter to specify the information that CACSPVCM needs to communicate with CICS and invoke the CICS transaction.

If you are writing your own stored procedure to communicate with CICS, then you should specify the CICS communication information using the RUN OPTIONS parameter instead of hard coding it in your stored procedure. Doing so allows you to drop the stored procedure definition, update the CREATE PROCEDURE statement, and redefine the stored procedure when your environment changes, without requiring program changes.

CICS transaction scheduling information is identified by the `_CICS` keyword in the RUN OPTIONS parameter. The format of the `_CICS` keyword is:

```
_CICS(Local-LU-Name,CICS-Applid,Logmode-Table-Name,Transaction-ID,Program-Name)
```

All subparameters must be supplied, comma delimited as shown, and must not contain any spaces.

The following table describes the size and purpose of each sub-parameter.

Table 40. CICS transaction scheduling subparameters

Subparameter name	Maximum length	Description
Local-LU-Name	8	Identifies the name of a pool of VTAM logical units that can be used by CACSPBR to communicate with CICS, such as CACPPC0*. In the above example, CACSPBR initially attempts to open an ACB for LU name CACPPC00. If an ACB open error is returned, or a CNOS negotiation error is reported by CICS, then CACSPBR attempts to open an ACB named CACPPC01. If that fails LU names CACPPC02-CACPPC09 are tried until no errors are reported or all names have been attempted. Only one set of wildcard characters can be specified (for example, CAC*PC0* is invalid). Up to seven wild card characters can be supplied (not recommended).
CICS-Appid	8	Identifies the APPLID of the CICS target subsystem.
Logmode-Table-Name	8	Identifies the VTAM logmode table entry to be used. This name must identify an entry in the Logon Mode Table definition, in the Local LU Name APPL definitions and in the CICS SESSIONS definitions.
Transaction-ID	4	Identifies the name of the CICS transaction that has been defined for CACSP62 to allow communications between the Server and CICS. This name must be 4 characters long.
Program-Name	8	Identifies the name of the CICS program that CACSP62 is to LINK to. This name must be defined as a PROGRAM to CICS.

Stored procedure and CICS communication

Communications with CICS is performed using VTAM LU 6.2 with the VTAM connection handler.

An API interface load module, CACSPBR, is provided. A stored procedure application running in the data server address space that communicates with the VTAM connection handler can call CACSPBR to perform these tasks:

- Establish a session with CICS
- Send data
- Receive data
- Perform address translation for the updated parameter list returned from the CICS application program
- End the session

The system also supplies a CICS LU 6.2 application, CACSP62, which is the partner for CACSPBR. This application is responsible for these tasks:

- Performing address translation for the parameter list being passed to the CICS transaction
- Invoking the specified CICS application program via an exec CICS link
- Sending return code or CICS abend codes back to the server
- Deallocating the session in the case of a CICS abend

The CACSPBR interface allows a stored procedure to send and receive multiple transmissions between itself and CICS. In normal situations, only a single send and receive is required. For these situations, the system supplies an Assembler Language stored procedure (CACSPVCM) that sends the client parameter list to CICS and receives (a possibly) updated parameter list from CICS. For most

situations, using CACSPVCM eliminates the need for you to develop your own stored procedure to invoke a CICS application program.

The following figure shows the processing flow when the stored procedure application program interfaces with CICS.

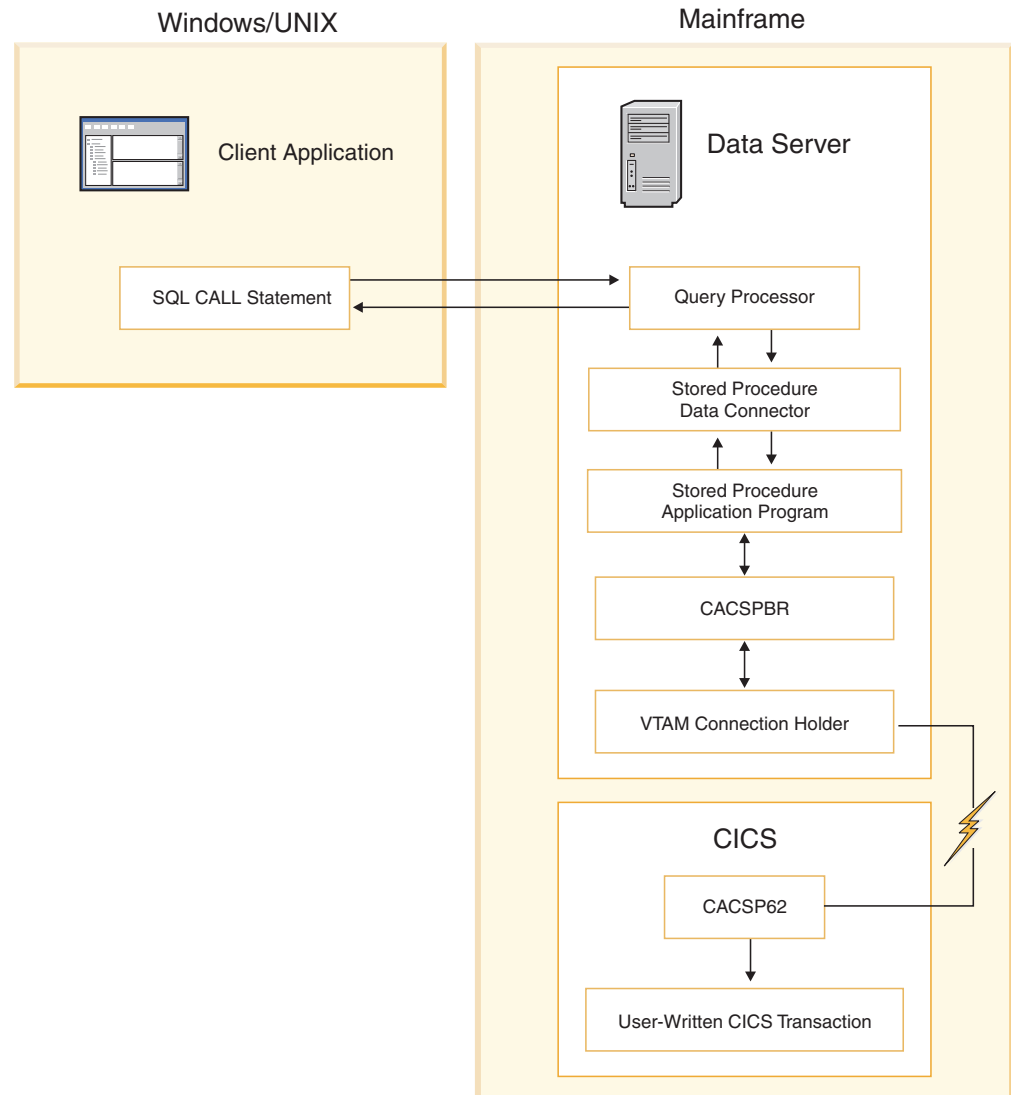


Figure 11. CICS processing flow

In addition, extensions to the RUN OPTIONS parameter enable you to specify the required CICS transaction scheduling information to invoke your user-defined CICS application program.

Important: If you write your own stored procedure application program and specify CICS transaction scheduling information on the RUN OPTIONS parameter, the information you supply overrides values that might be set in your stored procedure application.

CACSPBR interface

The API interface to the CICS interface bridge enables stored procedures to open CICS conversations and execute a CICS application program.

The CICS interface bridge (CACSPBR) is distributed as a separate load module that you can dynamically call or statically link to from your stored procedure.

The CACSPBR interface uses standard assembler linkage conventions. Stored procedures pass CACSPBR three parameters:

- An APPC function structure
- An APPC data structure
- The SQLDA that is passed to the stored procedure

Figure 12 shows how your stored procedure must pass parameters to CACSPBR. Additionally, your stored procedure must call CACSPBR by using variable-argument list calling conventions. Variable-argument list calling conventions are standard for most high-level languages, such as COBOL. Set the high-order bit of the SQLDA to a value of 1.

Additionally, CACSPBR is linked as an AMODE(31) RMODE(ANY) application. Stored procedures must pass all addresses to CACSPBR in 31-bit addressing mode.

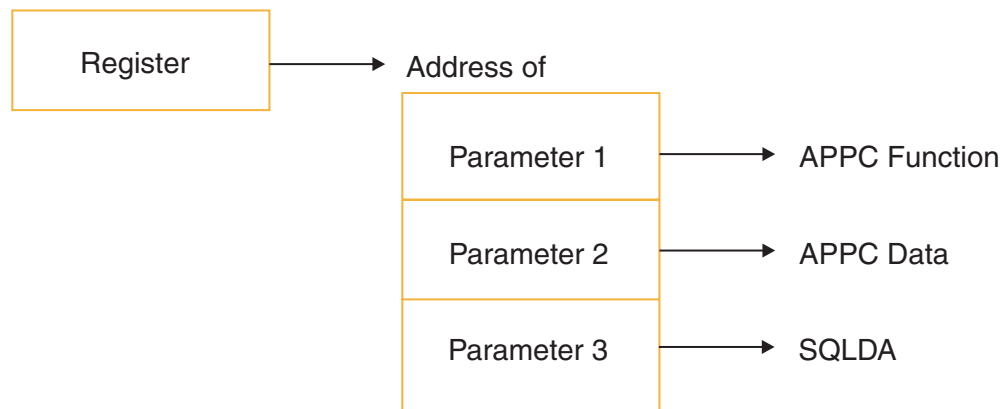


Figure 12. Parameters passed to CACSPBR

Upon return from CACSPBR, Register 15 contains a return code value that indicates whether the requested function completed successfully or not.

Sample member CACSPVTM is a COBOL copybook that shows the structure and contents of both the APPC function and APPC data structures.

```

01 APPC-FUNCTION.
  05 APPC-REQUEST PIC X(8).

01 APPC-DATA.
  05 APPC-DATA-IDENTIFIER PIC X(8).
  05 LOCAL-LU-NAME PIC X(8).
  05 CICS-SYSTEM-APPLID PIC X(8).
  05 APPC-MODE-ENTRY-NAME PIC X(8).
  05 CICS-TRANSACTION-ID PIC X(4).
  05 CICS-SP-PROGRAM-NAME PIC X(8).
  05 CICS-SP-RETCODE PIC 9(8) COMP.
  05 CICS-SP-ABENDCODE
    REDEFINES CICS-SP-RETCODE PIC X(4).
  05 COMM-RETCODE PIC S9(8) COMP.
  
```

The following table describes the contents of the APPC function structure.

Table 41. APPC function structure contents

COBOL name	SQL data type	Description
APPC-REQUEST	CHAR(8)	Identifies the function that CACSPBR needs to perform. Valid functions are: <ul style="list-style-type: none"> • OPEN - Start a conversation with CICS • SEND - Send the SQLDA to CICS • RECEIVE - Receive an updated copy of the SQLDA from CICS • CLOSE - Deallocate the CICS conversation

The following table describes the contents of the APPC data structure.

Table 42. APPC data structure contents

COBOL name	SQL data type	Description
APPC-DATA-IDENTIFIER	CHAR(8)	Signature field that identifies the structure as the APPC data structure. Must contain the value "APPCDATA".
LOCAL-LU-NAME	CHAR(8)	Identifies the name of a pool of VTAM logical units (LUs) that can be used by CACSPBR to communicate with CICS, for example, CACPPC0*. In Figure 12 on page 236, CACSPBR initially attempts to open an ACB for LU name CACPPC00. If an ACB open error is returned, or a CNOS negotiation error is reported by CICS, then CACSPBR attempts to open an ACB named CACPPC01. If that fails LU names CACPPC02-CACPPC09 are tried until no errors are reported or all names have been attempted. You can specify only one set of wildcard characters. (For example, CAC*PC0* is invalid.) You can use up to seven wild card characters, but supplying multiple wild cards is not a recommended practice.
CICS-SYSTEM-APPLID	CHAR(8)	Identifies the APPLID of the CICS target subsystem.
APPC-MODE-ENTRY-NAME	CHAR(8)	Identifies the VTAM logmode table entry. This name must identify an entry in the Logon Mode Table definition, in the Local LU Name APPL definitions and in the CICS SESSIONS definitions.
CICS-TRANSACTION-ID	CHAR(4)	Identifies the name of the CICS transaction that is defined for CACVT62 to allow communications between the server and CICS. This name must be 4-characters long.
CICS-SP-PROGRAM-NAME	CHAR(8)	Identifies the name of the CICS program that CACVT62 links to. This name must be defined as a PROGRAM to CICS.
CICS-SP-RETCODE	INTEGER	CICS error return code. The value is zero if no CICS errors are reported. If CICS-SP-RETCODE is less than zero, then a CICS error has occurred. The stored procedure should then inspect CICS-SP-ABENDCODE for the error code.
CICS-SP-ABENDCODE	CHAR(4)	Abend code that CICS returns.

Table 42. APPC data structure contents (continued)

COBOL name	SQL data type	Description
COMM-RETCODE	INTEGER	When CACSPBR detects a communications error, COMM-RETCODE contains a return code.

Typically, you can supply a value of spaces for the LOCAL-LU-NAME, CICS-SYSTEM-APPLID, APPC-MODE-ENTRY-NAME, CICS-TRANSACTION-ID, and CICS-SP-PROGRAM-NAME fields. Define the information associated with these parameters in the RUN OPTIONS parameter of the CREATE PROCEDURE statement instead. CACSPBR gives precedence to information you supply in RUN OPTIONS over information supplied in the APPC data fields.

Each time your stored procedure calls CACSPBR, the CICS-SP-RETCODE and COMM-RETCODE must contain zeros. Upon return from CACSPBR, your stored procedure must perform the following checks:

- Is the RETURN-CODE (Register 15) non-zero? If so, COMM-RETURN code might contain a data server communications return code.
- Inspect CICS-SP-RETCODE for a non-zero value. If the value is less than zero, inspect CICS-SP-ABENDCODE to determine the error code that CICS reports. If your stored procedure receives a non-zero CICS-SP-RETCODE or COMM-RETURN, CACSPBR reports it to the client application. You can suppress sending these return codes to the client application by setting the SQLCODE field in the SQLCA structure to a non-zero value.

Call CACSPBR with the APPC-REQUEST field values set in the following sequence.

1. OPEN
2. SEND
3. RECEIVE
4. CLOSE

The call can issue multiple sends and receives, but should do so only when you need to run multiple CICS applications. The stored procedure must pass to CACSPBR the SQLDA structure that the client application passed to the stored procedure. Additionally, you cannot alter the SQLDA structure contents. You can change only the data referenced by ARG-SQLDATA and the associated null indicators referenced by ARG-SQLIND. The following table provides an overview of the processing that CACSPBR and its counterpart CACSP62 perform for each APPC-REQUEST function.

Table 43. Processing overview by function

FUNCTION	CACSPBR actions	CACVT62 actions
OPEN	<p>Attempts to open a VTAM ACB based on the information supplied in the following fields:</p> <ul style="list-style-type: none"> • LOCAL-LU-NAME • CICS-SYSTEM-APPLID • APPC-MODE-ENTRY-NAME • CICS-TRANSACTION-ID <p>If the ACB open fails and LU pooling information was supplied in the field LOCAL-LU-NAME, the OPEN function generates a new name and makes another attempt until the OPEN function has tried all available LU names.</p> <p>In addition, the OPEN function passes the user ID and password to CICS for security checking. If the connected user that is running the stored procedure supplied no user ID, the OPEN function sends the value NO_USER and a blank password to CICS.</p>	<p>Saves the conversation ID supplied by CICS and allocates work buffers.</p>
SEND	<p>Sends the CICS-SP-PROGRAM-NAME and the SQLDA to CICS.</p>	<p>Receives a copy of the SQLDA and performs address translation so that the CICS application program can access the contents of the SQLDA ARG-SQLDATA and ARG-SQLIND fields that are referenced in the SQLDA.</p> <p>Issues a LINK for the program name that is contained in CICS-SP-PROGRAM-NAME, waits for control to return, and sends the copy of the SQLDA back to CACSPBR.</p> <p>If CICS reports an abend, the abend code is sent back to CACSPBR.</p>
RECEIVE	<p>Receives a copy of the SQLDA from CICS and performs address translation so that the new copy is referenced by the stored procedure application program.</p>	
CLOSE	<p>Closes the VTAM ACB, which terminates the conversation with CICS.</p>	<p>Receives a deallocation request and frees resources that are allocated when the conversation begins.</p>

Parameters passed to the CICS application program

When you execute CICS application programs, use the parameters that CACSP62 passes to a CICS application program to establish communication.

CACSP62 passes three parameters that are found in the communications area to CICS application programs. Sample member CACSPDFH is a COBOL copybook that defines the communications area. The following example shows the contents of that copybook:

```

01 DFHCOMMAREA.
   05 APP-RETURN-CODE          PIC 9(8) COMP.
   05 ARG-DATA-POINTER        POINTER.
   05 ARG-DATA-LENGTH         PIC 9(8) COMP.

```

The following table describes the contents of each of the fields in the communications area. Your application program must establish addressability to the SQLDA before inspecting or updating its contents.

For example, in COBOL you issue:

```
SET ARG-DATA TO ARG-DATA-POINTER .
```

The above example assumes that you included a copy of SCACSPAMP member CACSPSDA in the linkage section of your CICS application program.

Table 44. Communication area contents

COBOL name	SQL data type	Description
APP-RETURN-CODE	INTEGER	The fullword return code value that the CICS application uses to report whether processing was successful or not. The contents of this field are placed in the CICS-SP-RETCODE field in the APPC data area for inspection by your stored procedure running in the data server's address space. This return code percolates to the client application program, unless your stored procedure overrides this code by using the SQLCODE in the SQLCA.
ARG-DATA-POINTER	INTEGER	The address of the copy of the SQLDA that was sent to CICS. Your CICS application program must not modify this field.
ARG-DATA-LENGTH	INTEGER	The length of the SQLDA passed to your CICS application program. Your CICS application program must not modify this field.

After your CICS application program establishes addressability to the SQLDA, the CICS program establishes addressability to the data and null indicator values in the SQLDA.

Compiling and linking applications that use CACSPBR

Use the following sample job stream to guide you when you use a supplied JCL to compile and link applications that use CACSPBR.

About this task

The member CACSPCCC in the SCACSKEL library is a sample job stream that demonstrates how to compile and link CACSPCOM including CACSPBR. The following example is a copy of CACSPCCC.

To compile and link CACSPCOM by using the supplied Job Control Language (JCL):

1. Supply an appropriate job card.
2. Modify the PROC parameters (LE, COBOL, SOUT, and so on) to specify correct values for your site.
3. Modify the COMPILE step SYSIN DD statement to specify the correct source library.
4. Modify the LKED step OBJ DD statement to specify the correct object library.
5. Modify the LKED step SYSLMOD DD statement to specify the correct load library.
6. Modify the PROC statement to specify the correct high level name of the application libraries.

CACSPBR return codes

If CACSPBR detects an error, or an error is reported by the VTAM LU 6.2 communication handler, CACSPBR returns a negative return code.

The following table provides a brief description of each return code, possible methods to resolve the problem, and whether a more detailed explanation of the error condition can be found in the server's output log.

Table 45. CACSPBR return codes

Return code	Description	Logged
-102	Invalid buffer version identifier. Either your stored procedure did not pass the SQLDA that was passed to your application as the third parameter to CACSPBR, or the SQLDA was corrupted. Verify that your stored procedure passed SQLDA to CACSPBR as the third parameter. If you are passing the SQLDA as the third parameter, then contact IBM Technical Support.	No
-116	Invalid stored procedure internal identifier. See the description of return code -102 for problem resolution information.	No
-117	Invalid sqln value in the SQLDA. See the description of return code -102 for problem resolution information.	No
-121	Your stored procedure passed an incorrect number of parameters to CACSPBR. Verify that your program is passing the correct number of parameters to CACSPBR. If the correct number of parameters are passed, contact IBM Technical Support.	No
-122	Invalid APPC function was passed. Valid functions are OPEN, SEND, RECEIVE, and CLOSE.	No
-123	Invalid local LU name was passed to CACSPBR. Probable cause is that the LU name contains multiple sets of LU pooling wildcard characters or that too many wildcard characters were supplied. Verify that a correct local LU name was passed.	No
-131	CACSPBR's attempt to register itself with the VTAM LU 6.2 connection handler failed. Review the data server output log for the error that is reported.	Yes
-132	Error reported by the VTAM LU 6.2 connection handler while processing the OPEN function. Review the data server output log for the error that is reported.	Yes
-141	Error reported by the VTAM LU 6.2 connection handler while processing the SEND function. Review the data server output log for the error that is reported.	Yes
-151	Error reported by the VTAM LU 6.2 connection handler while processing a RECEIVE function. Review the data server output log for the error that is reported.	Yes
-152	Disconnect reported when processing the RECEIVE function. If the disconnect is reported due to an error condition, then the error condition is logged in the data server output. If the disconnect occurred because CACSP62 decided to deallocate the conversation, then no error is logged.	For some cases.
-162	Error reported by the VTAM LU 6.2 connection handler while processing the CLOSE function. Review the data server output log for the error that is reported.	Yes

CACSP62 abend codes

When a user-defined stored procedure requests communication with a CICS system, CACSP62 abend codes can occur.

The stored procedure implementation provides a CICS- based LU6.2 communication program, CACSP62. This program is needed when a user-defined

stored procedure program requests communication with a CICS system by using the data server. The communication functions that this program performs are specialized, and designed to support the stored procedure implementation.

This CICS program executes as a transaction. The transaction is initiated when the user-defined stored procedure program tells the data server to OPEN a communication session with a specific CICS system, identifying this program by transaction name. You assign the transaction name when the software is installed in the CICS system.

Certain failures might occur within this communication process. The user is notified of failures in this communication processor with standard CICS abends. Each error is assigned a specific abend code for easier problem determination. Normal CICS abend handling is allowed to produce diagnostic materials as defined by the user site. The abend code format is SP*nm*, where *nm* is replaced by alphanumeric characters. The abend code prefix SP is fixed and not user-configurable. The following table describes abends that can occur and the causes of these abends.

Table 46. CACSP62 abend codes

Code	Description
SP01	The transaction was initiated by a means other than an ALLOCATE from the data server. The transaction is designed to execute as a communications server using a specific protocol that the stored procedure implementation uses. Do not attempt to initiate the transaction by any other means.
SP02	An error condition has occurred during an LU6.2 RECEIVE operation. The data server that sent the message terminated. CICS resources are freed and this abend is issued to generate a transaction dump. EIB error information from the failing RECEIVE is captured for diagnostic purposes. If the cause of the failure cannot be independently determined by reviewing the reason the data server terminated, contact IBM Technical Support. IBM Technical Support will ask you for the transaction dump and associated output, including data server output.
SP03	An error condition has occurred during an LU6.2 RECEIVE operation. The data server was notified of the error and the data server issued a DEALLOCATE, effectively terminating the communication session. CICS resources are freed and this abend is issued to generate a transaction dump. EIB error information from the failing receive is captured for diagnostic purposes. Contact IBM Technical Support. IBM Technical Support will ask you for the transaction dump and associated output, including data server output.
SP04	An error condition has occurred during an LU6.2 RECEIVE operation. The data server was notified of the error and the data server issued an ISSUE ERROR, effectively indicating the communication session should be terminated. The data server was sent an ISSUE ABEND. CICS resources are freed and this abend is issued to generate a transaction dump. EIB error information from the failing receive is captured for diagnostic purposes. Contact IBM Technical Support. IBM Technical Support will ask you for the transaction dump and associated output, including data server output.
SP05	An error condition occurred during an LU6.2 RECEIVE operation. The data server was notified of the error and it asked for additional information. The attempt to SEND EIB error information to the data server also failed. The data server sent an ISSUE ABEND. CICS resources are freed and this abend is issued to generate a transaction dump. EIB error information from both the failing RECEIVE and the failing SEND is captured for diagnostic purposes. Contact IBM Technical Support. IBM Technical Support will ask you for the transaction dump and associated output, including data server output.
SP06	An LU6.2 SYNCPOINT request was detected. The stored procedure LU6.2 communications program does not support SYNCPOINT processing. The communication partner was sent an ISSUE ABEND. Verify the communication partner is a data server. If the cause of the failure cannot be resolved, contact IBM Technical Support. IBM Technical Support will ask you for the transaction dump and associated output, including data server output.

Table 46. CACSP62 abend codes (continued)

Code	Description
SP07	An LU6.2 SYNCPOINT ROLLBACK request was detected. The stored procedure LU6.2 communications program does not support SYNCPOINT ROLLBACK processing. The communication partner was sent an ISSUE ABEND. Verify the communication partner is a data server. If the cause of the failure cannot be resolved, contact IBM Technical Support. IBM Technical Support will ask you for the transaction dump and associated output, including data server output.
SP08	An LU6.2 ISSUE SIGNAL request was detected. The stored procedure LU6.2 communications implementation does not support ISSUE SIGNAL processing. The communication partner was sent an ISSUE ABEND. The stored procedure LU6.2 connection handler program is designed to support the product's stored procedure communications with a data server. Verify the communication partner is a data server. If the cause of the failure cannot be resolved, contact IBM Technical Support. IBM Technical Support will ask you for the transaction dump and associated output, including data server output.
SP09	An error condition occurred during an LU6.2 RECEIVE operation. An illogical condition between incomplete data received and no data received was detected. The data server was sent an ISSUE ABEND. CICS resources are freed and this abend is issued to generate a transaction dump. EIB error information from the failing receive is captured for diagnostic purposes. Contact IBM Technical Support. IBM Technical Support will ask you for the transaction dump and associated output.
SP10	An LU6.2 RECEIVE operation completed normally but was accompanied by a DEALLOCATE indicator. This means the data server is not in RECEIVE mode, thereby preventing the CICS component from returning (SENDing) any processed information to the data server. The CICS resources have been freed and this abend is issued to generate a transaction dump. EIB error information is captured for diagnostic purposes. If the reason the data server issued a DEALLOCATE cannot be independently determined, contact IBM Technical Support. IBM Technical Support will ask you for the transaction dump and associated output, including data server output.
SP11	An LU6.2 RECEIVE operation completed normally, but no data was received and the data server issued a DEALLOCATE. This means the data server is not in RECEIVE mode, which prevents the CICS component from returning (SENDing) any processed information to the server. The CICS resources are freed and this abend is issued to generate a transaction dump. EIB error information is captured for diagnostic purposes. If the reason the data server issued a DEALLOCATE cannot be independently determined, contact IBM Technical Support. IBM Technical Support will ask you for the transaction dump and associated output, including data server output.
SP12	The user-defined stored procedure program that executes in CICS receives a COMMAREA that contains the address of a pointer to the argument data buffer and the length of that buffer. The buffer cannot be moved or lengthened. The user-defined stored procedure program can specify a shorter argument data buffer for return to the data server by changing the buffer length field in the COMMAREA. This abend is issued to generate a transaction dump. You must request CLOSE to terminate the communication session and release the CICS resources. Modify the user-defined stored procedure program to prevent moving the argument data buffer.
SP13	An error condition has occurred during an LU6.2 SEND operation. The data server was sent an ISSUE ABEND. CICS resources have been freed and this abend was issued to generate a transaction dump. EIB error information from the failing send is captured for diagnostic purposes. If the cause of the failure cannot be independently determined, contact IBM Technical Support. You will be asked for the transaction dump and associated output. You will also be asked for data server output.
SP14	An LU6.2 RECEIVE operation completed normally. The data received is not the expected argument data buffer. The format and content of the data is unknown. No further processing can be performed. This abend was issued to generate a transaction dump. You must request CLOSE to terminate the communication session and release the CICS resources. If the reason incorrect data was received cannot be independently determined, contact IBM Technical Support. IBM Technical Support will ask you for the transaction dump and associated output, including data server output.

Table 46. CACSP62 abend codes (continued)

Code	Description
SP15	An LU6.2 RECEIVE operation completed normally. The argument data buffer received was not identified correctly. Either the buffer storage was corrupted or a data server did not create this buffer. The format and content of the data are suspect. No further processing can be performed. This abend is issued to generate a transaction dump. You must request CLOSE to terminate the communication session and release the CICS resources. If the reason the buffer is incorrectly identified cannot be independently determined, contact IBM Technical Support. IBM Technical Support will ask you for the transaction dump and associated output, including data server output.
SP16	An LU6.2 RECEIVE operation completed normally. The argument data buffer is not compatible with the connection handler transaction program that issued this abend. No further processing can be performed. This abend is issued to generate a transaction dump. You must request CLOSE to terminate the communication session and release the CICS resources. If you have recently upgraded your product suite, verify all components have been correctly installed. If the reason the buffer is incompatible cannot be independently determined, contact IBM Technical Support. IBM Technical Support will ask you for the transaction dump and associated output, including data server output.

CA-Datcom interface for stored procedures

The CACTDCOM interface enables you to perform CA-Datcom database operations within the local data server address space. The interface provides equivalent functionality to batch programs that access or update CA-Datcom systems.

The User Requirements Table (URT) name is available from the RUN OPTIONS statement in your stored procedure definition rather than from the LOADNAM= parameter in the DBURINF macro. (The interface macro DBURINF is not included when generating a User Requirements Table.) The stored procedure application program requirement to open and close the URT is the same as coding the OPEN=USER parameter in the DBURINF macro for your batch application.

The CACTDCOM interface requires that the CA-Datcom access service is active on the data server. CACTDCOM interfaces with the CA-Datcom access service to connect to and communicate with the CA-Datcom Multi-User Facility, which enables CACTDCOM to log errors. If the CA-Datcom access service trace level is set to a value less than three, additional diagnostic information is also available when an error occurs.

The CACTDCOM interface module allows you to open your URT from within the data server address space and issue CA-Datcom calls to the database. Before exiting your stored procedure application, you must call the CACTDCOM interface module one last time to close the URT.

By default, when you close the URT, any changes that your stored procedure application program made are committed to the database. Otherwise, the stored procedure should issue a ROLLBACK call before closing the URT.

Specifying CA-Datcom resource information

When you provide CA-Datcom resource information by defining a User Requirements Table (URT), you enable your stored procedure to communicate effectively with CA-Datcom. A URT provides security by restricting database access. A URT is also needed for efficient allocation of CA-Datcom resources.

When writing your own stored procedure to communicate with CA-Datcom, you should specify the User Requirements Table name using the RUN OPTIONS parameter instead of hard coding it in your stored procedure application program. Doing so allows you to simply drop the stored procedure definition (DROP PROCEDURE), update the stored procedure definition and then re-catalog the stored procedure definition (CREATE PROCEDURE) when your environment changes, without requiring any changes to your stored procedure application program.

You can also supply the User Requirements Table name programmatically. The sample stored procedure for CA-Datcom demonstrates how this can be done.

CA-Datcom resource information is identified by the `_DATACOM` keyword in the RUN OPTIONS parameter. The complete format of CA-Datcom resource information entry is:

```
_DATACOM(urt-name)
```

The CA-Datcom resource information entry is separated from preceding keyword entries by a comma. Order of the keyword entries is not mandated. The following examples show RUN OPTIONS statements with keyword entries specified in differing order. Regardless of the order, the resultant stored procedure processing is identical.

Example 1:

```
RUN OPTIONS '_DATACOM(urt-name),_CICS(transaction-scheduling-info)'
```

```
RUN OPTIONS '_CICS(transaction-scheduling-info),_DATACOM(urt-name)'
```

Example 2:

```
RUN OPTIONS '_DATACOM(urt-name),_CICS(transaction-scheduling-info)'
```

```
RUN OPTIONS '_CICS(transaction-scheduling-info),_DATACOM(urt-name)'
```

Stored procedure and CA-Datcom communication

The CACTDCOM interface load module communicates with the CA-Datcom access service to connect with CA-Datcom for opening and closing your URT and issuing your application calls.

The following figure shows the processing flow when your stored procedure application program needs to access CA-Datcom data.

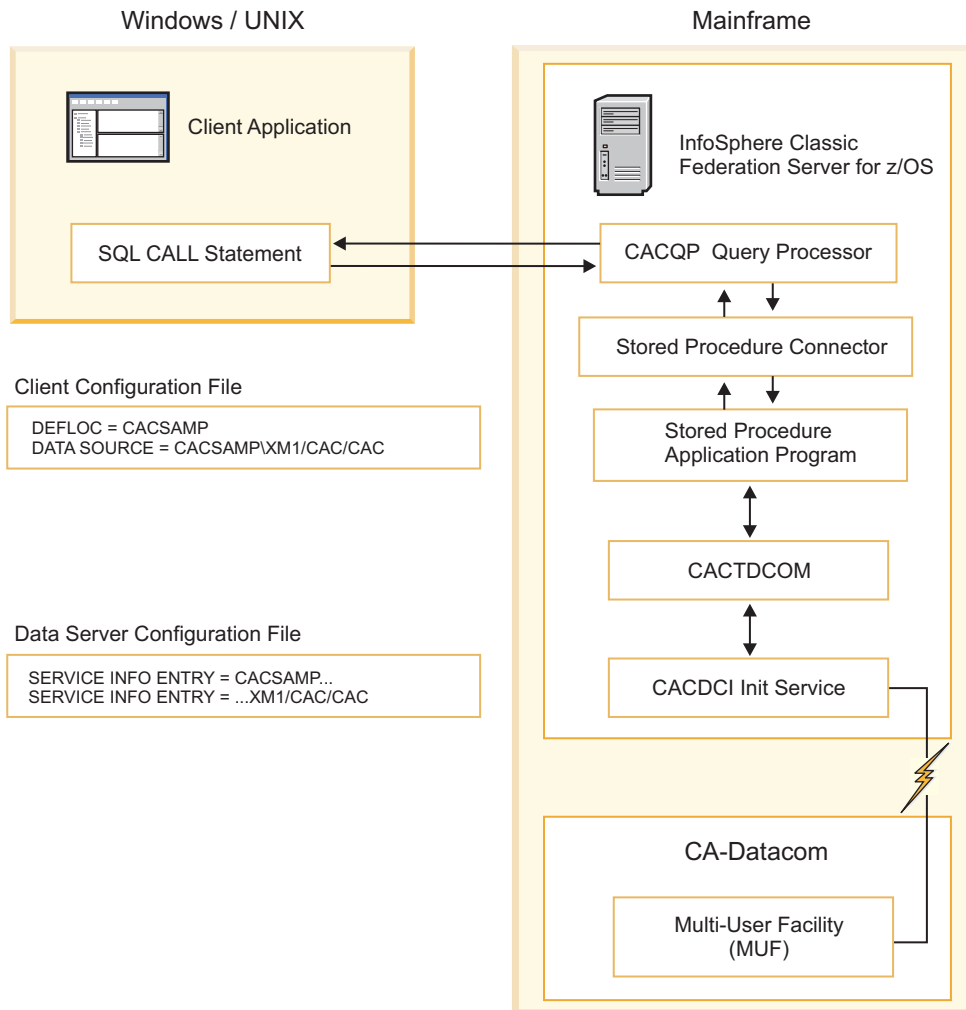


Figure 13. CA-Datacom processing flow

To make using the CACTDCOM interface module easier, the CA-Datacom call formats are identical to the call syntax used by native CA-Datacom. An exception is that the first parameter in the parameter list passed to the CACTDCOM interface module must be the address of the SQLDA parameter passed to your stored procedure application program by the stored procedure data connector. Calls to CA-Datacom are performed as if the program were written for direct access to CA-Datacom.

For example, processing can occur as follows:

- To open a URT you must provide:
 - A properly formatted User Information Block (UIB)
 - A Request Area containing the command OPEN
- To begin reading records using Set At A Time commands, you must provide:
 - A properly formatted User Information Block (UIB)
 - A Request Area containing the command SELFR along with the table name and DBID if required
 - A work area to receive the retrieved data
 - An element list describing the data to be retrieved

- A Request Qualification Area containing selection criteria and other parameters
- To close a URT you must provide:
 - A properly formatted User Information Block (UIB)
 - A Request Area containing the command CLOSE

Success or failure of every command is returned in the Request Area Return Code and Internal Return Code.

CACTDCOM interface

You can use the supplied CACTDCOM interface with your stored procedures to perform CA-Datcom database operations within the local server's address space.

The CACTDCOM interface is distributed as a separate load module that you can dynamically call or statically link into your stored procedure.

You call the CACTDCOM interface by using a parameter list. The first parameter that you pass is the SQLDA argument that you pass to the stored procedure. The remaining entries in the parameter list are the normal parameters you use to access CA-Datcom data. Each database operation requires from two to five parameters.

For a description of the required parameters, an explanation of how parameters are used, and what information they contain for the call or upon return, see the CA-Datcom documentation about commands.

You must call CACTDCOM by using variable-argument list calling conventions. That is, you must set the high-order bit of the last parameter to one. Variable-argument list is the standard calling convention for most high-level languages, including COBOL. CACTDCOM is linked as an AMODE(31) RMODE(ANY) application. You pass all addresses to CACTDCOM in 31-bit addressing mode.

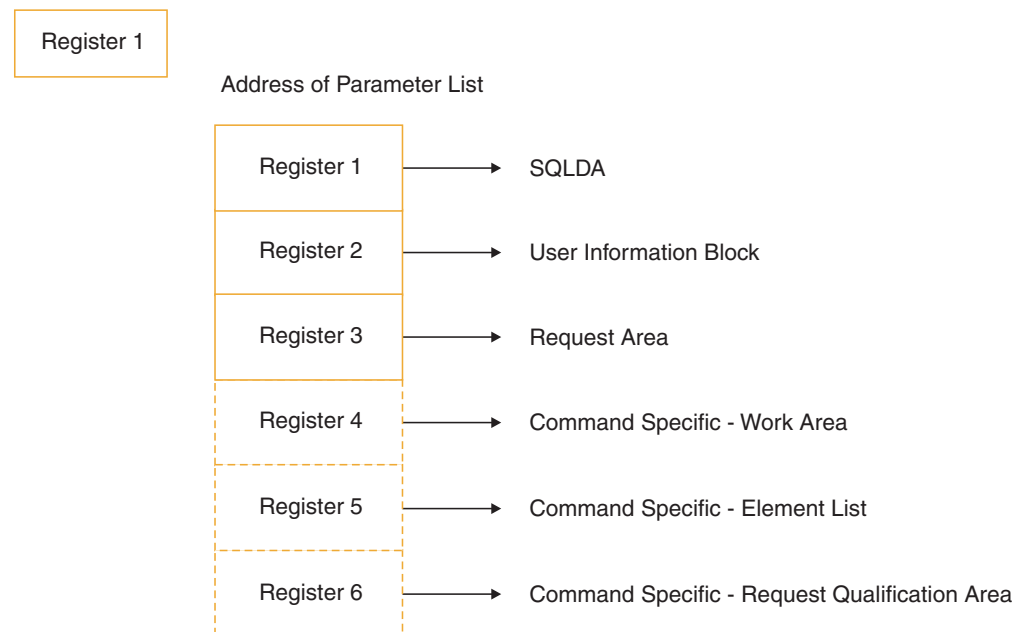


Figure 14. Parameters passed to CACTDCOM

As indicated by the diagram above, parameters 4, 5, and 6 are conditional, based upon the type of CA-Datcom command specified in the Request Area.

The first call that any stored procedure issues must be to OPEN the User Requirements Table. Opening the URT enables you to access any table the URT specifies, and you can also update those tables defined with UPDATE=YES. The following code example shows how to issue the OPEN command in a COBOL stored procedure. Included below is the portion of the stored procedures DATA DIVISION and the LINKAGE SECTION that is referenced in the executable code and the executable code to prepare for and issue the OPEN command.

All code examples shown here are hard-coded to process the table **DEMO-DEM-POH**. The CA-Datcom installation process provides this table. In this example, the stored procedure opens the User Requirements Table that you specified in the RUN OPTIONS of the cataloged stored procedure, because you supply no override `_DATACOM` keyword.

```
WORKING-STORAGE SECTION.
01  USER-INFO-BLOCK                PIC X(32).

01  POH-REQUEST-AREA.
    02  POH-REQ-COMMAND             PIC X(5).
    02  POH-REQ-TABLE-NAME         PIC X(3) VALUE "POH".
    02  FILLER                     PIC X(5).
    02  POH-REQ-RETURN-CODE        PIC X(2).
    02  POH-REQ-INTRNL-RTNCD       PIC X(1).
    :  :  :  :  :  :  :  :
    :  :  :  :  :  :  :  :
```

```
LINKAGE SECTION.
01  SQLDA-DATA.
    05  ARG-SQLDAID                PIC X(8).
    05  ARG-SQLDABC                PIC 9(8) COMP.
    05  ARG-SQLN                   PIC 9(4) COMP.
    :  :  :  :  :  :  :  :
```

```
* BUILD REQUEST AREA AND OPEN URT
9300-OPEN-URT.
```

```
    MOVE "MYPROGID"                TO USER-INFO-BLOCK.
    MOVE "OPEN"                    TO POH-REQ-COMMAND.
    MOVE SPACES                    TO POH-REQ-RETURN-CODE.
    MOVE 0                          TO POH-REQ-INTRNL-RTNCD.
```

```
* IF YOUR URT CONTAINS MULTIPLE "POH" TABLES, SUPPLY THE
* DATABASE ID WHERE TABLE "DEMO-DEM-POH" IS INSTALLED.
* THIS IS NORMALLY DATABASE 1 (DBID=00001).
* MOVE 1                          TO POH-REQ-DATABASE-ID.

* TO OVERRIDE THE URT IN THE METADATA "RUN OPTIONS",
* SUPPLY A URT NAME IN THE WORK AREA AND PASS THE WORK AREA
* ADDRESS AS THE FOURTH PARAMETER ON THE CALL THAT FOLLOWS.
* MOVE "_DATACOM(URTNAME)"        TO POH-WORK-AREA.
```

```
DISPLAY "CACSPDC1 CALLING CACTDCOM TO OPEN URT."
    UPON CONSOLE.
CALL "CACTDCOM" USING SQLDA-DATA,
                    USER-INFO-BLOCK,
                    POH-REQUEST-AREA.
IF POH-REQ-RETURN-CODE NOT EQUAL SPACES
    PERFORM 9900-DISPLAY-RC
END-IF.
```

```
IF RETURN-CODE NOT EQUAL ZEROS
```

```

        DISPLAY "CACSPDC1 OPEN URT ERROR. RC=" RETURN-CODE
        UPON CONSOLE
        GOBACK
    END-IF.

```

After the URT opens, the stored procedure can issue any CA-Datcom command against the database. The following sample code shows how to issue the **ADDIT** command for the table **DEMO-DEM-POH** in a COBOL stored procedure. The code sample shows a portion of the stored procedure's DATA DIVISION, the LINKAGE SECTION that is referenced in the executable code, and the executable code to prepare and issue the ADDIT command:

```

WORKING-STORAGE SECTION.
01  USER-INFO-BLOCK                PIC X(32).

01  POH-REQUEST-AREA.
    02  POH-REQ-COMMAND              PIC X(5).
    02  POH-REQ-TABLE-NAME          PIC X(3) VALUE "POH".
    02  FILLER                      PIC X(5).
    02  POH-REQ-RETURN-CODE        PIC X(2).
    02  POH-REQ-INTRNL-RTNCD       PIC X(1).
    . . . . .
    . . . . .

01  POH-WORK-AREA                  PIC X(20).
01  POH-RECORD REDEFINES POH-WORK-AREA.
    02  POH-PO                      PIC X(5).
    02  POH-LI                      PIC X(3).
    02  POH-RECORD-UPDATE.
        03  POH-VENDR                PIC X(3).
        03  POH-TPVAL                PIC X(8).
        03  POH-LATE                 PIC X.

01  POH-ELEMENT-LIST.
    02  POH-ELM1                    PIC X(5) VALUE "PO".
    02  POH-SEC-CD1                 PIC X  VALUE " ".
    02  POH-ELM2                    PIC X(5) VALUE "LI".
    02  POH-SEC-CD2                 PIC X  VALUE " ".
    02  POH-ELEMENT-LIST-UPDATE.
        03  POH-ELM3                 PIC X(5) VALUE "VENDR".
        03  POH-SEC-CD3              PIC X  VALUE " ".
        03  POH-ELM4                 PIC X(5) VALUE "TPVAL".
        03  POH-SEC-CD4              PIC X  VALUE " ".
        03  POH-ELM5                 PIC X(5) VALUE "LATE".
        03  POH-SEC-CD5              PIC X  VALUE " ".
        03  END-OF-ELEMENTS          PIC X(5) VALUE SPACES.

LINKAGE SECTION.
01  SQLDA-DATA.
    05  ARG-SQLDAID                  PIC X(8).
    05  ARG-SQLDABC                  PIC 9(8) COMP.
    05  ARG-SQLN                     PIC 9(4) COMP.
    . . . . .
    . . . . .

* BUILD CONTROL BLOCKS AND ISSUE ADDIT COMMAND TO INSERT RECORD
2200-ISSUE-ADDIT.

        MOVE "ADDIT"                TO POH-REQ-COMMAND.
        MOVE SPACES                  TO POH-REQ-RETURN-CODE.
        MOVE 0                        TO POH-REQ-INTRNL-RTNCD.
        DISPLAY "CACSPDC1 CALLING CACTDCOM TO ADDIT." UPON CONSOLE.
        CALL "CACTDCOM" USING SQLDA-DATA,
                                USER-INFO-BLOCK,
                                POH-REQUEST-AREA,
                                POH-RECORD,

```

```

                                POH-ELEMENT-LIST.
IF POH-REQ-RETURN-CODE NOT EQUAL SPACES
    PERFORM 9900-DISPLAY-RC
END-IF.

IF RETURN-CODE NOT EQUAL ZEROS
    PERFORM 10000-ERROR-CLOSE-URT
    DISPLAY "CACSPDC1 ADDIT ERROR. RC=" RETURN-CODE
    UPON CONSOLE
    GOBACK
END-IF.

```

After the database is modified, issue a COMMIT command. If the database modification is determined to be unsuccessful or incorrect, issue a ROLBK command to reverse all database modifications done after the last COMMIT or ROLBK command. If the stored procedure has issued neither COMMIT nor ROLBK, all database modifications done after the OPEN command are reversed. Issuing a CLOSE command executes an implied COMMIT, so the actual COMMIT command can be bypassed. Follow your site standards and procedures regarding the use of explicit or implicit COMMIT processing.

When the processing completes, the last call that the stored procedure issues must be to CLOSE the User Requirements Table. The following code sample shows how to issue the CLOSE command in a COBOL stored procedure. The sample shows a portion of the stored procedure's DATA DIVISION, the LINKAGE SECTION that is referenced in the executable code, and the executable code that prepares and issues the CLOSE command:

```

WORKING-STORAGE SECTION.
01  USER-INFO-BLOCK                PIC X(32).

01  POH-REQUEST-AREA.
    02  POH-REQ-COMMAND              PIC X(5).
    02  POH-REQ-TABLE-NAME          PIC X(3) VALUE "POH".
    02  FILLER                      PIC X(5).
    02  POH-REQ-RETURN-CODE        PIC X(2).
    02  POH-REQ-INTRNL-RTNCD       PIC X(1).
    . . . . .
    . . . . .

LINKAGE SECTION.
02  SQLDA-DATA.
    05  ARG-SQLDAID                PIC X(8).
    05  ARG-SQLDABC                PIC 9(8) COMP.
    05  ARG-SQLN                   PIC 9(4) COMP.
    . . . . .
    . . . . .

* BUILD REQUEST AREA AND CLOSE URT
9500-CLOSE-URT.

    MOVE "CLOSE"                   TO POH-REQ-COMMAND.
    MOVE SPACES                     TO POH-REQ-RETURN-CODE.
    MOVE 0                           TO POH-REQ-INTRNL-RTNCD.
* NO NEED FOR URT NAME HERE. IT IS REMEMBERED FROM THE OPEN.

    DISPLAY "CACSPDC1 CALLING CACTDCOM TO CLOSE URT."
    UPON CONSOLE.
    CALL "CACTDCOM" USING SQLDA-DATA,
                                USER-INFO-BLOCK,
                                POH-REQUEST-AREA.
IF POH-REQ-RETURN-CODE NOT EQUAL SPACES
    PERFORM 9900-DISPLAY-RC
END-IF.

```

```

IF RETURN-CODE NOT EQUAL ZEROS
  DISPLAY "CACSPDC1 CLOSE URT ERROR. RC=" RETURN-CODE
  UPON CONSOLE
  GOBACK
END-IF.

```

Compiling and linking applications that use CACTDCOM

Your stored procedure is compiled (or assembled) by using your site standard procedures.

Include the following control statement in your link edit input stream to link the CACTDCOM interface by using your site standard procedures:

```
INCLUDE LOAD(CACTDCOM)
```

where *LOAD* is the name of a DD statement supplied in your link edit JCL to identify the location of the CACTDCOM load module. Typically, this DD statement identifies your installation load library.

Sample JCL *USERHLQ.SCACSAMP(CACSPCCD)* is provided for compiling and linking a stored procedure that uses the CACTDCOM interface.

CACTDCOM return codes

CACTDCOM returns return code 0 in normal situations. If CACTDCOM detects an error, a non-zero return code is returned. Database errors that CA-Datcom reports are returned directly to the stored procedure application in the Request Area and are not intercepted or interpreted by the CACTDCOM interface.

The following table identifies the possible return code values and information you can find in the output log for the server when one of these errors is detected.

Recommendation: The CACTDCOM interface uses return code values between 101 and 199. To eliminate any confusion regarding the source of an error code, the stored procedure can use return codes starting at the number 200.

Table 47. CACTDCOM return code information

Value	Description	Log information
101	An invalid number of parameters were passed to CACTDCOM. Additional calls can be issued to CACTDCOM. This error should occur during development of the stored procedure.	Message 5701813 (0x005700B5) is generated.
102	The first parameter passed to CACTDCOM cannot be identified as an SQLDA or no parameter was passed to CACTDCOM. Additional calls to CACTDCOM should not be issued. This error should occur only during development of the stored procedure.	None. The information necessary to issue log calls is not available.
103	The identifier in the internal control block used to manage stored procedure processing by the CACTDCOM interface has been corrupted. Additional calls to CACTDCOM should not be issued. This is an internal error. Contact IBM Technical Support.	None. The information necessary to issue log calls is not available.
104	The length in the internal control block used to manage stored procedure processing by the CACTDCOM interface is corrupted. Additional calls to CACTDCOM should not be issued. This is an internal error. Contact IBM Technical Support.	None. The information necessary to issue log calls is not available.

Table 47. CACTDCOM return code information (continued)

Value	Description	Log information
105	The buffer version in the internal control block that is used to manage stored procedure processing by the CACTDCOM interface is not correct. Additional calls to CACTDCOM should not be issued. This error indicates that stored procedures of a prior release are installed with the current release of the product. Any prior stored procedure applications must be re-linked by using the new CACTDCOM interface.	None. The information necessary to issue log calls is not available.
106	CACTDCOM environment not properly initialized. Either the Stored Procedure Data Connector control block pointer is zero or the CA-Datcom Service anchor pointer is corrupted. Additional calls to CACTDCOM should not be issued. This is an internal error. Contact IBM Technical Support.	1. None. The information necessary to issue log calls is not available if the control block pointer is 0 or 2. Message 5701890 (0x00570102) is generated if the anchor pointer is corrupted.
107	CACTDCOM environment not initialized. Either the global system anchor pointer is zero or the CA-Datcom Service anchor pointer is zero. Additional calls to CACTDCOM should not be issued. This is an internal error. Contact IBM Technical Support.	1. None. The information necessary to issue log calls is not available if the global system anchor pointer is zero, or 2. Message 5701889 (0x00570101) is generated if the CA-Datcom Service anchor pointer is zero.
108	The stored procedure is attempting to issue a second OPEN command. Only one User Requirements Table can be open at a time. Additional calls can be issued to CACTDCOM. This error should occur only during development of the stored procedure.	Message 5701816 (0x005700B8) is generated.
109	CACTDCOM was unable to allocate memory for internal control blocks. Additional calls to CACTDCOM should not be issued. This error should only occur when the server is not configured properly.	Message 5701793 (0x005700A1) is generated.
110	No User Requirements Table name was provided to CACTDCOM. Either the RUN OPTIONS statement from the catalog or the programmatic override did not contain the keyword _DATACOM, or the _DATACOM keyword was not followed immediately by a (urtname) clause. Additional calls to CACTDCOM should not be issued. This error should occur only during development of the stored procedure.	Message 5701814 (0x005700B6) is generated.
111	The User Requirements Table name provided to CACTDCOM was less than one character or greater than eight characters in length. Additional calls to CACTDCOM should not be issued. This error should occur only during development of the stored procedure.	Message 5701815 (0x005700B7) is generated.
112	The User Requirements Table program could not be loaded. Additional calls to CACTDCOM should not be issued. The URT program must be in a load library that is included in the STEPLIB concatenation of the server.	Message 5701817 (0x005700B9) is generated.
113	The User Requirements Table program that was loaded does not appear to be a known format. Additional calls to CACTDCOM should not be issued. A portion of the URT is dumped in binary to the server log when the trace level is set to 4 or less. Review the URT content and determine if it is valid. Changes in the format of User Requirement Tables require code changes in CACTDCOM. Contact IBM Technical Support if you believe that the CACTDCOM interface requires code changes.	Message 5701818 (0x005700BA) is generated.

Table 47. CACTDCOM return code information (continued)

Value	Description	Log information
114	An attempt to connect with the CA-Datcom access service has failed. One of the error messages shown in the column to the right has been logged. Additional calls to CACTDCOM should not be issued. This probably indicates the CA-Datcom access service module is not active.	1. Message 5701897 (0x00570109) is generated 2. Message 5701903 (0x0057010F) is generated 3. Message 5701905 (0x00570111) is generated
115	An error has occurred while attempting to send an OPEN command to CA-Datcom. Additional calls to CACTDCOM should not be issued. The User Information Block (UIB) and the Request Area (RA) passed to CACTDCOM are dumped in binary to the server log when the trace level is set to 2 or less. Review the control blocks and determine if they contain valid data. Depending upon the return code information that is logged, this might be an internal error that should not occur, or it might be an error that can occur only during development of the stored procedure.	Raw system return code information is logged.
116	Resources required to communicate with CA-Datcom are not available. All retries have been exhausted. Additional calls to CACTDCOM should not be issued. The CA-Datcom control blocks passed to CACTDCOM are dumped in binary to the server log when the trace level is set to 2 or less. This is an internal error. Contact IBM Technical Support.	Message 5701819 (0x005700BB) is generated
117	The stored procedure has called CACTDCOM with a database command without first OPENing the URT. Additional calls can be issued to CACTDCOM. This error should occur only during development of the stored procedure.	Message 5701820 (0x005700BC) is generated
118	A CACTDCOM interface environment pointer is zero. Additional calls to CACTDCOM should not be issued. This is an internal error. Contact IBM Technical Support.	Message 5701821 (0x005700BD) is generated
119	An error has occurred while attempting to send a CLOSE command to CA-Datcom. Additional calls to CACTDCOM should not be issued. The User Information Block (UIB) and the Request Area (RA) passed to CACTDCOM are dumped in binary to the server log when the trace level is set to 2 or less. Review the control blocks and determine if they contain valid data. Depending upon the return code information that is logged, this might be an internal error that should not occur, or it might be an error that can occur only during development of the stored procedure.	Raw system return code information is logged.
120	The CACTDCOM interface was processing a command other than OPEN or CLOSE when it received a signal from the query processor to immediately terminate processing. The query processor is probably being stopped.	Message 5701894 (0x00570106) is generated
121	An error has occurred while attempting to send a command other than OPEN or CLOSE to CA-Datcom. Additional calls to CACTDCOM can be attempted to COMMIT or ROLBK as required by the stored procedure application program. Success of any subsequent call is dependent upon the type of error that occurred previously. The CA-Datcom control blocks passed to CACTDCOM are dumped in binary to the server log when the trace level is set to 2 or less. This is an internal error. Contact IBM Technical Support.	Raw system return code information is logged.
122	The User Requirements Table contains at least one table enabled for update processing. When the OPEN command was sent by the stored procedure application program, the Internal Return Code field in the Request Area was coded with the letter N, indicating no update processing was to be allowed. The stored procedure has sent an ADDIT, DELET or UPDAT command which has been rejected. Additional calls can be issued to CACTDCOM. This error should occur only during development of the stored procedure.	Message 5701822 (0x005700BE) is generated

IMS interface for stored procedures

The CACTDRA interface enables your stored procedures to perform database operations on IMS databases.

Stored procedures cannot use DL/I calls to access or update IMS data. A stored procedure does not have addressability to a PSB and the list of PCBs that a PSB contains.

The CACTDRA interface allows you to schedule a PSB and returns a pointer to the list of PCBs in the PSB. You can then establish addressability to one or more of these PCBs and issue ISRT, GU, GHU, and REPL DL/I calls against the PCB. Before exiting your stored procedure you call CACTDRA one last time to unschedule the PSB.

By default, when you unschedule the PSB any changes your stored procedure application program made are committed to IMS.

Recommendation: You should not issue a ROLLBACK call from your stored procedure unless it is necessary. Issuing a ROLLBACK causes an abend in the data server.

The CACTDRA interface uses the IMS DRA initialization service to request PSB scheduling, issue DL/I calls and finally unschedule the PSB. The IMS DRA initialization service must be active within the data server. The service allows CACTDRA to log errors and, if the trace level for the IMS DRA initialization service is set to a value less than three, to log the DL/I calls that are issued by the stored procedure that calls CACTDRA.

The following figure shows the processing flow when your stored procedure application program needs to access IMS data.

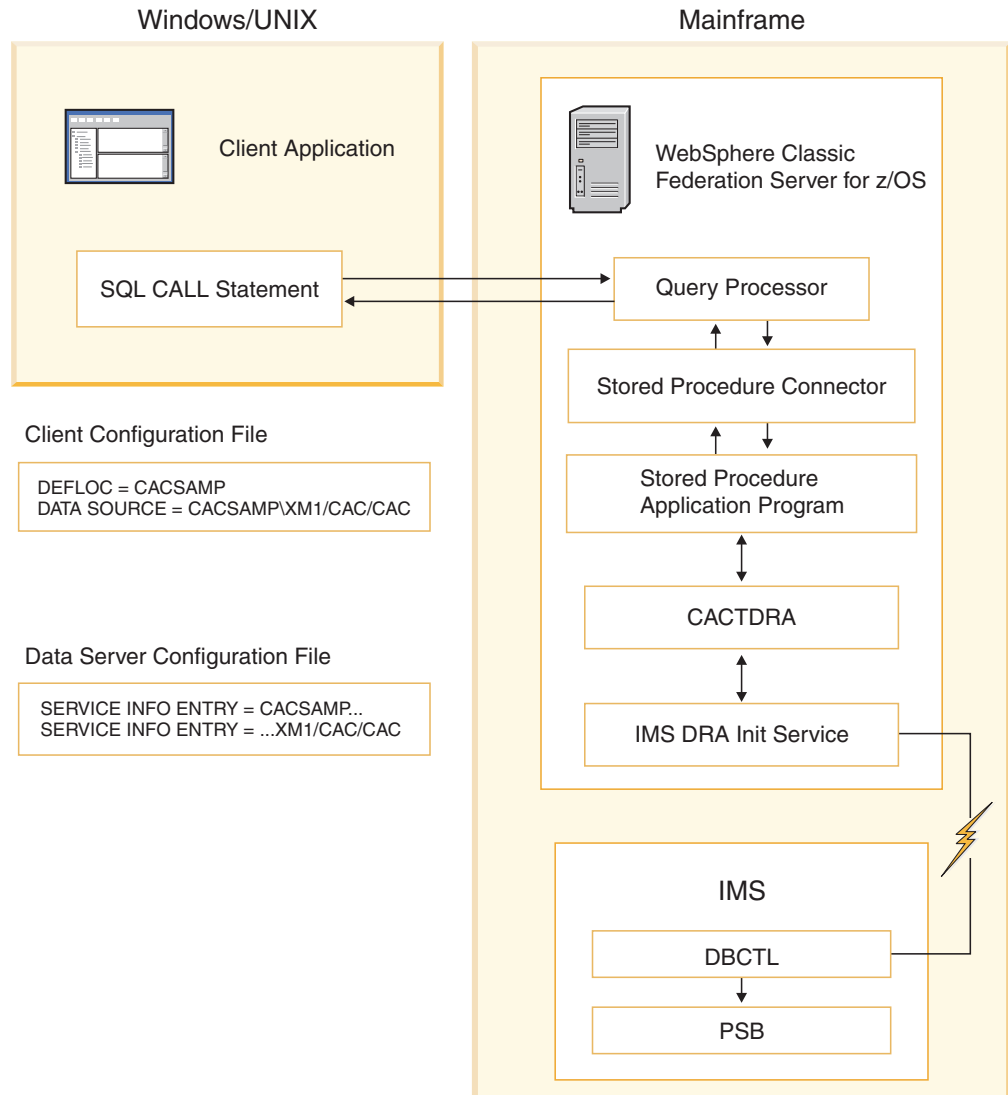


Figure 15. IMS DRA processing flow

To make using the CACTDRA easier, the DL/I call formats are identical to CBLTDLI call syntax with the exception that the first parameter in the parameter list must be the address of the SQLDA parameter passed to your stored procedure. Additionally, CACTDRA uses two other DL/I function codes:

- SCHED: Identifies the name of a PSB to be scheduled.
- TERM: Unchedules the PSB.

CACTDRA interface

The CACTDRA interface enables you to perform IMS database operations within the local server address space. The functionality of this interface is equivalent to using DL/I functions.

The CACTDRA interface is distributed as a separate load module that you can dynamically call or statically link into your stored procedure.

The CACTDRA interface uses calling conventions that resemble the way a standard program accesses IMS data. The primary difference is that the first parameter that is passed to CACTDRA must be the SQLDA argument list that is passed to the stored procedure.

The CACTDRA interface must be called using variable-argument list calling conventions. The high-order bit of the last parameter must be set to 1. Variable-argument list is the standard calling convention for most high-level languages, for example COBOL. Additionally, CACTDRA is linked as an AMODE(31) RMODE(ANY) application. All addresses passed to the CACTDRA interface must be passed in 31-bit addressing mode.

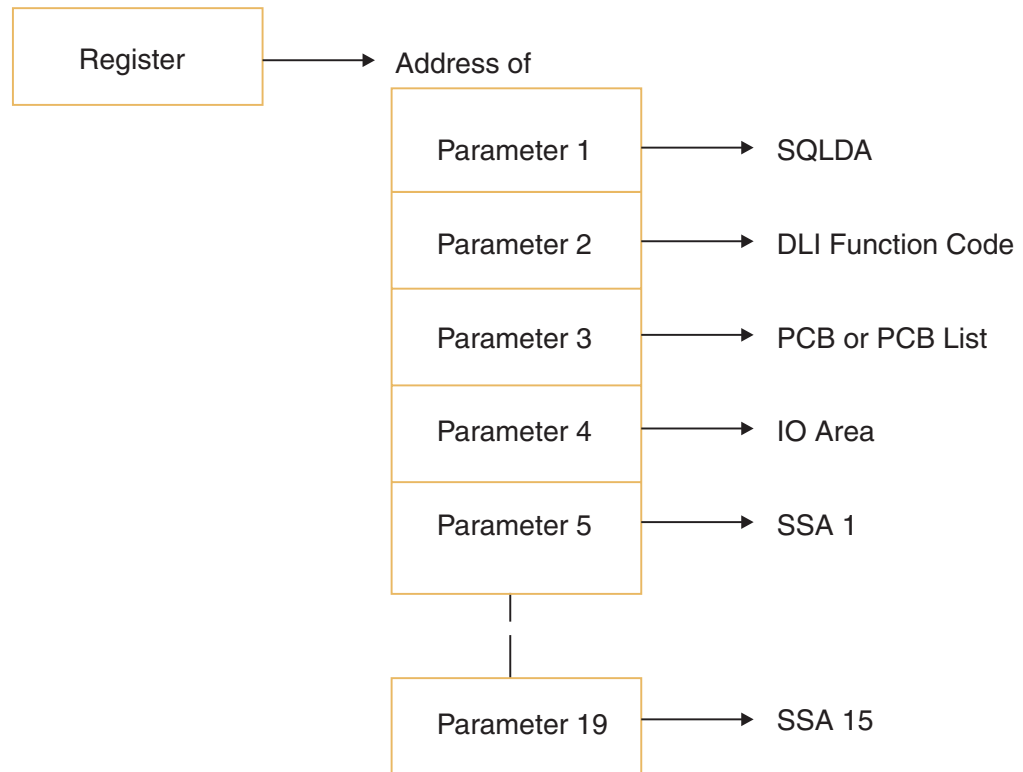


Figure 16. Parameters passed to CACTDRA

The actual number of parameters that need to be passed to the CACTDRA interface depends on the type of DL/I function being issued and the structure of the database being accessed. The first call that is issued to the CACTDRA interface is a CICS-like SCHED call that schedules a PSB. This call differs slightly from a standard DL/I call because:

- The name of the PSB is passed in the I/O area
- The CACTDRA interface returns the address of the list of PCBs that are defined within the PSB.

One of these PCBs is passed on subsequent calls to the CACTDRA interface to access or update IMS data. You can also pass the I/O PCB to issue checkpoint or rollback calls.

The following example shows how to issue the SCHED call in a COBOL stored procedure. The example shows the stored procedure's DATA DIVISION, part of the

LINKAGE section, and how to prepare for and issue the SCHED call. In the following examples, the SQLDA is named ARG-DATA.

```

WORKING-STORAGE SECTION.
01 PSB-NAME          PIC X(8)          PROCEDURE DIVISION
    VALUE "DFSSAM09".                USING ARG-DATA, ARG-SQLCA.
01 DLI-FUNC-CODE     PIC X(4).         MOVE "SCHED" TO DLI-FUNC-CODE.
01 DRA-PCB-LIST      POINTER.          CALL "CACTDRA" USING ARG-DATA,
01 DRA-PCB-LIST      POINTER.                DLI FUNC CODE,
    10 SSA-SEG-NAME   PIC X(9)          DRA-PCB-LIST
    VALUE "PARTROOT".                PSB-NAME.
01 IO-AREA.          IF RETURN-CODE NOT EQUAL ZEROS
    10 PART-KEY       PIC X(17).        ERROR OCCURED
    10 DESCRIPT       PIC X(20).        GOBACK
    10 FILLER         PIC X(13).        ELSE
LINKAGE SECTION.     SET ADDRESS OF PCB-LIST
COPY CACSPSDA.        TO DRA-PCB-LIST
COPY CACSPSCA.        SET ADDRESS OF RET-PCB
01 PCB-LIST.          TO DB-PCB1.
    05 IO-PCB         POINTER.
    05 DB-PCB1        POINTER.
    05 DB-PCB2        POINTER.
01 RET-PCB.
    10 PCB-DBD        PIC X(8).
    10 PCB-SEG-LVL    PIC 99.
    10 PCB-STATUS-CODE PIC XX.
    10 PCB-PROCOPT    PIC X(4).
    10 FILLER         PIC X(4).
    10 PCB-SEG-NAME   PIC X(8).
    10 PCB-KFBA-LEN   PIC 9(8) COMP.
    10 PCB-SENSEGS    PIC 9(8) COMP.
    10 PCB-KFBA       PIC X(33).

```

In the preceding example, the DFSSAM09 IMS sample PSB is scheduled. This PSB allows updates on all segments of the DI21PART IMS sample database. After the PSB is successfully scheduled, the stored procedure can obtain addressability to one or more of the PCBs in the PSB and issue standard DL/I calls against any of the PCBs that are available.

The following sample is a singlet of code from a COBOL program that shows how to insert a new root segment in the DI21PART database in a COBOL program.

Like normal DL/I calls, up to 15 SSAs can be passed to CACTDRA. In the same manner, DL/I calls must be issued in the proper sequence. For example, to update a segment a GHU call must be issued first, followed by a REPL call.

After all of the access and update DL/I calls are issued, a CHKP call is not required. When the PSB is unscheduled, a CHKP call is automatically issued. If the stored procedure decides that any updates should not be applied a ROLLBACK call can be issued. The stored procedure should not issue a ROLLBACK call that causes an abend. If such a ROLLBACK call is issued, the query processor task servicing the stored procedure becomes unusable.

The following sample code is for an ISRT call:

```

    SET ADDRESS OF RET-PCB TO DB-PCB1.

    MOVE SPACES TO IO-AREA.
*   GET ADDRESSABILITY TO SQLDA PARAMETERS AND INITIALIZE
*   IO-AREA FOR INSERT
    MOVE "ISRT" TO DLI-FUNC-CODE.

    CALL "CACTDRA" USING ARG-DATA,
        DLI-FUNC-CODE,

```

```

                                RET-PCB,
                                IO-AREA,
                                SSA.
IF RETURN-CODE NOT EQUAL ZEROS
    MOVE RETURN-CODE TO ORIG-RC
    ERROR HAS OCCURED
ELSE
    IF PCB-STATUS-CODE NOT = SPACES
        ERROR REPORTED BY IMS.

```

After the final DL/I call is issued, the CACTDRA interface must be called a final time to unschedule the PSB by using a CICS-like TERM call. The following sample code shows how to unschedule the DFSSAM09 PSB in a COBOL program:

```

MOVE "TERM" TO DLI-FUNC-CODE.
CALL "CACTDRA" USING ARG-DATA,
                    DLI-FUNC-CODE.
IF RETURN-CODE NOT EQUAL ZEROS
    DISPLAY "TERM CALL RC " RETURN-CODE.
GOBACK.

```

Compiling and linking applications that use CACTDRA

To compile and link your stored procedure to use the CACTDRA interface, modify the CICS CACSPCCC sample to compile and link JCL so that the ICCS CACSPCCC sample includes the CACTDRA interface instead of CACSPBR.

Provide configuration information in the LE/compile section of the customization parameters file. This information is used to generate the JCL to compile and link a CICS stored procedure.

Compile and link CACSPCOM using the supplied member CACSPCCC in the SCACSKEL library. (Sample JCL to compile and link a stored procedure that uses the CACTDRA interface is not supplied.)

Before submitting the job, modify the statement:

```
INCLUDE LOAD(CACSPBR)
```

to:

```
INCLUDE LOAD(CACTDRA)
```

If your stored procedure needs to both access and update IMS data and invoke a CICS application program, include both of the above statements in the link step.

CACTDRA return codes

CACTDRA returns a zero return code in normal situations. If CACTDRA detects an error, or an error is reported by DRA, a non-zero return code is returned.

CACTDRA issues 11 error return codes. Return code values between 1 and 100 are reserved for use by the CACTDRA interface. Application return codes must be higher than the reserved error return codes.

The application should also check DL/I status codes in addition to any errors that might be reported by the standard DL/I status codes placed in an IMS PCB.

The following table identifies the return code values and the information you can find in the output log for the data server when the system detects an error.

Table 48. CACTDRA return code information

Value	Description	Log information
1	An invalid number of parameters were passed to CACTDRA. You can issue additional calls to CACTDRA. This error should only be received during development of the stored procedure.	Message 5701825 (0x005700C1) is generated.
2	CACTDRA was unable to allocate memory for internal control blocks. Additional calls to CACTDRA should not be issued. This error should be received only when the data server is not properly configured.	Message 5701793 (0x005700A1) is generated.
3	IMS DRA initialization service not active. Additional calls to CACTDRA should not be issued. This error should be received only when the data server is not properly configured.	Message 5701717 (0x00570055) is generated.
4	CACTDRA environment not properly initialized. Additional calls to CACTDRA should not be issued. This is an internal error that should never occur.	None. The information necessary to issue log calls is not available.
5	The requested PSB could not be scheduled. The stored procedure application program can try to schedule another PSB. This error can occur during development of the stored procedure, in production situations when the data server has not been properly configured (for example, not enough DRA threads), and when the IMS Stage 1 gen. was not set up properly for the scheduled PSB.	Message 5701708 (0x0057004C) is generated.
6	DL/I call failed. You can issue additional calls to CACTDRA. This error should be received only during development of the stored procedure.	Message 5701715 (0x00570053) is generated.
7	An error occurred during TERM processing. Additional calls to CACTDRA should not be issued. This error should generally only be received if something has happened to IMS.	Message 5701711 (0x0057004F) is generated.
8	The first parameter passed to CACTDRA is not the SQLDA or the SQLDA has been corrupted. Additional calls to CACTDRA should not be issued. This error should only be received during development of the stored procedure application program.	None. The information necessary to issue log calls is not available.
9	The SQLDA is corrupted. Additional calls to CACTDRA should not be issued. This is an internal error that you should never encounter.	None. The information necessary to issue log calls is not available.
10	The application issued a SCHD call, but a PSB had already been scheduled. This error should be received only during development of the stored procedure.	Message 5701826 (0x005700C2) is generated.
11	The application issued a standard DL/I or a TERM call, but no PSB was scheduled. This error should be received only during development of the stored procedure.	Message 5701827 (0x005700C3) is generated.

Invoking existing IMS transactions from a stored procedure

To execute existing IMS transactions from a stored procedure, the stored procedure uses an APPC/MVS interface instead of the CACTDRA interface.

APPC/MVS communicates with APPC/IMS. APPC/IMS schedules the requested IMS transaction and returns the output messages that the IMS transaction generates and sends to the calling APPC/MVS application program (the stored procedure).

APPC/IMS overview:

The APPC/IMS interface communicates with Standard DL/I applications, modified standard DL/I applications, and CPI communication-driven applications.

The IMS Transaction Manager is normally implemented as an IMS DB/DC subsystem, but it can be implemented as an IMS DC subsystem.

APPC/IMS supports interfacing with the following types of IMS application programs:

Standard DL/I applications

Existing IMS applications that are unaware that they are not communicating with an LU 2 terminal. APPC/IMS converts the APPC data streams into the appropriate input messages, sends them to the IMS transaction, waits for an output message from the IMS application, and sends the output message to the invoking application as an LU 6.2 data stream. In most instances, the existing IMS transaction requires no modifications.

Modified standard DL/I applications

Existing IMS application that have been modified to issue CPI Communications calls, as well as normal DL/I calls that the application initially issued.

CPI communication-driven applications

IMS applications that use CPI communications calls to communicate with the partner program. They participate in the two-phase commit process by issuing SSRMIT or SSRBACK CPI calls. These types of applications can issue database-related DL/I calls to access and update full-function, DEDB, MSDB, and DB2 databases.

For information about how these types of application are designed, see the IMS documentation about application programming. For information about message flows and using sync-point conversations, see LU 6.2 documentation about partner program design.

APPC/MVS overview: The APPC/MVS interface is an extension to APPC/VTAM that allows MVS/ESA applications to use the full capabilities of LU 6.2. APPC/MVS provides a set of high-level language callable services that allows applications that use these APIs to communicate with other applications through the communications protocols provided by the SNA network.

The APPC/MVS API combines several CPI calls into single APPC/MVS API calls, and allows for state transitions that normally require individual CPI calls. APPC/MVS allows applications to be abstracted from the network definitions by using such things as symbolic destination names and selection of default outbound LUs.

Depending on the version of APPC/MVS installed, some of these APIs have different names and parameter lists. For information about writing APPC/MVS applications and the APPC/MVS APIs, see the z/OS documentation about APPC/MVS.

Configuring APPC/IMS and APPC/MVS:

When you set up an environment for a stored procedure you need to configure changes to IMS, create VTAM definitions, and install and configure APPC/MVS.

For information about installing and configuring APPC/MVS, see the z/OS documentation about APPC/MVS.

Application design requirements:

When you design your stored procedure application to invoke existing IMS transactions, define input and output message formats to enable your stored procedure to exchange messages with IMS.

To design a stored procedure that invokes an existing IMS transaction, you do not need to know the business logic that is implemented by the IMS transaction, but you do need to know the message flow and input and output message formats that the IMS transaction expects to receive.

Typically, the IMS transaction uses Message Format Service (MFS), so you should use the message input descriptor (MID) and message output descriptor (MOD) definitions to determine the input and output message formats. If the IMS transaction is implemented in COBOL, the COBOL program contains copy books or data structures that the IMS transaction uses.

If you use MFS MID and MOD definitions to identify the input and output messages formats that the transaction uses, you need to define a parameter for each unique input and output field defined in the MID and the MOD. The parameter length is for the length of the MID or MOD field, and the data type is usually CHAR. Use the following guidelines to determine what type of parameter to use in the CREATE PROCEDURE definitions:

- Identify fields that only appear in the MID as INPUT parameters.
- Identify fields that only appear in the MOD as OUTPUT parameters.
- Identify fields that occur in both the MID and the MOD as INOUT parameters.

When you develop stored procedures, consider that the input and output message formats that are sent to APPC/IMS do not exactly match the formats that the IMS transaction is expecting. The standard format for an IMS input message is:

- LL: Length of the message, including LL and ZZ
- ZZ: Zeros
- Data: The input fields that the transactions expect

APPC communications use unmapped conversations, so the send message format is:

- APPC LL
- IMS LL
- Data

The output messages generated by an IMS transaction have the same format as the input messages. However, APPC/IMS strips off the LL and ZZ fields, so these do not need to be defined in the received (output message) definition—just the APPC LL field.

Restriction: APPC/IMS automatically inserts the IMS transaction code at the beginning of the data portion of the message. As a result, invoking IMS transactions that do not expect the transaction code at the beginning of the input message require modifications to the existing IMS transaction. This problem should not occur if the transaction design specifies that all input messages begin with the 8-byte transaction code.

Stored procedure limitations for IMS transactions:

When you develop a stored procedure application that invokes an IMS conversational transaction, you need to define the conversation ID in hexadecimal format.

You can develop a stored procedure that invokes an IMS conversational transaction that returns multiple screens of data. When you allocate a conversation, an 8-byte conversation ID is returned. You must use this ID in subsequent APPC/MVS calls. The conversation ID is associated with the address space and can be used by any TCB in the address space until the system deallocates the conversation.

Therefore, if you add an additional input or output parameter to the stored procedure definition for a given conversation ID, it is possible to write a stored procedure that is called multiple times, each time returning one or more screens of data. If you take this approach, define the input or output conversation ID parameter as CHAR(16). Then the system converts the conversation ID to hexadecimal format for data exchanges with the client application.

Recommendation: Because stored procedures can only return a limited amount of information, create stored procedures by invoking IMS transactions to perform updates or to return a single screen of data.

Testing APPC/MVS stored procedures:

When you test your APPC/MVS stored procedure, you can activate IMS tracing and get trace output to verify that you are formatting input messages correctly.

About this task

Testing an APPC/MVS stored procedure is not difficult because APPC/MVS returns useful information from the APPC/MVS Error_Extract API. The API includes the message text that is usually displayed on the console, including error message numbers, which help you diagnose and debug APPC/MVS-related errors.

APPC/MVS supports very good tracing and debugging facilities, which enable you to get message flows and contents to help you determine whether the input messages are formatted correctly.

Recommendation: Tracing the actual APPC/MVS message flows and content is generally not necessary, unless the stored procedure is attempting to interface with a very complicated IMS transaction.

An alternate approach is to use the IMS trace facilities. With this approach, IMS logs the DL/I calls that are generated by the transaction. By inspecting the SSAs that are generated and the data returned, you can determine whether the input messages are formatted correctly. However, you need some knowledge about the IMS transaction to determine whether the call patterns and returned information is correct.

To activate IMS tracing and get trace output:

Procedure

1. Activate IMS tracing by issuing the command:
`/TRACE SET ON PSB (PSB-Name)`

where *PSB-Name* is the name of the PSB associated with the IMS transaction being tested.

2. For each test run, perform the following steps:
 - a. Run a client application that invokes the stored procedure.
 - b. After the stored procedure completes, issue the IMS /CHECKPOINT command to flush the IMS log buffers to disk.
 - c. Run the IMS trace log print utility DFSERA10. Reference the online logs for your IMS system.
 - d. Review the output from DFSERA10 to see if the DL/I calls look appropriate.

Sample stored procedures for IMS transactions:

You can use stored procedure samples to test conversational and non-conversational transactions with IMS.

The following sample stored procedures interface with two of the IMS IVP sample transactions. The stored procedures are located in the SCACSAMP library:

- CACSPTNO—Interfaces with the IMS IVTNO non-conversational transaction
- CACSPTCV—Interfaces with the IMS IVTCV conversational transaction

In addition, member CACSPMPP contains the CREATE PROCEDURE syntax for these two sample stored procedures.

The sample IMS transactions are very simple, and provide QUERY, INSERT, UPDATE and DELETE functions to a sample employee database. IVTNO expects a single message with a command code, and the database is immediately updated and a single message is returned. IVTCV performs the same operation, but the changes are not committed until you send an END message to IVTCV.

The sample stored procedures are simplified, in that they do not use security or sync-point control. The processing flow for the CACSPTNO stored procedure is as follows:

1. Establish addressability to the input/output parameters it was passed
2. Allocate the conversation using hard-coded values.
3. Format the input message buffer.
4. Send the message to IMS.
5. Wait for the output message from IMS.
6. Update any output or input/output parameters.
7. De-allocate the conversation.

The CACSPTCV stored procedure is similar to the CACSPTNO stored procedure, and includes additional send and receive steps to send the **END** message and wait for the response.

Both samples contain limited error detection and reporting. If an error is reported, the APPC/MVS Error_Extract service (API) is called to obtain detailed information about the error condition. The error information is displayed on the console and a general error return code is returned to the calling application. This approach was taken because after the stored procedure is tested, errors should not occur in production operations. Another reason is that end-users probably will not know what to do with the detailed information that is provided when an APPC/MVS error is reported.

Adding transaction security:

You can use supplied subroutines to secure the sample stored procedures that test IMS transactions.

The sample stored procedures that test IMS transactions are unsecured transactions, and do not pass a user ID or password in the allocate call. To use secured transactions, the stored procedure can use the CACSPGUI (Get User ID) and CACSPGPW (Get Password) subroutines to obtain the ID and password of the current user for inclusion in the allocate call.

Sync-point conversations:

You can issue calls that use LU 6.2 sync-point protocols to communicate with APPC/IMS.

These calls are less complex than the ones that the sample stored procedures contain to allocate the conversation, send and receive messages, de-allocate the conversation, and obtain error information. For information about sync-point control flows and the formats of the calls that you need to issue, see the IMS documentation about application programming design.

Chapter 5. Tuning

Table definition techniques and query writing techniques can help ensure optimal performance.

When you map data, the indexes, keys, and columns for the database or file system must be defined to the data server. All other optimization built into the data server depends on these mappings.

Performance varies based on the type of data accessed and how the data is accessed. Query optimization also varies based on how the data servers and client applications are tuned. Another important factor is the service definition for the query processor.

Query optimization techniques

The query processor optimizes queries that are written on the databases. However, this optimization is limited to the information that the query processor knows about in the databases and the organization of the databases.

Write effective queries and define logical tables to maximize query performance. Query optimization is based on the extent to which the database or file system performs the filtering that is required to obtain the final result set.

Example: A three-segment IMS HIDAM DBD contains 10,000 instances of the lowest level segment (also called the *leaf segment*).

- If the query processor can build a *segment search argument (SSA)*, which contains a search argument for every segment, a single access is required. In this case, the query processor retrieves the final result set, and the connector or the query processor does not need to perform additional filtering.
- If the query processor cannot build an SSA, 10,000 IMS GET commands are issued. In this case, the connector or the query processor must filter the intermediate result set to obtain a single row result set.

The full retrieval of the mapped segments (or an entire VSAM file) is called a *full table scan*. The IMS retrieval of the single row is faster if the SSA contains primary or secondary index fields. The scenario is the same for VSAM access that involves primary and alternate indexes rather than SSAs.

If the TRACELEVEL configuration parameter for the query processor service is set to 2, the filtering information that obtains the final result set is written to the log. For IMS, you use the TRACELEVEL configuration parameter for the IMS service.

Keys to optimize queries

Use keys in your queries when possible to optimize query performance.

To identify keys, include index definitions on logical tables. When you define index information, front-end tools access the data server to create optimized queries. These queries are based on the SYSIBM.SYSINDEXES and SYSIBM.SYSKEYS information in the metadata catalogs. These index definitions do not actually index the data. Instead, the definitions describe the existing physical indexes on the data.

The entire logical table is scanned in these instances:

- Queries without qualifying WHERE clause information on indexes
- Queries without qualifying information on non-indexed columns

To perform these scans, the query processor must read the entire portion of the database or file that the table defines. The scanning process can result in poor performance on large databases, particularly when an SQL join is performed and an inner table must be scanned multiple times.

When a new application uses existing data, add additional indexes to your data to meet the needs of Classic federation queries. Review your queries to determine whether new indexes can improve performance without impact to another application that needs to use the data.

Join operations to optimize queries

Join processing uses the nested-loop access method for all queries that contain joined columns.

Nested-loop processing for a two-table join involves reading an outer table and, for each row selected in that table, reading the inner table to join with that row, based on the join columns.

Join processing applies to every row of the inner table where the value of its join columns satisfies the specified relational conditions with the correlating join columns of the selected row of the outer table. This processing involves these actions:

- Creates a row in the result table with the requested columns of both the inner and outer tables.
- Selects the next row of the outer table and repeats this process until there are no more rows to select from the outer table.

Example 1. Join query to avoid:

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

The query processor reads all rows in TABLE1 by default. The query processor then reads all rows in TABLE2 to join every row in TABLE1 with every row in TABLE2. If TABLE1 has 100 rows and TABLE2 has 100 rows, the result set contains 10,000 rows for this query.

The number of rows is calculated as follows:

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

The actual number of database reads is 10,100. 100 reads are for the outer table and 10,000 (100 x 100) reads are for the inner table. While this type of Cartesian join is not commonly performed in SQL queries, it demonstrates the type of join queries to avoid.

Example 2. Change the previous query as follows:

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

This query might return a small number of rows. However, because there are no other qualifications for TABLE1 or TABLE2, nested-loop join processing might

require reading every row from TABLE1 and matching it with every row of TABLE2 (10,100 reads). This is especially true if the join column in the inner table (COL_B1 in TABLE2) is not indexed.

If COL_B1 in TABLE2 is a uniquely indexed column, the number of necessary reads is reduced from 10,100 to 200. Each row in the outer table (TABLE1) is read once and a single indexed read is performed on the inner table (TABLE2).

Example 3. A unique index is defined on COL_B1 in TABLE2, and the outer and inner tables are switched as follows:

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

The column COL_A1 in TABLE1 is not indexed. 10,100 rows are read because the inner table, which is now TABLE1, must be scanned for each row in the outer table TABLE2. Again, COL_A1 is not an indexed column in TABLE1, whereas COL_B1 in TABLE2 is indexed.

When a query includes a join table, the query optimizer determines ordering optimization that is based on information in the WHERE clause. This optimization is attempted in phases:

1. The optimizer checks if the default outer table in the join contains any WHERE clause information that is not part of a join condition with a column in another table. If there is no non-join WHERE information, the optimizer checks the remaining tables in the query and uses the first table with non-join WHERE information as the outer table in the join.
2. The optimizer runs additional optimization if the first phase reorders the default outer-inner table processing of the join and more than two tables are referenced in the join. The optimizer attempts to order all of the inner tables of the join such that each processed inner table contains a column that is joined with a table that precedes it in the outer to inner order. Filtering of rows from outer-to-inner table processing is maximized to keep the number of access reads to a minimum.

Query processor optimization

The query processor optimizes SQL queries based on information in the WHERE clause, index information, and configuration parameters that activate optimization.

Connectors and query processor interaction

During query processing, the query processor runs SQL queries, accesses the metadata catalogs, and calls connectors.

A connector uses the information in the WHERE clause of a query to attempt to access the database or file with optimum performance. The query processor inspects the data that the connector returns to determine if the data meets the WHERE qualification, if specified. The data that remains is then staged so that query results are not returned to the client until the complete result set is built. After all the data is read and staged, the required post-processing is performed, and those results are staged. Post processing includes sorting when an ORDER BY clause is specified and when evaluating rows for aggregate functions. After the query processor finishes processing the staged results, the final result set is returned to the client application.

The connectors optimize query processor performance using index information. If the connector cannot optimize access to the database or file, the connector attempts another optimization method. The connector filters the number of records returned to the query processor by analyzing the WHERE clause on the data that is returned from the database or file system. If the connector can successfully filter the records that are returned, the connector notifies the query processor not to run the corresponding filtering instructions at the query processor level.

Configuration parameters for optimization

Use the configuration parameters to improve query processing performance.

Use the configuration parameters to activate the following optimization strategies:

- `STATICCATALOGS` for static catalog access
- `BTREEBUFFS`, `FETCHBUFSIZE`, and `TEMPFILESSPACE` for result set staging

The success of these optimization strategies depends on the query that is issued and on the size of the expected result set.

Static catalogs

You can identify the metadata catalogs that the query processor accesses as static to optimize the compiler process. You can activate static catalog processing with the `STATICCATALOGS` parameter.

Typically, the query processor opens and closes the metadata catalogs for each logical table that a query references during the compile process. Locks must be established to prevent updates when a query processor is processing a request from the Classic Data Architect or the metadata utility. A lock prevents one query processor instance from updating catalog data that another query processor instance is using. All locks are requested by a query processor.

When the catalogs are static, the metadata catalog files are opened once when you first issue a query and closed when you disconnect from the query processor. After catalogs are defined as static, updates are not permitted from another source while the data server is active.

Static catalog processing improves query performance when client applications issue several queries that return small result sets. You can also create linear versions of the metadata catalog files to optimize the compiler process.

Creating a linear catalog:

You create and access linear catalogs in memory rather than on disk. The sample JCL member, `CACLCAT`, contains JCL to allocate a linear version of the system catalog. `CACLCAT` is located in the `SCACSAMP` data set.

About this task

You can populate a linear system catalog by copying the contents from a previously populated sequential or linear catalog. The `INIT` operation of the catalog initialization and maintenance utility is not used to initialize a linear system catalog.

Procedure

1. Allocate a sequential catalog.
2. Use the `INIT` function to initialize and populate the catalog.

3. Start the data server using the sequential catalog.
4. Populate the sequential catalog with the necessary table mappings using the metadata utility or the Classic Data Architect.
5. Shut down the server.
6. Allocate the linear catalog.
7. Use the COPY function to populate the linear catalog.
8. Update the data server configuration to set `STATIC CATALOGS = 1`.
9. Start the data server using the linear catalog.

Result set staging

A set of configuration parameters help you to make choices between virtual storage and disk resources to stage data. You can tune the space for staged result sets with the `BTREEBUFFS`, `FETCHBUFFSIZE`, and `TEMPFILESPEACE` parameters.

If you do not activate immediate return of data processing, or if this processing cannot be applied to a query, the result set is staged before it is returned to the client application. Most of the columns in a result set row are written to a B-tree file. If you activate immediate return of data processing, staging occurs but the staged result set size is reduced because data is being read and deleted as new data is added.

This set of parameters control the number of B-tree buffer caches in use and the physical type, size, and temporary storage in use after all memory in the cache is exhausted. The temporary storage can be hiperspace or disk storage.

Recommendation: Use hiperspace space to avoid physical I/Os. If you configure `TEMPFILESPEACE` to use hiperspace, the number of btree buffers (in-core buffers) has less effect on optimization. When hiperspace is enabled, you should use the `BTREEBUFFS` default.

IMS access optimization

You can optimize access to IMS data with optimizing methods for queries, Program Communication Block (PCB) selection options, and Program Specification Block (PSB) scheduling.

Keyed access techniques, SSA, and IMS optimization

You can optimize native access to your IMS databases by using the keyed access techniques that IMS provides.

Optimized access to the database relies on the IMS index and Segment Search Argument (SSA). You can use primary indexes, secondary indexes, and search fields.

For optimum performance, the columns that a WHERE clause references need to supply a key value or multiple key values. Similarly, in join operations, the inner tables in a join need to include key information so that a qualified SSA call can optimize access. A qualified SSA is only generated for either condition when index information is available in the catalogs for the logical tables that an SQL statement references.

Primary indexes for IMS optimization

Define columns that map to the primary index field in an IMS database. This technique optimizes queries that contain the WHERE and JOIN qualification on those columns.

Qualified SSAs are created when a query meets the following criteria:

- The WHERE clause of the query contains the key values.
- Tables from the outer loops in a join operation supply a unique key for the subqueries in the inner loop.

Optimizing HDAM databases

Hierarchical Direct Access Method (HDAM) databases do not contain a primary index. Instead, they use a key-hashing technique to gain fast access to data. Logical tables mapped to an HDAM database might not be retrieved in ascending key sequence, because the key sequence is a function of the HDAM randomizer, which is an IMS user exit. You can order keys in an HDAM database either by specifying an ORDER BY clause in the SELECT statement or by specifying the name of a column that maps an XDFLD statement that contains the HDAM primary key as its source.

Tip: An XDFLD statement is associated with the target segment. XDFLD specifies the name of an indexed field, the name of the source segment, and the field used to create the secondary index from the source segment. You can use XDFLD statements only when a secondary index exists.

Secondary indexes for IMS optimization

When you define secondary indexes for a table with the CREATE INDEX statement, the query optimizer uses secondary indexes to optimize each query based on information in the WHERE clause.

If both a primary sequence field and a secondary index are available when the query accesses IMS data, the primary sequence field is given precedence.

Required: If you define an IMS secondary index with a CREATE INDEX statement, the PSB used to access the target table must contain one or more PCBs with the correct processing sequence (PROCSEQ). If the PSB does not contain one or more PCBs, a runtime error occurs when the query processor selects the mapped index for keyed processing.

CREATE INDEX statement for IMS optimization

You can use the CREATE INDEX statement to define IMS indexes, with certain processing requirements and restrictions.

You can use the CREATE INDEX statement to define the primary sequence field on a Hierarchical Indexed Direct Access Method (HIDAM) database. You cannot use the CREATE INDEX statement to define the sequence field on a Hierarchical Direct Access Method (HDAM) database. You can use XDFLD statements for either HIDAM or HDAM databases. Specific limitations on keyed access in HDAM can mislead optimization algorithms if the primary sequence field is defined as an index. However, HDAM keyed access is used when possible.

When you define an IMS index, the columns in the CREATE INDEX statement must match either the sequence field for a HIDAM root segment or the SRCH fields for an XDFLD statement in the DBD. The metadata utility validates all

CREATE INDEX statements on the DBD by matching column offset and length information with the offset and length information for sequence and SRCH fields within the DBD. This matching is only performed on the root segment for the defined table. The validation is based on the INDEXROOT clause in the CREATE TABLE statement. If the INDEXROOT value is not specified, the physical ROOT default is used.

The order of the columns in a CREATE INDEX statement is significant during the matching process. The validation flags as an error columns that match sequence or SRCH fields, but are not in the CREATE INDEX order.

The columns in a CREATE INDEX statement can sub-define sequence or SRCH fields in the DBD. For example, you can map a primary HIDAM sequence field defined as 8 bytes as two 4-character columns.

Search fields for IMS optimization

You can use an IMS search field to optimize access to your IMS data. If you define a search field as a column in the metadata grammar and then reference that column in a WHERE clause, a Segment Search Argument (SSA) is generated that contains the WHERE qualification for the search field.

The WHERE clause in a query needs to refer to the columns that are associated with the root segment primary index or to a secondary index that maps to the root segment of the database. When using a secondary index, the DBD hierarchy might be inverted. For optimum performance, include a WHERE qualification on subordinate segment key fields or normal search fields in the root or subordinate segments.

Include a WHERE qualification to build a single SSA. As a result, IMS returns only the segments that match the qualification in the WHERE clause, which eliminates or minimizes any query processor and connector filtering required.

Partial keys for IMS optimization

The partial key information in a WHERE clause helps optimize access to IMS data.

You can map multiple columns to an IMS field. The columns can be the primary key or a secondary index (XDFLD). When you map multiple columns to a single IMS field and a WHERE clause references a subset of those columns, the resulting condition is known as a *partial key*.

Optimization is accomplished by generating a key range based on the parts of the key in the WHERE clause. The key range specifies the lowest and highest key values of the columns in the WHERE clause. To optimize the query and generate SSAs, specify partial keys in the sequence in which they map to the IMS field.

Example: An IMS field named FIELD1 is 10 bytes long. You map three columns to the IMS field in the following sequence:

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

You then issue the following query:

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

The following SSA is generated:

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

In this example, IMS only returns the segments that match the WHERE qualification. However, if you specify a WHERE clause that only references COLUMN2 or COLUMN3, the query processor cannot generate an SSA. Instead, the query processor or connector must retrieve all of the mapped segments and perform all filtering logic.

Restriction: Multiple part field mapping applies to IMS root segment sequence fields and secondary index keys only. Partial field mapping is not recognized for ordinary IMS segment fields that are not part of a key.

Path calls for IMS optimization

Path calls access data sources when WHERE qualification criteria exists for segments at levels other than the root of the hierarchy.

Path calls can significantly reduce the number of Data Language Interface (DL/I) calls required in such cases. To use path calls, PSBs that access IMS databases must contain PCBs that are defined with a PROCOPT value that includes the value P.

Example:

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

If you are not certain whether your specific PSBs support path calls, review the PSBs in question with your IMS database administrator.

IMS optimization for HIDAM, HDAM, and DEBD

Optimizing access to IMS data involves considerations that are specific to the HIDAM, HDAM, and DEBD databases.

HDAM and HIDAM optimization

Optimizing access to IMS data with HDAM and HIDAM databases primarily focuses on DBDs. In addition, optimization differs when accessing HIDAM and HDAM databases and HDAM databases impose some limitations.

With HIDAM databases, SSAs are built when the following conditions are met:

- WHERE clauses contain columns that map to fields.
- Reads on fields contain the EQ, LT, LE, GT, or GE operators with any combination of AND and OR conditions against any combination of mapped segments in the database.

With HDAM database, queries are optimized under the following conditions:

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

HDAM restrictions

SSA support for HDAM databases is restrictive due to IMS restrictions. For optimization to occur, the WHERE clause cannot result in the request for a range of HDAM keys.

A limitation of IMS HDAM processing affects full table scans. For many HDAM queries, full table scans can occur. The default HDAM RANDOMIZER might not store keys in sequenced order. For example, a segment with a key field of 4 is not

necessarily ordered in sequence following a segment with a key field of 3. While IMS allows OR conditions on an HDAM key qualification in the SSA, IMS might not return the correct result set. IMS might not return rows that satisfy the WHERE clause.

Result set processing

Classic federation ensure that the correct result set is returned by taking one of the following actions:

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

For further information on HDAM processing, see the IMS documentation about application programming.

DEDB optimization

You can access IMS Fast Path DEDB databases by using a Bean-Managed Persistence (BMP) service or by using the Database Resource Adapter (DRA) interface.

Disabling high-speed sequential processing:

When accessing DEDBs as a Bean-Managed Persistence (BMP) service, you must disable high-speed sequential processing (HSSP) to successfully process databases.

About this task

About this task

PCBs are defined as HSSP when the PROCOPT keyword parameter includes the value H.

Procedure

To disable HSSP in a BMP:

Include a DFSCCTL JCL statement in the data server startup JCL with a SETO control statement.

Example SETO statement:

```
//DFSCCTL DD *  
SETO DB=IVPDB3,PCB=HSSP,NOPROCH
```

Failure to disable PROCOPT=H PCBs results in an FY status call when you access the DEDB, and in an unexpected DLI status error message.

Setting Fast Path buffers:

Fast Path buffers must be available to process DEDB databases.

About this task

About this task

If IMS runs out of buffers during a query, an FR status code is returned when it processes a Data Language Interface (DL/I) call. The client returns an unexpected status code error message.

Procedure

To set the Fast Path buffers:

- BMP environment: Set the NBA and OBA keyword parameters in the data server JCL to pass to IMS at BMP startup time.
- DRA environment: Define the CNBA and FPBxx parameters in the DFSPRP macro when you generate the DRA startup table that the data server uses.

PCB selection options for IMS optimization

The query processor selects a PCB to access the segments that are mapped by a logical table or column. To optimize performance, you can perform IMS PCB selection by using an explicitly named PCB or by using ordinal PCB numbers. Otherwise, the query processor uses the PCB by verification method.

The PCBPREFIX is an optional clause on the CREATE TABLE and CREATE INDEX statements that you can use to specify a set of candidate PCB names and sets and ranges of PCB numbers that are eligible to use for accessing the IMS database. If a PCBPREFIX is specified on an IMS table definition, you should also specify the PCBPREFIX clause on the CREATE INDEX statement.

You can use the Classic Data Architect to modify the PCB selection for IMS tables or indexes.

PCB selection by name

When you use PCB selection by name, you need to carefully coordinate the PCB definitions and the related table definitions.

About this task

PCB selection by name selects the PCB by using a partial or full PCB name that is associated with a logical table or index. Specifying PCB names is useful when you use the AIBTDL/I interface to select the PCB used to access the IMS database but the PCB names do not match the naming conventions required for selection using the PCBPREFIX option.

Important: Ensure that the named PCB can access the defined tables. Use the same PCB prefix name for all logical tables that access the same logical or physical DBD by using the same primary or secondary index.

If the PCB selected does not have the correct IMS path access for the table (SENSEGS), an error is returned from the query.

PCB selection by number

PCB selection by number requires knowledge about the PSB being used to access the IMS database and can be impacted by changes to the contents of a PSB.

Specifying PCB numbers is useful when you want to eliminate any IMS call before the database is accessed to process the SQL statement.

When using PCBNUM, you specify the actual number of the PCB that you want to use to access the IMS database. You can also specify ranges of PCBs that are

eligible for use and specify multiple ranges of PCB numbers for use when the same PSB is used in SQL queries that contain references to multiple tables that use the same PSB to access the IMS database. The first PCB in the PSB is numbered one.

PCB selection by verification

PCB selection by verification is the default method that the query processor uses to select a PCB for processing.

With this method, the query processor issues DL/I calls to verify that the PCB selected can successfully access the database path that the logical table will access. The correct PROCSEQ must be specified if a column that maps to an XDFLD is specified in a WHERE clause.

PSB scheduling for IMS optimization

PSBs contain PCBs. A *program specification block (PSB)* is the unit of access that a given program uses to interface with IMS. The process of interfacing with IMS is known as PSB scheduling.

You can use the ASMTDLI interface or the DRA interface for PSB scheduling. Depending on which IMS interface is used, the processes involved in scheduling a PSB for a data server and how you specify which PSB a data server uses differ.

Use the DRA interface to access IMS data when possible.

BMP and DBB interfaces for PSB scheduling

The BMP and DBB interfaces enable the data server instance to access IMS as a BMP, DBB, or DL/I region. But, this interface can require significant use of IMS resources.

Use the DRA interface to access IMS data when possible. If you cannot use the DRA interface, then configure data servers with the BMP or DBB interface as single-user servers. A smaller number of PCBs are required to be defined within the PSB. A smaller PSB requires significantly less resources for the data server instance.

The BMP and DBB interfaces limit access to IMS data to a single PSB for a data server instance. The IMS region controller only permits a single PSB to be scheduled and has a limit of one instance of an IMS region controller per z/OS address space.

The PSB scheduled in this environment must contain enough PCBs to support all concurrent user access to the IMS data if you plan on take any of the following actions:

- Access multiple DBDs
- Perform joins of IMS data
- Support multiple users

Very large PSBs require significant use of IMS resources.

DRA interface for PSB scheduling

With DRA, you can schedule multiple PSBs from a single data server. DRA also can support multiple user connections at the same time.

The ability to schedule multiple PSBs enables the PSBs to be small in size. Small PSBs reduce the IMS resources that are required for a query of IMS data.

With the CREATE TABLE statement for an IMS logical table, you can use SCHEDULEPSB parameter to associate two PSBs with each logical table. The first PSB that you specify is referred to as the standard PSB. The second PSB that you specify is the join PSB.

Standard PSB

The standard PSB is scheduled to service queries between two or more IMS logical tables that do not contain a join operation. This PSB only needs to contain a single PCB. The PCB is sensitive to the segments that make up the IMS path to which the IMS logical table maps.

Join PSB

The join PSB is scheduled for queries where the query contains a join between two or more IMS logical tables. A join PSB is specified for the first table in the join. The join PSB typically contains multiple PCBs, at least one for each IMS logical table that the join references.

Before subsequent PSBs are scheduled, the currently scheduled PSBs are inspected to determine if they contain a PCB that can be used to service the query.

- If a PCB is available, another PSB is not scheduled. Instead, the available PCB issues IMS calls for access to the referenced table.
- If a PCB is not available and a join PSB is specified for this table, that join PSB is scheduled.
- If a PCB is not available and a join PSB is not specified, the standard PSB that is associated with that logical table is scheduled.

Default PSB

You can use the DEFAULTPSBNAME configuration parameter to specify a default PSB. If a standard PSB is not defined for an IMS logical table, the default PSB is scheduled. The default PSB is inspected to determine if a PCB exists that can be used to service the query. If a default PSB is scheduled, it contains PCBs for all IMS databases, segment paths, and secondary indexes that need to be accessed. The default PSB for the DRA interface is similar to the PSB for the BMP and DBB interfaces.

To set up the DRA interface:

- Create standard PSBs for all IMS logical tables if you plan to use the DRA interface and join IMS data with composite join PSBs. The need to schedule multiple PSBs for join operations is eliminated.
- You do not need to create individual PSBs for each logical table. You can share the same standard PSB or join PSB among multiple logical tables. In addition, if you use multiple logical tables to access multiple paths in the same IMS database, you can create a single PSB that contains one or more PCBs that map all paths in the DBD to be accessed. You can then associate this PSB with all of the logical tables that are mapped to that database. Similarly, share the PSB for all of the logical tables that map to a DBD that uses the same secondary index.

VSAM access optimization

You can optimize access to VSAM data by using keyed access techniques, configuration parameters, and the VSAM service.

The techniques that you can use to optimize access to VSAM data only apply to VSAM KSDS data sets.

With ESDS and RRDS data sets, the entire contents of the files must be read to process a query. You cannot access the data directly, with the exception of accessing an ESDS data set through an alternate index.

Keyed access techniques for VSAM optimization

You can use a primary index, an alternate index, or partial keys to optimize access to VSAM data.

Primary and alternate indexes for VSAM optimization

The query processor supports both primary and alternate indexes when it processes SQL requests of VSAM data. You can use the Classic Data Architect to define a primary VSAM index or an alternate VSAM index.

Although these indexes are usually transparent, the performance of queries that use primary or alternate indexes improves when you retrieve data from large VSAM files. Performance particularly improves during join processing.

Primary indexes:

The query processor determines the use of primary indexes at run time.

The query processor identifies primary indexes that are based on the following criteria:

- Existence of index information that defines the columns that contain the key
- Key values that are supplied in the SQL WHERE clause

The query processor automatically performs keyed reads of the data set, if possible.

Alternate indexes:

The query processor can use alternate indexes. The use of an alternate index refers to ESDS or KSDS alternate indexes.

An alternate index is used to satisfy an SQL query if the following conditions are met:

- A column in the WHERE clause maps to an alternate index field.
- The data server can access the VSAM alternate index path data set or DD name that was supplied in the CREATE INDEX definition.

Partial keys for VSAM optimization

When you map multiple columns to an index, and a WHERE clause references a subset of those columns, the resulting condition is known as a *partial key*. The query processor attempts to optimize VSAM access by using the partial key information supplied in the WHERE clause.

The query processor generates a key range that is based on the parts of the key in the WHERE clause. The key range specifies the lowest and the highest key values based on the values for those columns in the WHERE clause. The query processor can only perform this optimization when the columns of the partial key are specified in the sequence in which they are mapped to the index.

Example: A primary index is 10 bytes long. You map three columns to the index in the following sequence:

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

You then issue the following query:

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

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

Assume that you specify a WHERE clause that only references COLUMN2 or COLUMN3, or both, and does not reference COLUMN1. In this case, a key range cannot be generated. Instead, the entire VSAM file must be processed sequentially, and any record filtering is performed within the connector or the query processor.

Configuration parameters for VSAM optimization

You can use the VSAMAMPARMS parameter to optimize access to your VSAM files.

With VSAMAMPARMS, you can specify the number of in-core buffers to cache VSAM index and data components and the overall amount of memory that VSAM can use for buffer caching.

If a large number of records will be read, increase the size of the VSAM data buffer pool to substantially improve query performance because more of the VSAM data can be cached in core.

VSAM service for optimization

The VSAM service enables multiple concurrent users to share opened files with either the local VSAM or CICS VSAM.

This service reduces overhead because files are opened and closed less often. The VSAM service is particularly useful in join situations where a VSAM file is joined to itself.

Data server optimization

You can improve query performance with the dispatching priority of the data server and the Workload Manager (WLM) exit.

The dispatching priority at which the data server runs can have a strong affect on query performance. In addition, you can use the WLM system exit to place individual queries in WLM goal mode to control query resources.

Dispatching priority for query optimization

The dispatching priority set for a data server can significantly improve performance.

The data server runs most efficiently at lower dispatching priorities. In general, you can follow these guidelines to set the dispatching priority:

- For the data server, set the dispatching priority greater than the TCP/IP subsystems that communicate with remote clients.
- Set the dispatching priority at or above the dispatching priorities of the CICS, IMS, or DB2 subsystems that run at your z/OS site.

Consult your data center administrator to select an appropriate dispatching priority that does not adversely affect the other applications that run at your z/OS site.

WLM exit for query optimization

You can use the goal mode and the service class of the Workload Manager (WLM) exit to improve query performance.

With the sample WLM exit, you can place individual queries in WLM goal mode. In goal mode, WLM controls the amount of resources that are available for the query to use.

With the WLM exit, you can also use service classes. Each service class can have different rules for the amount of resources that z/OS assigns to run the query. With a service class, z/OS applies period switching rules for each query that is run. Typically, in goal mode with period switching, the longer a query runs the less resources it uses.

Small queries

You can set up a service class for small queries that need to run quickly and therefore be given more resources.

Longer running queries

You can assign different DATASOURCE names to longer running queries. You can also assign these queries to a different service class. With different service classes, z/OS reduces the amount of resources that these queries use as they run longer, without impacts to the small high-performance queries.

Chapter 6. Integration with IBM InfoSphere Information Server

IBM InfoSphere Classic Federation Server for z/OS integrates with InfoSphere Information Server. The Classic federation extract programs enables integration of Classic federated data with InfoSphere DataStage. From InfoSphere DataStage you can connect to the Classic data server for direct access to z/OS files.

Classic federation and InfoSphere DataStage integration

Classic federation provides an extract program, `cacfx.exe`, that enables integration of Classic federation data with IBM InfoSphere DataStage.

The `cacfx` program is a Classic federation Call Level Interface (CLI) client that runs in any environment that supports the Classic federation ODBC/CLI driver. The extract program uses the column definitions that are defined in Classic federation table mappings. The definitions are imported into DataStage as table definitions that can be loaded into the External Source stage for reference purposes.

You can run the `cacfx` program in data extract mode or in schema generation mode.

Data extract mode

The `cacfx` program issues a native query to a Classic federation server and streams the data to the standard output (`stdout`) file. This file is then processed directly by the External Source stage of DataStage.

Schema generation mode

The client program connects to a Classic federation server and queries the system catalog to generate a schema description for a specific catalog table. Schema information is written to the `stdout` file in the orchestrate schema syntax that DataStage uses. You can place the schema information into a file and import the file into DataStage separately to use for reference while designing DataStage jobs.

Configuring the CLI client for the `cacfx` program

CLI client configuration is required to use the `cacfx` extract program.

You can use the `cacfx` extract program in any environment that supports both the Classic federation ODBC/CLI driver and IBM InfoSphere DataStage.

Configuration setup is required for CLI clients on supported operating systems. The supported operating systems include AIX, HP/UX, Solaris, Microsoft Windows, and Linux on IBM System z. To configure a CLI client, follow the setup steps described for the CLI client on the appropriate operating system:

- CLI client for UNIX, Linux, and Windows
- CLI client for native z/OS
- CLI client for UNIX System Services (USS)

As a CLI client, the `cacfx` program does not require an ODBC driver manager. Configuration setup is not required for ODBC.

Designing a InfoSphere DataStage job that uses the cacfx program

You use the External Source stage to design a IBM InfoSphere DataStage job that calls the cacfx extract program to generate a schema.

Before you begin

Before you use the cacfx program, you must map data by using the Classic Data Architect. Table mappings are required to define the column mappings that are used in InfoSphere DataStage jobs.

About this task

You can use the cacfx extract program in any environment that supports both the Classic federation ODBC/CLI driver and InfoSphere DataStage.

The cacfx program is required in both the Windows DataStage designer environment and the DataStage server environment.

- In the designer environment, the cacfx program generates orchestrate schema definitions for import purposes.
- In the server environment, the cacfx program streams data into the external source stage when the InfoSphere DataStage job runs.

Restriction: The IBM z/OS floating point data type is supported on the IBM AIX, UNIX System Services (USS), and Linux on IBM System z operating systems. In other environments, you must use the Classic federation stage of InfoSphere DataStage for tables that contain floating point columns.

Procedure

1. In the DataStage Administrator, add the environment variable CAC_CONFIG to point to the cac.ini file as shown in the following example: C:\Program Files\IBM\ISClassic113\ODBC\lib\cac.ini.
2. Generate a schema definition to a file. When you run the cacfx program to generate the schema, direct the output of the program to a file on the local file system to use when importing into the InfoSphere DataStage Designer client.
3. Import the schema definition into InfoSphere DataStage. Use the InfoSphere DataStage Designer client to import the table definition. Select the **Import > Table Definitions > Orchestrate Schema Definitions > Import Orchestrate Schema** menu options to choose the file that you want to import.
4. Build and test the InfoSphere DataStage job. Choose External Source as the initial stage in the InfoSphere Classic Federation Server for z/OS job.
 - Select cacfx as the program to call from the External Source stage by using the -output data option.
 - Select **Specific Program** from the Source Method dropdown list.
 - Load the table definition that was imported from generating the schema as output to the External Source stage.
 - Define record-level formatting options when copying an imported schema into the External Source stage by selecting the **Format** tab. When you define columns to the external store stage, describe the record formatting options of those columns. These are the same columns that you originally mapped by using the Classic Data Architect.

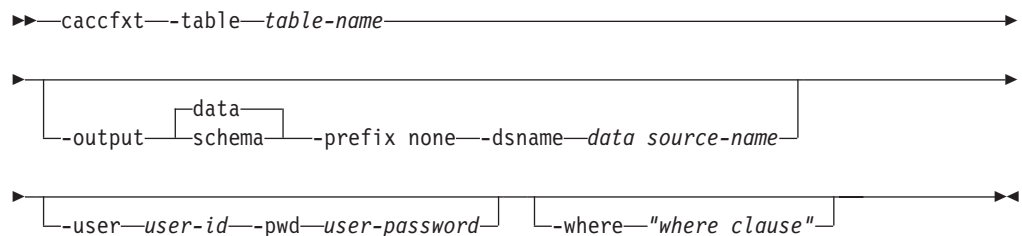
The formatting options include:

- Record level:
 - Final delimiter = none
 - Record prefix = 2
- Field defaults:
 - Delimiter = none
 - Quote = none
- General Type defaults:
 - Character set = EBCDIC
 - Byte order = native-endian
 - Data format = binary
- Remove position information in downstream stages when copying column information to avoid any problems. Position information is always generated because sparse mapping (that is, mappings that can leave out data elements) of mainframe records is supported. Position information ensures that the InfoSphere DataStage import process manages the data correctly in the External Source stage. In some cases, removing column position information might require a Copy or Transform stage.

Classic federation extract program: caccfxt

The caccfxt program is a command line program that extracts data from Classic federation into InfoSphere DataStage.

Syntax



Parameters

table *table-name*

Defines a table or view name to process in data or schema output mode. The table parameter is required.

The format of the table parameter is single token with no embedded spaces that contains a two-part name separated by the period (.) character.

output [**data** | **schema**]

Controls the mode of operation for the caccfxt program. The output parameter is optional.

In data mode, the result set for the query is streamed to the standard output (stdout) file for InfoSphere DataStage processing. The data is streamed in binary mode. Each record contains a two-byte prefix in native-endian format, unless you specify the `prefix none` parameter.

In schema mode, the caccfxt program generates an orchestrate schema definition for the table that is identified by the table parameter. The output is

streamed to a file on the local file system and is imported separately into InfoSphere DataStage for reference during the design of a InfoSphere DataStage job.

prefix none

Removes all record prefixes from the mainframe data streamed to the stdout file.

The caccfxt program places a two-byte prefix in native-endian format in each row of data that is returned in the result set for a table extract. This prefix is created from the length field that is returned in RAW_DATA() queries for mainframe information.

In some cases, this prefix can complicate working with the data and is not necessary for delineation between records in a mainframe database or file. This is particularly true if the mainframe records are fixed length. By including the prefix none parameter, all record prefixes will be removed from the mainframe data streamed to the stdout file.

dsname *data-source-name*

Identifies which data source to use in the configuration file when connecting to the data server.

As a CLI client, the caccfxt program is configured by using a separate configuration file that can contain more than one data source definition for different data servers in an installation.

user *user-id*

Identifies the user name to send to the data server. The user parameter accepts a single token.

pwd *password*

Identifies the user password to send to the data server. The password parameter accepts a single token. To ensure security, store this value as an encrypted job variable in InfoSphere DataStage jobs.

where *"where clause"*

Defines an optional where clause for filtering extracted information from the mainframe table or view. This parameter consists of column names and comparison operators that you can append to the where keyword in the query.

Example

To generate a schema definition for the table named IMSADMIN.EMPLOYEES and filter the output, you can issue the following command:

```
-table IMSADMIN.EMPLOYEES -output schema -where "BRANCH_OFFICE = '305'  
AND SALARY > 50000"
```

InfoSphere DataStage and z/OS file access

The z/OS File stage enables direct access from InfoSphere DataStage to the Classic data server to access z/OS files.

z/OS File stage

Use the z/OS File stage to identify and define file-level information about a z/OS file.

About this task

The z/OS File stage within InfoSphere DataStage works with IBM InfoSphere Classic Federation Server for z/OS to provide direct access to z/OS sequential files and VSAM files from all operating systems that InfoSphere DataStage supports.

IBM InfoSphere Classic Federation Server for z/OS reads records from and writes records to the z/OS file. TCP/IP is used to pass blocks of records back and forth between IBM InfoSphere DataStage and Classic federation data server.

Procedure

Configure the Classic federation data server environment. If you are configuring a new Classic federation environment, you need to customize the data server. See *Installing a new Classic federation data server*.

What to do next

For detailed information about z/OS File stage, see the IBM InfoSphere Information Server documentation.

Log messages for file access service requests

When you connect from InfoSphere DataStage to the Classic data server to access z/OS sequential files, error information is communicated in the InfoSphere DataStage job log and in Classic system log messages.

When a file access request does not complete successfully, the set of related messages that the z/OS allocation services produce are included in the InfoSphere DataStage job log and included in the Classic system log. The number of messages that the z/OS allocation services produce for a given request varies.

z/OS error messages are written to the system log in the form of trace records.

Examples

The following examples show messages in the Classic system log and possible messages in the InfoSphere DataStage log. The log includes the following messages:

- x00DA0201 – The log includes one occurrence of this message that includes the following information:
 - Parameters string presented to the z/OS allocation service
 - Error information feedback items that the z/OS allocation service returns. These are the S99ERROR, S99INFO, and S99ERSN items in the message.
- x00DA0214 – The log includes one or more occurrence of this message for each allocation services message that z/OS returns.

Example 1: Results of attempting to allocate a data set that is not cataloged.

InfoSphere DataStage log messages:

```
[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0201.  
ALLOC error, rc=4, reason=5896(0x00001708), info=2(0x00000002)
```

```
[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0214.  
IKJ56228I DATA SET USER.STORPROC.XCL NOT IN CATALOG OR CATALOG CAN NOT BE ACCESSED
```

Classic system log messages:

```
2009/09/30 14:50:10:0964 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(51), Task(9217672)
QP, func dynAlloc, line 335 in //DSN:USER.TEMP.SOURCE(AARCSPDA)
'osdynalloc() string:
'retddn=?,retdsn=?,retvolume=?,reason=?,InfoReason=?,smsreason=?,
'emsgp=?,msgcount=?,dsn=USER.STORPROC.XCL,disp=SHR
'-----
'S99ERROR=0x00001708 S99INFO=0x00000002 S99ERSN=0x00000FD6

2009/09/30 14:50:10:0964 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(51), Task(9217672)
QP, func dynAlloc, line 386 in //DSN:USER.TEMP.SOURCE(AARCSPDA)
'DYNALLOC message 1 of 1
'IKJ56228I DATA SET USER.STORPROC.XCL NOT IN CATALOG OR CATALOG
'CAN NOT BE ACCESSED
```

Example 2: Results of attempting to create a data set with a space quantity that is not sufficient.

InfoSphere DataStage log messages:

```
[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0201.
ALLOC error, rc=4, reason=38668(0x0000970c), info=0(0x00000000)

[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0214.
IKJ56893I DATA SET SYSUID.STORPROC.HUGE NOT ALLOCATED+

[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0214.
IGD17273I ALLOCATION HAS FAILED FOR ALL VOLUMES SELECTED FOR DATA SET

[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0214.
SYSUID.STORPROC.HUGE

[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0214.
IGD17277I THERE ARE (25) CANDIDATE VOLUMES OF WHICH (25) ARE ENABLED OR QUIESCED

[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0214.
IGD17290I THERE WERE 1 CANDIDATE STORAGE GROUPS OF WHICH THE FIRST 1

[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0214.
WERE ELIGIBLE FOR VOLUME SELECTION.

[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0214.
THE CANDIDATE STORAGE GROUPS WERE:DEVELOP

[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0214.
IGD17279I 25 VOLUMES WERE REJECTED BECAUSE THEY DID NOT HAVE SUFFICIENT
SPACE (041A041D)
```

Classic system log messages:

```
2009/09/30 14:29:41:6934 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(59), Task(9217672)
QP, func dynAlloc, line 335 in //DSN:SYSUID.TEMP.SOURCE(AARCSPDA)
'osdynalloc() string:
'retddn=?,retdsn=?,retvolume=?,reason=?,InfoReason=?,smsreason=?,
'emsgp=?,msgcount=?,dsn=SYSUID.STORPROC.HUGE,disp=NEW,ndisp=CATLG,
'unit=SYSALLDA,cyl,primary=4000
'-----
'S99ERROR=0x0000970C S99INFO=0x00000000 S99ERSN=0x00004379

2009/09/30 14:29:41:6936 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(59), Task(9217672)
QP, func dynAlloc, line 386 in //DSN:SYSUID.TEMP.SOURCE(AARCSPDA)
'DYNALLOC message 1 of 8
```

```

      'IKJ56893I DATA SET SYSUID.STORPROC.HUGE NOT ALLOCATED+
2009/09/30 14:29:41:6936 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(59), Task(9217672)
QP, func dynAlloc, line 386 in //DSN:SYSUID.TEMP.SOURCE(AARCPDA)
'DYNALLOC message 2 of 8
'IGD17273I ALLOCATION HAS FAILED FOR ALL VOLUMES SELECTED FOR DATA
'SET
2009/09/30 14:29:41:6937 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(59), Task(9217672)
QP, func dynAlloc, line 386 in //DSN:SYSUID.TEMP.SOURCE(AARCPDA)
'DYNALLOC message 3 of 8
'SYSUID.STORPROC.HUGE
2009/09/30 14:29:41:6937 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(59), Task(9217672)
QP, func dynAlloc, line 386 in //DSN:SYSUID.TEMP.SOURCE(AARCPDA)
'DYNALLOC message 4 of 8
'IGD17277I THERE ARE (25) CANDIDATE VOLUMES OF WHICH (25) ARE
'ENABLED OR QUIESCED
2009/09/30 14:29:41:6938 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(59), Task(9217672)
QP, func dynAlloc, line 386 in //DSN:SYSUID.TEMP.SOURCE(AARCPDA)
'DYNALLOC message 5 of 8
'IGD17290I THERE WERE 1 CANDIDATE STORAGE GROUPS OF WHICH THE FIRST
' 1
2009/09/30 14:29:41:6939 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(59), Task(9217672)
QP, func dynAlloc, line 386 in //DSN:SYSUID.TEMP.SOURCE(AARCPDA)
'DYNALLOC message 6 of 8
'WERE ELIGIBLE FOR VOLUME SELECTION.
2009/09/30 14:29:41:6940 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(59), Task(9217672)
QP, func dynAlloc, line 386 in //DSN:SYSUID.TEMP.SOURCE(AARCPDA)
'DYNALLOC message 7 of 8
'THE CANDIDATE STORAGE GROUPS WERE:DEVELOP
2009/09/30 14:29:41:6941 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(59), Task(9217672)
QP, func dynAlloc, line 386 in //DSN:SYSUID.TEMP.SOURCE(AARCPDA)
'DYNALLOC message 8 of 8
'IGD17279I 25 VOLUMES WERE REJECTED BECAUSE THEY DID NOT HAVE
'SUFFICIENT SPACE (041A041D)

```

Example 3: Results of attempting to create a new data set with a name that duplicates the name of an existing cataloged data set.

InfoSphere DataStage log messages:

```

[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0201.
CREATE error, rc=4, reason=38668(0x0000970c), info=0(0x00000000)

[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0214.
IKJ56893I DATA SET SYSUID.STORPROC.JCL NOT ALLOCATED+

[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0214.
IGD17101I DATA SET SYSUID.STORPROC.JCL

[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0214.
NOT DEFINED BECAUSE DUPLICATE NAME EXISTS IN CATALOG

[IS Classic][ODBC/CLI Driver][Data Server] SQLExecute - Error code: x00DA0214.
RETURN CODE IS 8 REASON CODE IS 38 IGG0CLEH

```

Classic system log messages:

```
2009/10/08 14:29:38:3741 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(59), Task(9217672)
QP, func dynAlloc, line 335 in //DSN:SYSUID.TEMP.SOURCE(AARCSPDA)
'osdynalloc() string:
'retddn=?,retdsn=?,retvolume=?,reason=?,InfoReason=?,smsreason=?,
'emsgp=?,msgcount=?,dsn=SYSUID.STORPROC.JCL,member=SEQNEW,disp=NEW
'-----
'S99ERROR=0x0000970C S99INFO=0x00000000 S99ERSN=0x000042CD

2009/10/08 14:29:38:3742 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(59), Task(9217672)
QP, func dynAlloc, line 386 in //DSN:SYSUID.TEMP.SOURCE(AARCSPDA)
'DYNALLOC message 1 of 4
'IKJ56893I DATA SET SYSUID.STORPROC.JCL NOT ALLOCATED+

12009/10/08 14:29:38:3743 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(59), Task(9217672)
QP, func dynAlloc, line 386 in //DSN:SYSUID.TEMP.SOURCE(AARCSPDA)
'DYNALLOC message 2 of 4
'IGD17101I DATA SET SYSUID.STORPROC.JCL

2009/10/08 14:29:38:3744 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(59), Task(9217672)
QP, func dynAlloc, line 386 in //DSN:SYSUID.TEMP.SOURCE(AARCSPDA)
'DYNALLOC message 3 of 4
'NOT DEFINED BECAUSE DUPLICATE NAME EXISTS IN CATALOG

2009/10/08 14:29:38:3744 RC(08), SpcRC(00da0214), Data(00000000,00000000)
Node(59), Task(9217672)
QP, func dynAlloc, line 386 in //DSN:SYSUID.TEMP.SOURCE(AARCSPDA)
'DYNALLOC message 4 of 4
'RETURN CODE IS 8 REASON CODE IS 38 IGG0CLEH
```

Chapter 7. Troubleshooting and support

To isolate and resolve problems with your IBM software, you can use the troubleshooting and support information, which contains instructions for using the problem-determination resources that are provided with your IBM products.

Troubleshooting a problem

Troubleshooting is a systematic approach to solving a problem. The goal of troubleshooting is to determine why something does not work as expected and how to resolve the problem.

The first step in the troubleshooting process is to describe the problem completely. Problem descriptions help you and the IBM technical-support representative know where to start to find the cause of the problem. This step includes asking yourself basic questions:

- What are the symptoms of the problem?
- Where does the problem occur?
- When does the problem occur?
- Under which conditions does the problem occur?
- Can the problem be reproduced?

The answers to these questions typically lead to a good description of the problem, which can then lead you a problem resolution.

What are the symptoms of the problem?

When starting to describe a problem, the most obvious question is “What is the problem?” This question might seem straightforward; however, you can break it down into several more-focused questions that create a more descriptive picture of the problem. These questions can include:

- Who, or what, is reporting the problem?
- What are the error codes and messages?
- How does the system fail? For example, is it a loop, hang, crash, performance degradation, or incorrect result?

Where does the problem occur?

Determining where the problem originates is not always easy, but it is one of the most important steps in resolving a problem. Many layers of technology can exist between the reporting and failing components. Networks, disks, and drivers are only a few of the components to consider when you are investigating problems.

The following questions help you to focus on where the problem occurs to isolate the problem layer:

- Is the problem specific to one platform or operating system, or is it common across multiple platforms or operating systems?
- Is the current environment and configuration supported?

If one layer reports the problem, the problem does not necessarily originate in that layer. Part of identifying where a problem originates is understanding the environment in which it exists. Take some time to completely describe and document the problem environment, including the following items:

- Operating system
- Product version
- All corresponding software and versions, hardware information, and any maintenance that was applied
- The product that you are accessing (for example IMS, Adabas, CA-Datcom) and the product version
- The client application that you are using, if applicable. If you are using a client application other than Classic Data Architect or the sample application clisamp, confirm if you can successfully run the query using either the Classic Data Architect or clisamp.

Confirm that you are running within an environment that is a supported configuration; many problems can be traced back to incompatible levels of software that are not intended to run together or have not been fully tested together.

When does the problem occur?

Develop a detailed timeline of events leading up to a failure, especially for those cases that are one-time occurrences. You can most easily develop a timeline by working backward: Start at the time an error was reported (as precisely as possible, even down to the millisecond), and work backward through the available logs and information. Typically, you need to look only as far as the first suspicious event that you find in a diagnostic log.

To develop a detailed timeline of events, answer these questions:

- Does the problem happen only at a certain time of day or night?
- How often does the problem happen?
- What sequence of events leads up to the time that the problem is reported?
- Does the problem happen after an environment change, such as upgrading or installing software or hardware?

Responding to these types of questions can give you a frame of reference in which to investigate the problem.

Under which conditions does the problem occur?

Knowing which systems and applications are running at the time that a problem occurs is an important part of troubleshooting. These questions about your environment can help you to identify the root cause of the problem:

- Does the problem always occur when the same task is being performed?
- Does a certain sequence of events need to occur for the problem to surface?
- Do any other applications fail at the same time?

Answering these types of questions can help you explain the environment in which the problem occurs and correlate any dependencies. Remember that just because multiple problems might have occurred around the same time, the problems are not necessarily related.

Can the problem be reproduced?

From a troubleshooting standpoint, the ideal problem is one that can be reproduced. Typically, when a problem can be reproduced you have a larger set of tools or procedures at your disposal to help you investigate. Consequently, problems that you can reproduce are often easier to debug and solve. However, problems that you can reproduce can have a disadvantage: If the problem is of significant business impact, you do not want it to recur. If possible, re-create the problem in a test or development environment, which typically offers you more flexibility and control during your investigation.

- Can the problem be re-created on a test system?
- Are multiple users or applications encountering the same type of problem?
- Can the problem be re-created by running a single command, a set of commands, or a particular application?
- Can the problem be re-created by issuing a specific query?

Searching for messages

You can search for messages in the information center.

In the search box that is located in the **top-left toolbar** of this information center, enter the message number; for example, enter: 0x00670014.

Important: You need to enter the search string in the format of the full message number, in this example 0x00670014. Do not specify partial message numbers or wild cards (* or ?) in the search string.

Searching knowledge bases

You can often find solutions to problems by searching IBM knowledge bases. You can optimize your results by using available resources, support tools, and search methods.

About this task

You can find useful information by searching the information center, but sometimes you need to look beyond the information center to answer your questions or resolve problems.

Procedure

To search knowledge bases for information that you need, use one or more of the following approaches:

- Find the content that you need by using the IBM Support Portal.
The IBM Support Portal is a unified, centralized view of all technical support tools and information for all IBM systems, software, and services. The IBM Support Portal lets you access the IBM electronic support portfolio from one place. You can tailor the pages to focus on the information and resources that you need for problem prevention and faster problem resolution.
Familiarize yourself with the IBM Support Portal by viewing the demo videos about this tool. These videos introduce you to the IBM Support Portal, explore troubleshooting and other resources, and demonstrate how you can tailor the page by moving, adding, and deleting portlets.
- Search for content by using one of the following additional technical resources:

- APARs (problem reports). You can locate APARs on the IBM Support site or by using an external search engine. To locate APARs on the IBM Support site:
 1. Select Information Management in the **Choose support type** box.
 2. Select a Classic product in the **Choose product** box.
 3. Enter an APAR number in the **Search** box.

Tip: To narrow the search for Classic products only, specify the component identifier “5697I8200” for Classic products.
- Information management forums. This page lists a variety of forums about specific IBM Information Management products.
- Search for content by using the IBM masthead search. You can use the IBM masthead search by typing your search string into the Search field at the top of any ibm.com page.
- Search for content by using any external search engine. If you use an external search engine, your results are more likely to include information that is outside the ibm.com domain. However, sometimes you can find useful problem-solving information about IBM products in newsgroups, forums, and blogs that are not on ibm.com.

Tip: Include “IBM” and the name of the product in your search if you are looking for information about an IBM product.

Getting fixes

A product fix might be available to resolve your problem.

About this task

Procedure

- To find and install fixes:
 1. Access downloads and fixes:
 - If you know a PTF number, go to Download specific fixes. You might know a PTF number from a technote or APAR description that you found by entering keywords for a search of the product support web site.
 - If you do not know a PTF number, go to the IBM Support Portal. From there, you can search for fixes for your product. If you have not visited the IBM Support Portal in the past, you can customize it so that you can view Support-related information for the specific products that you use. Alternatively, visit the Get zSeries related fixes web site.
 2. Follow the instructions at the eServer™ zSeries website to locate a fix that might solve your problem.
 3. When you find a fix that you are interested in, click the name of the fix to read its description. If you believe that the fix can resolve your problem, download the fix and apply it.
 4. Optional: Subscribe to receive weekly email notifications about fixes and other IBM Support information.
- To find a list of fixes for a product rollup, see the Release Notes for the rollup that you need.
- To find fixes for the Classic Data Architect, run the IBM Installation Manager. See Applying maintenance to the Classic Data Architect for instructions.

Contacting IBM Support

IBM Support provides assistance with product defects, answering FAQs, and performing rediscovery.

Before you begin

After trying to find your answer or solution by using other self-help options such as technotes, you can contact IBM Support. Before contacting IBM Support, your company must have an active IBM maintenance contract, and you must be authorized to submit problems to IBM. For information about the types of available support, see the Support portfolio topic in the *Software Support Handbook*.

Procedure

Complete the following steps to contact IBM Support with a problem:

1. Define the problem, gather background information, and determine the severity of the problem. For more information, see the Getting IBM support topic in the *Software Support Handbook*.
2. Gather diagnostic information.
3. Submit the problem to IBM Support in one of the following ways:
 - Online through the IBM Support Portal: You can open, update, and view all your Service Requests from the Service Request portlet on the Service Request page.
 - By phone: For the phone number to call in your country, see the Directory of worldwide contacts web page.

Results

If the problem that you submit is for a software defect or for missing or inaccurate documentation, IBM Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the IBM Support website daily, so that other users who experience the same problem can benefit from the same resolution.

Exchanging information with IBM

To diagnose or identify a problem, you might need to provide IBM Support with data and information from your system. In other cases, IBM Support might provide you with tools or utilities to use for problem determination.

Collecting diagnostic information

You can use Classic Data Architect to collect diagnostic information from all connected servers. These diagnostics can be exported to a file for evaluation.

About this task

When you start collecting diagnostic information, all of the information is stored in Classic Data Architect cached memory. You can set the frequency at which diagnostic information is collected from the servers and the maximum number of results to be cached in memory. These options are available on the **Window > Preferences > Classic Data Architect > Diagnostic Metric Options** panel.

The information is also displayed in the Diagnostic Metric view. The metrics that have been collected from the selected server are displayed in this view, sorted by timestamp. Previous results (up to the maximum that you specify) and results for non-selected servers are still available in cached memory and can be exported to a file.

Procedure

1. To display the Diagnostic Metric view, select **Window > Views > Diagnostic Metric view**.
2. To start collecting diagnostics, right-click in the Diagnostic Metric view and select **Start Diagnostic Metric Collection**.
3. Select a server to view the most up-to-date metrics for a particular server. When you select an object in the tree, the diagnostic information to the right is updated dynamically with the latest results collected.
4. To export diagnostic information to a file, right-click the Diagnostic Metric view and select **Export Diagnostic Metrics**. The metrics are exported as comma-separated values (CSV) to a file of your choosing. You can export the metrics currently displayed in the view or all metrics cached in memory. Metrics exported from the view are exported in the order that they are displayed. Metrics exported from memory are sorted by timestamp.
5. To clear all collected diagnostics, including those stored in memory, right-click the Diagnostic Metric view and select **Clear Diagnostic Metric History**.

Sending information to IBM Support

To reduce the time that it takes to resolve your problem, you can send trace and diagnostic information to IBM Support.

Procedure

To submit diagnostic information to IBM Support:

1. Open a problem management record (PMR).
2. Collect the diagnostic data that you might need, either manually or automatically, depending on the data. Diagnostic data helps reduce the time that is spent resolving your PMR. For example, having access to any relevant messages, error codes, log data, all data server output, trace output, or dump output, can speed the resolution process.
3. Compress the files by using one of the following methods, depending on the file type.
 - Use the AMATERSE program, which is a tool that is available for products that run in a z/OS environment. For more information about what z/OS versions support this program, search for either program name on ibm.com.
 - For UNIX files, use the tar and gzip programs to create compressed archive files.
 - a. Run the tar program against the file.
 - b. Run the gzip program against the file.For example: `tar -cvf - inputfile1 inputfile2 | gzip > file.tar.gz`
 - For Microsoft Windows files, create a ZIP file.
4. Transfer the files to IBM. You can use one of the following methods to transfer the files to IBM:
 - Service Request tool
 - Standard data upload methods: FTP, HTTP

- Secure data upload methods: FTPS, SFTP, HTTPS
- Email

If you are using a z/OS product and you use ServiceLink/IBMLink to submit PMRs, you can send diagnostic data to IBM Support in an email or by using FTP.

All of these data exchange methods are explained on the IBM Support site.

Receiving information from IBM Support

Occasionally an IBM technical-support representative might ask you to download diagnostic tools or other files. You can use FTP to download these files.

Before you begin

Ensure that your IBM technical-support representative provided you with the preferred server to use for downloading the files and the exact directory and file names to access.

Procedure

To download files from IBM Support:

1. Use FTP to connect to the site that your IBM technical-support representative provided and log in as anonymous. Use your email address as the password.
2. Change to the appropriate directory:
 - a. Change to the `/fromibm` directory.
`cd fromibm`
 - b. Change to the directory that your IBM technical-support representative provided.
`cd nameofdirectory`
3. Enable binary mode for your session.
`binary`
4. Use the **get** command to download the file that your IBM technical-support representative specified.
`get filename.extension`
5. End your FTP session.
`quit`

Subscribing to Support updates

To stay informed of important information about the IBM products that you use, you can subscribe to updates.

About this task

By subscribing to receive updates, you can receive important technical information and updates for specific Support tools and resources. You can subscribe to updates by using one of two approaches:

RSS feeds and social media subscriptions

The following RSS feeds and social media subscriptions are available:

- RSS feeds for various Information Management communities. See the Information Integration section of Information Management community.

- RSS feed for developerWorks® resources, such as articles, tutorials, downloads, and forums. See developerWorks.

For general information about RSS, including steps for getting started and a list of RSS-enabled IBM web pages, visit the IBM Software Support RSS feeds site.

My Notifications

With My Notifications, you can subscribe to Support updates for any IBM product. You can specify that you want to receive daily or weekly email announcements. You can specify what type of information you want to receive (such as publications, hints and tips, product flashes (also known as alerts), downloads, and drivers). My Notifications enables you to customize and categorize the products about which you want to be informed and the delivery methods that best suit your needs.

Procedure




To subscribe to Support updates:

1. To subscribe to My Notifications, begin by going to the IBM Support Portal and clicking **My Notifications** in the **Notifications** portlet.
2. If you have already registered for My support, sign in and skip to the next step. If you have not registered, click **Register now**. Complete the registration form using your email address as your IBM ID and click **Submit**.
3. Click **Edit profile**.
4. Click **Add products** and choose a product category; for example, **Software**. A second list is displayed.
5. In the second list, select a product segment; for example, **Data & Information Management**. A third list is displayed.
6. In the third list, select a product subsegment, for example, **Databases**. A list of applicable products is displayed.
7. Select the products for which you want to receive updates.
8. Click **Add products**.
9. After selecting all products that are of interest to you, click **Subscribe to email** on the **Edit profile** tab.
10. Select **Please send these documents by weekly email**.
11. Update your email address as needed.
12. In the **Documents list**, select the product category; for example, **Software**.
13. Select the types of documents for which you want to receive information.
14. Click **Update**.

Results

Until you modify your RSS feeds and My Notifications preferences, you receive notifications of updates that you have requested. You can modify your preferences when needed (for example, if you stop using one product and begin using another product).

Related information

-  [IBM Software Support RSS feeds](#)
-  [Subscribe to My Notifications support content updates](#)
-  [My notifications for IBM technical support](#)

 My notifications for IBM technical support overview

Chapter 8. Reference

The reference for Classic federation includes descriptions of the configuration parameters for the data server, services, and clients; utilities; SQL information; programming information about the drivers; and technical details about multilingual data; sample VTAM and CICS definitions for federated or stored procedure access; and field procedures.

Services and configuration parameters

Summary of services

The services required for the Classic data server are customized and running when you complete the installation customization process.

The following table summarizes these services.

Table 49. Summary of services

Service type	Task name	Service class
CA-Datacom access service	CACDCI	DCI
Client connection handler service	CACINIT	INIT
IBM DB2 call attachment facility (CAF) initialization service	CACCAF	CAF
IMS BMP/DBB access service	CACIMSIF	IMSI
IMS open database access (ODBA) access service	CACRRSI	ODBA
Logger service	CACLOG	LOG
Monitoring service	CECMAA	MAA
Query processor service	CACQP	QP
Region controller service	CACCNTL	CNTL
Stored procedure service	CECSPS	SPS
Remote operator command service	CACOPER	OPER
Two-phase commit query processor service	CACQPRRS	QPRR
VSAM access service	CACVSMS	VSMS
Workload Manager (WLM) service	CACWLM	WLM

The IBM Language Environment initialization service is deprecated in Version 10.1.

Call attachment facility (CAF) service

The CAF service is assigned to the CAF service class. The task name for the CAF service class is CACCAF.

The CAF is an initialization service that connects to an IBM DB2 for z/OS subsystem to access and update DB2 data.

The following table summarizes the configuration parameters that define call attachment facility services in the CAF service class.

Table 50. Configuration parameters for the CAF service class

Parameter	Default value	Description
DB2PLANNNAME	CACPLAN	IBM DB2 application plan for accessing DB2 for z/OS data
DB2SUBSYSTEMNAME	DSN	DB2 for z/OS subsystem name
IDLETIMEOUT	300000 MS (5 MIN)	The amount of time in milliseconds that a service remains idle before it polls the local message queue for messages that are to be processed
INITIALTHREADS	1	Number of instances of this service that the region controller starts during data server initialization
MAXTHREADS	1	Maximum number of instances of this service that the region controller is allowed to start
MAXUSERS	10	Maximum number of user connections
OPTIONALUSERPARM	None	Optional user parameters
RESPONSETIMEOUT	300000 MS (5 MIN)	Maximum amount of time in milliseconds to wait for response before terminating a connection
SEQUENCE	0	Sequence number that is assigned to services
THREADMGMTTEXT	None	Thread management exit
TRACELEVEL	4	Trace level

CA-Datacom access service

The CA-Datacom access service is assigned to the CA-Datacom initialization (DCI) service class. The task name for the DCI service class is CACDCI.

The CA-Datacom service initializes the Classic data server for connections to the CA-Datacom Multi-User Facility (MUF).

The following table summarizes the configuration parameters that define CA-Datacom access services in the DCI service class.

Table 51. Configuration parameters for the DCI service class

Parameter	Default value	Description
IDLETIMEOUT	300000 MS (5 MIN)	The amount of time in milliseconds that a service remains idle before it polls the local message queue for messages that are to be processed
INITIALTHREADS	1	Number of instances of this service that the region controller starts during initialization of the Classic data server
MAXTHREADS	1	Maximum number of instances of this service that the region controller is allowed to start
MAXUSERS	50	Maximum number of user connections

Table 51. Configuration parameters for the DCI service class (continued)

Parameter	Default value	Description
RESPONSETIMEOUT	300000 MS (5 MIN)	Maximum amount of time in milliseconds to wait for response before terminating a connection
SEQUENCE	0	Sequence number that is assigned to services
URTPOOL	FALSE	Specifies whether the Datacom access service acquires pooled user requirements tables. The default value of FALSE repeatedly opens and closes user resource tables when each query runs. Set this parameter to TRUE to open a user requirements table once, and keep it open for the life of the Datacom access service. The user requirements table remains open until the Datacom access service or the Classic data server stops, or until you close the user requirements table.
TASKAREASACQ	1	The total number of CA-Datacom task areas to acquire for accessing CA-Datacom. If you enable pooled user requirements tables, the Datacom access service provides an independent group of task areas for RRS processing.
TASKAREASRES	0	The total number of CA-Datacom task areas reserved for modification queries, which perform inserts, updates, and deletes. If you enable pooled user requirements tables, this parameter has no effect on the task areas that perform RRS processing.
TRACELEVEL	4	Trace level

Connection handler service

The connection handler service is assigned to the INIT service class. The task name for the INIT service is CACINIT.

A connection handler listens for connection requests from client applications and routes the requests to the appropriate monitoring, operator, and query processor tasks. The connection handler task can load the following communication protocols:

- TCP/IP IBM z/OS
- Cross-memory services

A local client application can connect to a Classic data server by using any of these protocols. Remote client applications use TCP/IP to communicate with a Classic data server.

The following table summarizes the configuration parameters that define connection handler services in the INIT service class.

Table 52. Configuration parameters for the INIT service class

Parameter	Default value	Description
COMMSTRING	TCP/0.0.0.0/9087	Client connection listen string

Table 52. Configuration parameters for the INIT service class (continued)

Parameter	Default value	Description
IDLETIMEOUT	5M	The amount of time that a service remains idle before it polls the local message queue for messages that are to be processed
INITIALTHREADS	1	Number of instances of this service that the region controller starts during initialization of the Classic data server
MAXTHREADS	1	Maximum number of instances of this service that the region controller is allowed to start
MAXUSERS	100	Maximum number of user connections
RESPONSETIMEOUT	5M	Maximum amount of time to wait for response before terminating a connection
SEQUENCE	0	Sequence number that is assigned to services.
TRACELEVEL	4	Trace level

IMS BMP/DBB access service

The IBM IMS BMP/DBB access service is assigned to the IMSI service class. The task name for the IMSI service class is CACIMSIF.

The IMS BMP/DBB service initializes the IMS region controller to access IMS data by using the BMP or DBB interface.

The following table summarizes the configuration parameters that define IMS BMP/DBB access services in the IMSI service class.

Table 53. Configuration parameters for the IMSI service class

Parameter	Default value	Description
IDLETIMEOUT	300000 MS (5 MIN)	The amount of time in milliseconds that a service remains idle before it polls the local message queue for messages that are to be processed
INITIALTHREADS	1	Number of instances of this service that the region controller starts during initialization of the Classic data server
MAXTHREADS	1	Maximum number of instances of this service that the region controller is allowed to start
MAXUSERS	10	Maximum number of user connections
RESPONSETIMEOUT	300000 MS (5 MIN)	Maximum amount of time in milliseconds to wait for response before terminating a connection
SEQUENCE	0	Sequence number assigned to services
TRACELEVEL	4	Trace level

IMS open database access (ODBA) access service

The IBM IMS open database access (ODBA) access service is assigned to the ODBA service class. The task name for the ODBA service class is CACRRSI.

The ODBA service enables client applications to run distributed transactions with two-phase commit protocols. The ODBA interface must be used in conjunction with the two-phase commit query processor. The QPRR service class defines two-phase commit query processor.

The ODBA interface uses the DRA interface to connect to IMS.

The following table summarizes the configuration parameters that define IMS ODBA access services in the ODBA service class.

Table 54. Configuration parameters for the ODBA service class

Parameter	Default value	Description
IDLETIMEOUT	300000 MS (5 MIN)	The amount of time in milliseconds that a service remains idle before it polls the local message queue for messages that are to be processed
INITIALTHREADS	1	Number of instances of this service that the region controller starts during initialization of the Classic data server
MAXTHREADS	1	Maximum number of instances of this service that the region controller is allowed to start
MAXUSERS	1	Maximum number of user connections
PSBPARAM	None	PSB parameter
RESPONSETIMEOUT	60000 MS (1 MIN)	Maximum amount of time in milliseconds to wait for response before terminating a connection
SEQUENCE	0	Sequence number that is assigned to services
SSNPARAM	None	Subsystem name (SSN) parameter
TRACELEVEL	4	Trace level

Logger service

The logger service is assigned to the LOG service class. The task name for the LOG service class is CACLOG.

The logger service receives messages from all services in the data server and coordinates writing the messages to a common log. The logger also reports data server activities and is used for error diagnosis.

Restriction: A single logger task can run within a data server.

When the logger service is initialized, the configuration parameters in the LOG service class are defined with the parameter default values. You can modify the default values as needed.

The following table lists the configuration parameters that apply to the LOG service class.

Table 55. Configuration parameters for the LOG service class

Parameter	Default value	Description
CONSOLELEVEL	4	The amount of event messages that data server tasks record in the event log
DISPLAYLOG	FALSE	Display log

Table 55. Configuration parameters for the LOG service class (continued)

Parameter	Default value	Description
EIFEVENTSERVERS*	None	The service list for URL values that identifies the event server to which Event Integration Facility (EIF) events will be sent
EVENTLOG	None	The name of the event message log that is defined to the logger service
IDLETIMEOUT	5M	The amount of time that a service remains idle before it polls the local message queue for messages that are to be processed
INITIALTHREADS	1	Number of instances of this service that the region controller starts during data server initialization
LOGBUFSIZE	65536	Log buffer size
LOGURL	None	Log URL that provides a method of moving logging storage outside of the address space MESSAGEPOOLSIZE storage and into a data space.
MAXTHREADS	1	Maximum number of instances of this service that the region controller is allowed to start
MAXUSERS	100	Maximum number of user connections
MSGLIST	None	Represents a message list that is maintained as a service list.
RESPONSETIMEOUT	5M	Maximum amount of time to wait for response before terminating a connection
SEQUENCE	0	Sequence number that is assigned to services
STREAMNAME	None	Stream name used for the diagnostic log
TRACELEVEL	1	Trace level

* The EIFEVENTSERVERS parameters does not apply to Classic federation.

Monitoring service

The monitoring service is assigned to the MAA service class. The task name for the MAA service class is CECMAA.

The monitoring service provides management functions that include reporting, receiving display and report commands, and producing the reports needed from runtime information.

The following table lists the configuration parameters that apply to the MAA service class.

Table 56. Configuration parameters for the MAA service class.

Parameter	Default value	Description
IDLETIMEOUT	5M	The amount of time that a service remains idle before it polls the local message queue for messages that are to be processed
INITIALTHREADS	1	Number of instances of this service that the region controller starts during initialization of the Classic data server

Table 56. Configuration parameters for the MAA service class. (continued)

Parameter	Default value	Description
MAXTHREADS	1	Maximum number of instances of this service that the region controller is allowed to start
MAXUSERS	100	Maximum number of user connections
NMICOMMSTRING	None	The communication path for the Network Management Interface (NMI) AF_UNIX domain socket.
RESPONSETIMEOUT	3S	Maximum amount of time to wait for response before terminating a connection
SAFEXIT	None	Name of the System Authorization Facility (SAF) system exit
SEQUENCE	0	Sequence number that is assigned to services
TRACELEVEL	4	Trace level

Operator service

The command operator service is assigned to the OPER service class. The task name for the OPER service class is CACOPER.

The operator service supports a command operator interface for distributed client applications.

The operator service also handles communications between the Classic data server and the configuration support in the Classic Data Architect. To use the configuration support in the Classic Data Architect, the operator service must be running on the Classic data server.

The following table summarizes the configuration parameters that define command operator services in the OPER service class.

Table 57. Configuration parameters for the OPER service class

Parameter	Default value	Description
IDLETIMEOUT	5M	The amount of time that a service remains idle before it polls the local message queue for messages that are to be processed
INITIALTHREADS	1	Number of instances of this service that the region controller starts during initialization of the Classic data server
MAXTHREADS	1	Maximum number of instances of this service that the region controller is allowed to start
MAXUSERS	100	Maximum number of user connections
RESPONSETIMEOUT	5M	Maximum amount of time to wait for response before terminating a connection
SAFEXIT	None	Name of the System Authorization Facility (SAF) system exit
SEQUENCE	0	Sequence number that is assigned to services
SMFEXIT	None	Name of the System Management Facility (SMF) accounting exit that reports clock time and CPU time for a user session

Table 57. Configuration parameters for the OPER service class (continued)

Parameter	Default value	Description
SQLSECURITY	FALSE	Determines the level of access privilege verification that will be performed on operator commands
TRACELEVEL	4	Trace level

Query processor service

The single-phase commit query processor service is assigned to the QP service class. The task name for the QP service class is CACQP.

The query processor is the subcomponent of the Classic data server that processes SQL requests. The SQL requests can access a single database or file system or reference multiple types of databases or file systems.

The single-phase commit query processor accesses and joins information from multiple data sources and performs updates to a single data source.

The following table summarizes the configuration parameters that define query processor services in the QP service class.

Table 58. Configuration parameters for the QP service class

Parameter	Default value	Description
BTREEBUFFS	4	Number of in-memory B-tree buffer caches used before data is written out
CPUGOVERNOR	None	CPU governor
IDLETIMEOUT	5M	The amount of time that a service remains idle before it polls the local message queue for messages that are to be processed
INITIALTHREADS	5	Number of instances of this service that the region controller starts during initialization of the Classic data server
JOINTABLES	4	Join optimization
MAXROWSEXAMINED	0	Maximum rows examined
MAXROWSEXCACTION	1 = RETURN	Maximum rows exceeded action
MAXROWSRETURNED	0	Maximum number of rows returned
MAXTHREADS	10	Maximum number of instances of this service that the region controller is allowed to start
MAXUSERS	20	Maximum number of user connections
RESPONSETIMEOUT	5M	Maximum amount of time to wait for response before terminating a connection
SAFEXIT	None	System Authorization Facility (SAF) system exit
SEQBUFNO	0	Effective BUFNO value for a given file.
SEQBUFNOCALC	0	Method to use for determining the BUFNO value when accessing sequential files.
SEQUENCE	0	Sequence number that is assigned to services

Table 58. Configuration parameters for the QP service class (continued)

Parameter	Default value	Description
SMFEXIT	None	System Management Facility (SMF) accounting exit that reports clock time and CPU time for a user session
STMTRETENTION	0 = SYNCPOINT	Statement retention that defines the behavior of a prepared statement when a commit or rollback operation occurs
TEMPFILESSPACE	HIPERSPACE, INIT=8M, MAX=2048M, EXTEND=8M	Temporary file space in megabytes
TRACELEVEL	4	Trace level
USERSUBPOOLMAX	8192	User pool maximum
VSAMAMPARMS	None	VSAM buffering information for VSAM files
WLMUOW	None	Workload Manager (WLM) unit of work activities

Region controller service

The region controller service is assigned to the CNTL service class. The task name for the CNTL service class is CACCNTL.

The region controller service monitors and controls the other services that run within the Classic data server.

The region controller directly or indirectly activates each service according to the configuration parameters that you define. The region controller starts, stops, and monitors the other tasks that run within the Classic data server.

The region controller also includes an IBM z/OS MTO (master terminal operator) interface that you can use to monitor and control an address space for a Classic data server.

The following table lists the configuration parameters for the CNTL service class.

Table 59. Configuration parameters for the CNTL service class

Parameter	Default value	Description
DBCSCODEPAGE*	0	Double-byte CCSID that the z/OS operating system uses where the Classic data server is running.
HOSTCODEPAGE	37	Host code page of the Classic data server
IDLETIMEOUT	5M	The amount of time that a service remains idle before it polls the local message queue for messages that are to be processed
INITIALTHREADS	1	Number of instances of this service that the region controller starts during initialization of the Classic data server
MAXTHREADS	1	Maximum number of instances of this service that the region controller is allowed to start
MAXUSERS	100	Maximum number of user connections

Table 59. Configuration parameters for the CNTL service class (continued)

Parameter	Default value	Description
RESPONSETIMEOUT	5M	Maximum amount of time to wait for response before terminating a connection
SEQUENCE	0	Sequence number that is assigned to services
TRACELEVEL	4	Trace level

* The DBCSCODEPAGE parameter does not apply to Classic federation.

Two-phase commit query processor service

The two-phase commit query processor service is assigned to the QPRR service class. The task name for the QPRR service class is CACQPRRS.

The two-phase commit query processor accesses and joins information from multiple data sources and performs updates to multiple data sources. This query processor supports the CA-Datcom, IBM DB2 for z/OS, IBM IMS, and transactional VSAM (TVS) data sources.

The two-phase commit query processor uses z/OS Resource Recovery Services to coordinate the data source updates. This query processor can participate in distributed transactions by using a JDBC client and a distributed transaction manager (such as IBM WebSphere Application Server).

The following table summarizes the configuration parameters that define two-phase commit query processor services in the QPRR service class.

Table 60. Configuration parameters for the QPRR service class

Parameter	Default value	Description
BTREEBUFFS	4	Number of in-memory B-tree buffer caches used before data is written out
CPUGOVERNOR	None	CPU governor
IDLETIMEOUT	300000 MS (5 MIN)	The amount of time in milliseconds that a service remains idle before it polls the local message queue for messages that are to be processed
INITIALTHREADS	1	Number of instances of this service that the region controller starts during initialization of the Classic data server
JOINTABLES	4	Join optimization
MAXROWSEXAMINED	0	Maximum rows examined
MAXROWSEXCACTION	1 = RETURN	Maximum rows exceeded action
MAXROWSRETURNED	0	Maximum number of rows returned
MAXTHREADS	1	Maximum number of instances of this service that the region controller is allowed to start
MAXUSERS	20	Maximum number of user connections
RESPONSETIMEOUT	300000 MS (5 MIN)	Maximum amount of time in milliseconds to wait for response before terminating a connection
SAFEXIT	None	System Authorization Facility (SAF) system exit

Table 60. Configuration parameters for the QPRR service class (continued)

Parameter	Default value	Description
SEQBUFNO	0	Effective BUFNO value for a given file.
SEQBUFNOCALC	0	Method to use for determining the BUFNO value when accessing sequential files.
SEQUENCE	0	Sequence number that is assigned to services
SMFEXIT	None	System Management Facility (SMF) accounting exit that reports clock time and CPU time for a user session
STMTRETENTION	0 = SYNCPOINT	Statement retention that defines the behavior of a prepared statement when a commit or rollback operation occurs
TEMPFILESPEC	HIPERSPACE, INIT=8 M, MAX=2048 M, EXTEND=8 M	Temporary file space in megabytes
TRACELEVEL	4	Trace level
USERSUBPOOLMAX	8192	User pool maximum
VSAMAMPARMS	None	VSAM buffering information for VSAM files
WLMUOW	None	Workload Manager (WLM) unit of work activities

Stored procedure service

The stored procedure service is assigned to the SPS service class. The task name for the SPS service class is CECSPS.

The stored procedure service runs a stored procedure program on behalf of a query processor service. Upon receiving a stored procedure request from a client application, the query processor service passes the request to the stored procedure connector. The stored procedure connector then coordinates its asynchronous processing with the stored procedure service which frees the query processor service to handle other requests while the stored procedure is running and improves performance.

The region controller automatically starts the stored procedure service on behalf of the stored procedure connector. When a stored procedure is called and a service configuration does not exist for service class SPS, the service is automatically defined and started. The service definition persists in the configuration of the Classic data server.

You can start, stop, or delete the stored procedure service.

- If the Classic data server is restarted with a persistent definition of the stored procedure service, the stored procedure service is started.
- If the service is stopped and a stored procedure service configuration exists, when a query processor issues a stored procedure call, the service is started automatically.
- If you delete the service and later call a stored procedure, the service is defined again because it is required to run the stored procedure.

Because the stored procedure service is automatically defined and administered when a stored procedure is called, you do not need to start and configure the

service. However defining, configuring, starting and stopping the stored procedure service is allowed. You can access the service definition in Classic Data Architect.

The following table summarizes the configuration parameters that define stored procedure services in the SPS service class.

Table 61. Configuration parameters for the SPS service class

Parameter	Default value	Description
IDLETIMEOUT	300000 MS (5 MIN)	The amount of time in milliseconds that a service remains idle before it polls the local message queue for messages that are to be processed
INITIALTHREADS	1	Number of instances of this service that the region controller starts during initialization of the Classic data server
MAXTHREADS	30	Maximum number of instances of this service that the region controller is allowed to start
MAXUSERS	20	Maximum number of user connections
RESPONSETIMEOUT	300000 MS (5 MIN)	Maximum amount of time in milliseconds to wait for response before terminating a connection
TRACELEVEL	4	Trace level
USERSUBPOOLMAX	262144	User pool maximum

VSAM access service

The VSAM access service is assigned to the VSMS service class. The task name for the VSMS service class is CACVSMS.

The VSAM service enables multiple users to share access to an open file.

Note: This service does not apply to Classic CDC for IMS.

The following table summarizes the configuration parameters that define VSAM access services in the VSMS service class.

Table 62. Configuration parameters for the VSMS service class

Parameter	Default value	Description
CLOSEONIDLE	FALSE	Close on idle
IDLETIMEOUT	5M	The amount of time that a service remains idle before it polls the local message queue for messages that are to be processed
INITIALTHREADS	1	Number of instances of this service that the region controller starts during initialization of the Classic data server
MAXTHREADS	1	Maximum number of instances of this service that the region controller is allowed to start
MAXUSERS	50	Maximum number of user connections
RESPONSETIMEOUT	5M	Maximum amount of time to wait for response before terminating a connection
SEQUENCE	0	Sequence number that is assigned to services
TRACELEVEL	4	Trace level

Workload Manager (WLM) service

The IBM z/OS WLM service is assigned to the WLM service class. The task name for the WLM service class is CACWLM.

The WLM service initializes the z/OS Workload Manager subsystem using the WLM system exit to enable query processing in WLM goal mode.

The following table summarizes the configuration parameters that define WLM services in the WLM service class.

Table 63. Configuration parameters for the WLM service class

Parameter	Default value	Description
EXITNAME	None	Name of the WLM exit to invoke
IDLETIMEOUT	0	the amount of time in milliseconds that a service remains idle before it polls the local message queue for messages that are to be processed
INITIALTHREADS	1	Number of instances of this service that the region controller starts during initialization of the Classic data server
MAXTHREADS	1	Maximum number of instances of this service that the region controller is allowed to start
MAXUSERS	10	Maximum number of user connections
RESPONSETIMEOUT	300000 MS (5 MIN)	Maximum amount of time in milliseconds to wait for response before terminating a connection
SEQUENCE	0	Sequence number that is assigned to services
SUBNAME	None	Name of the subsystem that the unit of work is reported under in WLM. This name and subsystem type are used to connect to WLM and classify work received.
SUBTYPE	None	Generic subsystem type under which the unit of work is reported in WLM
TRACELEVEL	4	Trace level

Configuration parameters for Classic data servers and services

Configuration parameters define settings for Classic data servers and for the services required for Classic data servers.

The global configuration parameters define server-wide settings. Standard configuration parameters define settings that are common to most services. All other configuration parameters are service-specific.

Global parameters for Classic data servers

Global parameters define configuration values that affect the entire Classic data server. Unlike other configuration parameters, global parameters are not related to a single service.

The following table summarizes the global configuration parameters and lists parameter default values.

Table 64. Global configuration parameters

Parameter	Default value	Description
DATACONVERRACT	0	Data conversion action
DATAVALIDATEACT	0	Data validation action
DECODEBUFSIZE	8192	Decode buffer size
FETCHBUFSIZE	32000	Size of the result set buffer returned to a client application
MESSAGEPOOLSIZE	16777216	Message pool size. This value is set during the installation and customization process.
REPORTLOGCOUNT	0	Log record limit for badly-formed data or conversion errors
STATICCATALOGS	0	Activates static catalog processing
TASKPARM	None	Specifies runtime options that are passed to subtasks through the IBM z/OS ATTACH macro

DATACONVERRACT:

The DATACONVERRACT parameter identifies the action for the Classic data server to take if a conversion error occurs when it converts numeric data between zoned and packed decimal formats and between binary and packed decimal formats.

Specification

Use: Global configuration parameter for the Classic data server.

Data type: INT

Default: 0 = FAIL

Valid values: 0 - 2

0 = FAIL

Specifies that the query ends with a -4908 return code that indicates non-valid mapped data.

1 = REPAIR

The Classic data server changes non-valid data to -99...99s, and the SQL statement ends successfully with a SQL_SUCCESS_WITH_INFO return code. One or more 002f0002 warning messages are also returned to the client application. A 002f0002 warning message is returned for each column each row that is changed to a value of -99..99s. The Classic data server does not write any log messages to indicate that conversion errors occurred.

Example: The Classic data server changes nonvalid data to the highest possible negative value for the column precision. A column with a precision of DECIMAL(3,0) changes to a value of -999, and a column with a precision of DEC(3,2) changes to a value of -9.99. If you sort the result set, rows with nonvalid data appear last.

2 = REPAIR REPORT

The Classic data server processes the conversion error as the REPAIR option

describes. The Classic data server also writes the data conversion error message x002f0001 to the server log for each row that contains one or more conversion errors.

DATAVALIDATEACT:

The DATAVALIDATEACT parameter enables additional validation of data types that are not converted from source data to a different SQL data type. This parameter controls whether additional data type validation occurs and identifies the action for the Classic data server to take if a validation error occurs due to badly-formed data.

Specification

Use: Global configuration parameter for the Classic data server.

Data type: INT

Default: 0 = NO VALIDATION

Valid values: 0 - 3

0 = NO VALIDATION

The data is not checked. This is the default

1 = REPAIR

The Classic data server changes non-valid data as described in the table below, and the SQL statement ends successfully with a SQL_SUCCESS_WITH_INFO return code. One or more 002f0002 warning messages are also returned to the client application. A 002f0002 warning message is returned for each column and each row that is changed. The Classic data server does not write any log messages to indicate that conversion errors occurred.

2 = REPAIR REPORT

The Classic data server processes the validation error as the REPAIR option describes. The Classic data server also writes the data conversion error message x002f0001 to the server log for each row that contains one or more conversion errors.

3 = FAIL

The query ends with a -4908 return code that indicates non-valid mapped data.

Table 65. Data type validation

Data type value	SQL data type	Data validated	Repair value
P	DECIMAL	Packed decimal data	-9
V	VARCHAR	Field length	0: For a negative length value Maximum length: If the length is greater than the maximum length of the field

Table 65. Data type validation (continued)

Data type value	SQL data type	Data validated	Repair value
VB	VARBINARY	Field length	0: For a negative length value Maximum length: If the length is greater than the maximum length of the field
UP	DECIMAL	Packed positive number	-9
UF	INTEGER	Value between 0 and X'7FFFFFFF'	-9
UH	SMALLINT	Value between 0 and X'7FFF'	-9

DECODEBUFSIZE:

DECODEBUFSIZE defines the size of the DECODE buffer. This buffer is a staging area that decodes data from the network format into the host local data format.

Description

Data is taken from the FETCH buffer in pieces that are the size that is specified for the DECODE buffer. The data is converted until a single row of data is completely processed and returned to the application. For optimum use, set the DECODE buffer to a size that is at least equivalent to a single row of data.

The DECODEBUFSIZE and FETCHBUFSIZE parameters work together. If the DECODEBUFSIZE is omitted, its value is set to the value of FETCHBUFSIZE. If a value higher than the FETCHBUFSIZE is used, the value of DECODEBUFSIZE is set to the FETCHBUFSIZE. Thus, coordinate the settings of the DECODEBUFSIZE and FETCHBUFSIZE parameters.

Specifications

Use: Global configuration parameter for the Classic data server.

Data type: INT

Default: 8192

Valid values: 4096 - 64000

FETCHBUFSIZE:

The FETCHBUFSIZE parameter specifies the size of the result set buffer that is returned to a client application. You specify this parameter in the configuration file for the client application.

Description

Regardless of the specified size of the fetch buffer, federation always returns a complete row of data in this buffer. When you set the fetch buffer size to 1, single rows of data are returned to the client application.

An appropriate `FETCHBUFSIZE` depends upon the average size of the result set rows that are sent to the client application and the optimum communication packet size. To improve performance, pack as many rows as possible into a fetch buffer. The default fetch buffer size is generally adequate for most queries.

If `FETCHBUFSIZE` is set smaller than a single result set row, the size of the actual fetch buffer that is transmitted is based on the result set row size. The size of a single result set row in the fetch buffer depends on the number of columns in the result set and the size of the data that is returned for each column.

The `FETCHBUFSIZE` and `DECODEBUFSIZE` parameters work together. If the `DECODEBUFSIZE` is omitted, its value is set to the value of `FETCHBUFSIZE`. If a value higher than the `FETCHBUFSIZE` is used, the value of `DECODEBUFSIZE` is set to the `FETCHBUFSIZE`.

You can use the following calculations to determine the size of a result set row in the buffer:

$$\text{fetch buffer row size} = (\text{number of data bytes returned}) \times (\text{number of columns} * 6)$$

Each fetch buffer has a fixed overhead. You can compute the overhead as follows:

$$\text{fetch buffer overhead} = 100 + (\text{number of columns} * 8)$$

If your applications routinely retrieve large result sets, contact your network administrator to determine the optimum communication packet size. Then, set the `FETCHBUFSIZE` to a size that accommodates large result sets.

Specifications

Use: Global configuration parameter for the Classic data server.

Data type: INT

Default: 32000

Valid values: 1 - 524288

MESSAGEPOOLSIZE:

The **MESSAGEPOOLSIZE** parameter specifies the size of the memory region in bytes for most memory allocation.

Description

Specify a region size that is at least 8 MB lower than the site limit, and use the greater of these values:

- 8 MB higher than the message pool
- 20% higher than the message pool

If the 8 MB gap between the region and the message pool is still not sufficient, increase this difference in increments of 8 MB.

Set the **MESSAGEPOOLSIZE** parameter to the greater of these values:

- 20% less than the region size
- 8 MB below the `REGION` value or 8 MB below any site limit imposed by exits.

If you increase the value of the **MESSAGEPOOLSIZE** parameter, set the region size higher to maintain the 8 MB gap.

Specification

Use: Global configuration parameter for the Classic data server.

Data type: INT

Default: 16777216

Valid values: 1048576 - upper limit not applicable

REPORTLOGCOUNT:

The **REPORTLOGCOUNT** parameter sets the maximum number of messages written to the log for badly-formed data or conversion errors.

Description

This parameter value prevents excessive logging to the log file for the Classic data server when a large amount of badly-formed data records are processed. The count controls the number of rows in the result set that generate the log messages for a given access to a table or view.

Specification

Use: Global configuration parameter for the Classic data server.

Data type: INT

Default: 0 = No logging limit

Valid values: 0 - 100000

STATICCATALOGS:

The **STATICCATALOGS** parameter activates static catalog processing for the system catalog data sets that are referenced by the Classic data server.

Description

With static catalog processing, the system catalog files are opened once for a query processor task. The system catalog files remain open until that Classic data server is shut down. In normal operating mode, the system catalogs are closed after the required table and column information is retrieved in order to process a query, for each query that is processed by the query processor.

Activate static catalog processing to substantially improve query performance in outer cursor and inner cursor situations when a large number of queries are issued serially.

Close the static catalog when the system is not updating catalogs information. Use this parameter when the Classic data server operates in production mode and the system catalogs are static.

Specification

Use: Global configuration parameter for the Classic data server.

Data type: INT

Default: 0

Valid values: 0 - 1

- 0 Close system catalog files and establish read locks for each query.
- 1 Close system catalog files when the Classic data server is shut down.

TASKPARM:

The TASKPARM parameter specifies IBM C runtime options that are passed to system child tasks through the z/OS ATTACH macro.

Description

One common use of this parameter is to pass TCP/IP information to the Communications Interface task. IBM Software Support can provide a current value.

Specifications

Use: Global configuration parameter for the Classic data server.

Data type: CHAR(64)

Default: None

Standard parameters for services

Standard service parameters are available in more than one service, but like service-specific parameters, you can define values separately for each service.

IDLETIMEOUT:

IDLETIMEOUT indicates idle time out.

Description

The IDLETIMEOUT value specifies the amount of time that a service remains idle before it polls the local message queue for messages that are to be processed.

Specification

Use: Configuration parameter for the Classic data server that applies to services that close connections after a specific time period.

Data type: TIME

Default values:

- 5M

Valid formats:

nMS *n* milliseconds

nS *n* seconds

nM *n* minutes

nH *n* hours

Setting the value to zero (0) indicates no time out. However, the setting the value to 0 is not recommended.

INITIALTHREADS:

INITIALTHREADS identifies minimum tasks.

Description

The value of INITIALTHREADS specifies the number of instances of this service that the region controller starts during initialization of the Classic data server. You can use the MTO START command to start an occurrence when needed, unless the service already has MAXTHREADS instances.

The default settings for this parameter are adequate for a default data server configuration. It is important to be cautious when you set the INITIALTHREADS parameter to a value that is greater than the default value. The region controller might not be able to start as many threads as the number specified due to system resource restrictions.

Specification

Use: Configuration parameter for the Classic data server that applies to all services.

Data type: INT

Default values:

- 1: For all services (exception: query processor service)
- 5: For the query processor service only

Valid values: 0 and above.

- 1** If a service must be limited to a single occurrence.
- 0** Indicates to the region controller that occurrences of this service must not be started during initialization of the Classic data server.

MAXTHREADS:

MAXTHREADS identifies the maximum number of instances for a service.

Description

The value of MAXTHREADS specifies the maximum number of instances of this service that the region controller can start.

The default settings for this parameter are adequate for a default data server configuration.

Specification

Use: Configuration parameter for the Classic data server.

Data type: INT

Default values:

- 1: For all services (exception: query processor service)
- 10: For the query processor service only

Valid values: 0 and above.

1 If a service must be limited to a single instance.

0 The currently deployed number of threads remains the same. A new instance is not started.

MAXUSERS:

MAXUSERS identifies the maximum number of connections per task.

Description

The MAXUSERS value is the maximum number of connections that are allowed per instance of this service. Set this field to 1 to disable multi-tasking for all instances of this service.

Specification

Use: Configuration parameter for the Classic data server that is common to all services.

Data type: INT

Default value: Varies by service.

1	ODBA
10	CAF DRA IMSI WLM
20	QP QPRR
50	DCI VSMS
100	CNTL INIT LOG MAA OPER

Valid values: 1 and above

RESPONSETIMEOUT:

RESPONSETIMEOUT specifies the maximum amount of wait time for an expected response.

Specification

Use: Configuration parameter for the Classic data server.

Data type: TIME

Default values:

- 5M: For all federation services (exception: ODBA service)
- 1M: For the ODBA service only

Valid formats:

nMS Number of milliseconds

nS Number of seconds

nM Number of minutes

If you set the RESPONSETIMEOUT value to zero (0), the response timeout function is disabled.

SEQUENCE:

Each service in the Classic data server is assigned a SEQUENCE number. The SEQUENCE parameter controls the order in which the region controller starts services in the Classic data server.

Description

The controller service starts first and is assigned a SEQUENCE value of 1. The logger service starts next and is assigned a SEQUENCE value of 2. You cannot modify the values of these core services.

When you add a service, that service is assigned the next available SEQUENCE value of 3 or higher. You can change the order in which services start by using SEQUENCE to reassign sequence numbers to services. You can modify these SEQUENCE values.

The non-core services can be assigned the same SEQUENCE number value. If this occurs, the services with the same SEQUENCE value will be started in alphabetical order based on service name.

SEQUENCE also affects termination processing for the Classic data server. Services stop in the reverse order that they start.

Specification

Use: Configuration parameter for the Classic data server that is common to all services.

Data type: INT

Default value: 0

Valid values:

- 1: Assigned to controller service
- 2: Assigned to logger service
- 3 - 999: Services are assigned the next available SEQUENCE value in the range of 3 - 999 when the service is added. You can modify these SEQUENCE values.

TRACELEVEL:

The TRACELEVEL parameter regulates the amount of information that tasks in the Classic data server record in the trace log.

Specification

Use: Configuration parameter that is common to all services.

Data type: INT

Default values:

- 4: For all services (exception: logger service)
- 1: For the logger service only. This value controls what other services send to the logger service and what the logger service writes to the log.

Valid values: 0 - 20:

- 20** No trace information generated
- 12** Identify non-recoverable error conditions
- 8** Identify all recoverable error conditions
- 4** Generate warning messages
- 3** Generate debugging information
- 2** Generate a detailed trace, but do not include binary buffers.
- 1** Generate function call information
- 0** Trace all

Important: Change this parameter only at the request of IBM Software Support. Settings lower than 4 cause response time degradation and higher CPU costs.

Service-specific parameters

Service-specific parameters pertain to a single service, so the values that you define affect only that service.

This section provides an explanation of each configuration parameter. “Summary of services” on page 299 summarizes the configuration parameters associated with each service.

BTREEBUFFS:

The BTREEBUFFS parameter determines the number of B-tree buffer caches in memory that are used before spooling the staged result set to hiperspaces or to physical files.

Description

You can set BTREEBUFFS to override the default value of four. If sufficient memory is available in the MESSAGEPOOLSIZ memory pool, this parameter can be increased, and performance might improve, depending on the size of the result set and whether the result set is ordered or grouped.

Specification

Use: Configuration parameter for the QP and QPRR services.

Data type: INT

Default value: 4

Valid values: 4 - 214730

COMMSTRING:

The COMMSTRING parameter specifies the protocol identifier and address for communication.

Description

The connection handler supports TCP/IP and XM protocols. The COMMSTRING value defines the protocol followed by the protocol specific information

- For developing remote client applications with a TCP/IP connection handler, you need the protocol identifier TCP, followed by the IP address of the machine that the Classic data server is running on, and the port number that is assigned to this server as a listen port. For example: TCP/*host-name/port-number*.

The client connection string supports Internet Protocol Version 4 (IPv4) and Internet Protocol Version 6 (IPv6). For example:

```
SET,CONFIG,SERVICE=INIT,COMMSTRING='TCP/0.0.0.0/9087'; (IPv4)
SET,CONFIG,SERVICE=INIT,COMMSTRING='TCP>::/9087'; (IPv6)
```

If you try to connect to the data server by using IPv6, you might need to provide a scope if you are using a link-local address. The scope is typically the network interface name that you specify following the IPv6 address. The format is *ipv6 address%scope*. For example:

```
SET,CONFIG,SERVICE=INIT,COMMSTRING='TCP/fe80::xxxx:xxxx:xxxx:xxxx%INTF0001/9087';
```

- For developing client applications on the same z/OS system with a connection handler that uses cross-memory services, you define a unique dataspace name and queue name. For example: XM1/*Dataspace-Name/Queue-Name*. XM1 is the cross-memory protocol identifier, *Dataspace-Name* is the name of the dataspace to use, and *Queue-Name* is the name of the XM queue. The dataspace or queue name can contain up to 4 characters.

Specification

Use: Configuration parameter for the connection handler service.

Service class: INIT

Service task: CACINIT

Data type: CHAR(64)

Default: TCP/0.0.0.0/9087

CONNECTINTERVAL:

The CONNECTINTERVAL parameter defines the frequency at which the DRA service retries connecting to IMS when IMS is not available.

Description

The default value for this parameter is 15S (seconds). This value indicates that the DRA service will retry failed connections to IMS every 15 seconds.

Specifications

Use: Configuration parameter for the DRA service.

Service class: DRA

Service task: CACDRA

Data type: TIME

Default: 15S

Valid values:

nMS *n* milliseconds

nS *n* seconds

nM *n* minutes

nH *n* hours

CONSOLELEVEL:

CONSOLELEVEL is a required parameter that controls when event messages are sent to the z/OS console.

Description

All event messages are written to the event log and the system trace. The value of the CONSOLELEVEL parameter controls when event messages are also routed to the z/OS console. When the trace level of an event message equals or exceeds the value specified for the CONSOLELEVEL parameter, that message is sent to the console.

Specifications

Use: Configuration parameter for the logger service.

Service class: LOG

Task name: CACLOG

Data type: INT

Default: 4

Valid values:

- 20** Generates no event messages to the console. Messages are written to the event log.
- 4** Generates the following event messages:
- Stream activation and destruction messages issued from the log reader service
- Specifying a value less than 4 generates more console messages and increases the number of messages in the console buffers.
- 3** Generates the following event messages:
- Table, view, and DBMS object cache operation messages issued from the administration service
 - SSID start and stop messages issued from the IMS log reader service
- 0** Writes all event messages to the console.

CPUGOVERNOR:

The CPUGOVERNOR parameter specifies the name and the time limit for the exit to implement a CPU resource governor. If this parameter is omitted, there is no limit to the amount of CPU time for query processing.

Specifications

Use: Configuration parameter for the query processor services.

Service classes: QP, QPRR

Service tasks: CACQP, CACQPRRS

Data type: CHAR(64)

Default: None

DEFAULTPSBNAME:

The DEFAULTPSBNAME parameter identifies a default PSB name.

Description

The PSB name is used when a CREATE TABLE statement references an IMS table that contains no PSB name.

Specification

Use: Configuration parameter for the DRA access service.

Service class: DRA

Task name: CACDRA

Data type: CHAR(8)

Default value: NOPSB

DISPLAYLOG:

The DISPLAYLOG parameter allows you to view log messages for the logger service.

Description

This parameter controls whether log records are mirrored to the data set that is specified in the SYSOUT DD statement. The default data set is the system output data set (SYSOUT).

Specifications

Use: Configuration parameter for the logger service.

Service class: LOG

Task name: CACLOG

Data type: Boolean

Default: FALSE

DRATABLESUFFIX:

The DRATABLESUFFIX parameter identifies the suffix of the DRA startup table.

Description

Use DRATABLESUFFIX to specify the suffix of the load module name that you created for IMS DRA initialization.

Specification

Use: Configuration parameter for the IMS DRA access service.

Service class: DRA

Task name: CACDRA

Data type: CHAR(3)

Default value: None

DRAUSERID:

The DRAUSERID parameter identifies the DRA user ID.

Description

Use DRAUSERID to specify the default DRA user ID to use for connecting to and registering with DBCTL. The DRA user ID is the name by which the Classic data server is known to the IMS database manager subsystem, DBCTL.

Specification

Use: Configuration parameter for the IMS DRA access service.

Service class: DRA

Task name: CACDRA

Data type: CHAR(9)

Default value: None

EVENTLOG:

EVENTLOG is an optional parameter identifies the name of the event message log that is defined to the logger service.

Description

The logger service writes event messages to the log specified in the EVENTLOG parameter. This parameter identifies a z/OS system log stream. The event log file cannot be shared among multiple Classic data servers. You can use one event log file with one Classic data server.

If you do not specify the EVENTLOG parameter, event messages are not captured. Event messages will not be available for retrieval by the Classic Data Architect. In this case, the Classic data server formats event messages to SYSPRINT and incurs processing overhead at runtime.

Specifications

Use: Configuration parameter for the logger service.

Service class: LOG

Task name: CACLOG

Data type: CHAR (26)

Default: None

HOSTCODEPAGE:

The HOSTCODEPAGE parameter identifies the CCSID that the z/OS operating system uses where the data server is running.

Specification

Use: Configuration parameter for the region controller service.

Service class: CNTL

Service tasks: CACCNTL

Data type: INT

Default: 37

Valid values: 0 - 99999

JOINTABLES:

The JOINTABLES parameter determines the optimization method in queries that contain join operations.

Description

The estimated reads analysis evaluates all possible combinations of table processing order. The number of possible combinations for analysis is a factorial of the number of tables in a join.

Recommendation: To optimize performance, use a value no greater than 4. For a value of 4, the count is factorial 4, or 24 (1 * 2 * 3 * 4). For a value of 5, the count is factorial 5, or 120 (1 * 2 * 3 * 4 * 5). The latter value specifies that the estimated reads analysis evaluate 120 table combinations.

Join queries that include more than the specified number of tables automatically use the simple optimization method. The maximum value is 15 because the maximum number of tables in a SQL statement is 15. While the parameter has no fixed limit on the value, the query processor treats any higher value as equivalent to 15.

Specification

Use: Configuration parameter for the query processor services.

Service classes: QP, QPRR

Service tasks: CACQP, CACQPRRS

Data type: INT

Default: 4

Valid values: 0 - 15

- 0** Bypasses the join optimization method and the query is processed as is.
- 1** Indicates that the simple join optimization method is applied to all joins.

Greater than 1

Identifies the maximum number of tables in the join that are evaluated by the estimated reads analysis.

LOGBUFSIZE:

The LOGBUFSIZE parameter defines the size of the log buffer.

Specifications

Use: Configuration parameter for the logger service.

Service class: LOG

Task name: CACLOG

Data type: INT

Default: 65536

Valid values: 4096 - 1024000

LOGURL:

The LOGURL parameter identifies the communication protocol for the logger service.

Description

You can use LOGURL to override the protocol defined for local queues. This parameter is typically used for XM queues.

Specifications

Use: Configuration parameter for the logger service.

Service class: LOG

Task name: CACLOG

Data type: CHAR(32)

Default: None

Example

The following sample command sets the value of the **LOGURL** parameter for a logger service with the name LOG.

```
F <Data-Server-Name>,SET,CONFIG,SERVICE=LOG,LOGURL=XM1/DSLGI/LOGQ1/256
```

MAXROWSEXAMINED:

The MAXROWSEXAMINED parameter implements the governor.

Description

MAXROWSEXAMINED provides protection from excessive resource use that inefficient or erroneous queries cause. The governor limits the number of examined rows. In the examination phase, a restriction on the number of examined rows is put into effect after native retrieval is performed and before additional filtering takes place to satisfy any WHERE clause specifications.

In general, you should set the value of MAXROWSEXAMINED to the default value or to meet the requirements of the table size. You can use a larger value of roughly 10000 in a test environment.

Specification

Use: Configuration parameter for the query processor services.

Service classes: QP, QPRR

Service tasks: CACQP, CACQPRRS

Data type: INT

Default: 0

Valid values: 0 - 2147483647

MAXROWSEXCATION:

The MAXROWSEXCATION parameter determines the behavior of the governor when the MAXROWSEXAMINED or the MAXROWSRETURNED governor limits are reached.

Description

The MAXROWSEXCATION parameter tests a query to ensure that the query returns the correct result set and does not expend large amounts of resources. This test is performed before the full query is run.

Specification

Use: Configuration parameter for the query processor services.

Service classes: QP, QPRR

Service tasks: CACQP, CACQPRRS

Data type: INT

Default: 1

Valid values: 0 - 1

Valid values and results:

0 = ABORT

Stops the query when a governor limit for MAXROWSEXAMINED or MAXROWSRETURNED is reached. The -9999 return code is issued.

1 = RETURN

Returns a normal result set when a governor limit is reached. A truncated result set is returned to the client. The result might not be complete, and there is no indication that the governor limit is reached.

MAXROWSRETURNED:

The MAXROWSRETURNED parameter specifies the maximum number of rows a query can return to the client. The governor imposes the restriction you specify after any WHERE clause is fully processed, but before the result table is returned to the application.

Description

MAXROWSRETURNED provides protection from excessive resource use that inefficient or erroneous queries cause.

Specifications

Use: Configuration parameter for the query processor services.

Service classes: QP, QPRR

Service tasks: CACQP, CACQPRRS

Data type: INT

Default: 0

Valid values: 0 - 2147483647

NMICOMMSTRING:

The NMICOMMSTRING parameter specifies the communication path for the Network Management Interface (NMI) AF_UNIX domain socket.

Description

The value of the NMICOMMSTRING configuration parameter defines the fully-qualified path and file name of the AF_UNIX domain socket that is used for NMI client connections to the NMI. You specify this value in the following format:

```
NMICOMMSTRING='/var/sock/uniqueName'
```

where '/var/sock/uniqueName' is the path and unique name of the file. The maximum length that you can specify is 60 characters.

Recommendation: Use single quotes to preserve the case (uppercase or lowercase) of the characters in the AF_UNIX domain socket path. The case of the characters specified for this value must match the case of the Unix System Services file system path.

You can dynamically set or change the value of the NMICOMMSTRING configuration parameter by using the Classic Data Architect or the MTO SET,CONFIG,SERVICE command. The change takes affect when you stop and restart the monitoring service or the Classic data server.

Specification

Use: Configuration parameter for the monitoring service.

Service class: MAA

Service task: CECMAA

Data type: CHAR

Default: None

RECONNECTWAIT:

The RECONNECTWAIT parameter defines the minimum amount of time that the DRA service waits after an IMS disconnect before attempting to reconnect to IMS.

Description

This wait time ensures that the DRA service does not try to reconnect to IMS while IMS is still in the process of stopping and prevents abends in the DBCTL interface.

The RECONNECTWAIT parameter is enforced when the console message CAC00137W is issued to ensure that IMS has enough time to stop before the DRA service attempts to reestablish a connection.

The default value for this parameter is 1M (minute). This value indicates that the DRA service will start trying to connect to IMS one minute after a disconnect is received from IMS and message CAC00137W is issued.

Specifications

Use: Configuration parameter for the DRA service.

Service class: DRA

Service task: CACDRA

Data type: TIME

Default: 1M

Valid values:

nMS *n* milliseconds

nS *n* seconds

nM *n* minutes

nH *n* hours

SAFEXIT:

The SAFEXIT parameter specifies the System Authorization Facility (SAF) system exit that performs authorization checks for the monitoring and console connections to the Classic data server.

Description

The values that you specify for the SAFEXIT parameter control the actions that a user can perform when connected to a Classic data server for the following types of connections:

- Monitoring connections established by the Classic Data Architect. These connections are authenticated at the z/OS host. Monitoring connections also include Network Management Interface (NMI) connections to the z/OS data server. You use NMI connections to retrieve metrics data and for access to subscription states or statuses.
- Console connections from the Classic Data Architect that allow remote operators to issue console commands to the Classic data server.

- Query processor connections that use the SAF exit to authenticate users who map tables and run test queries.

You specify the optional parameters for the SAF exit in the following format:

`CACSX04{,optional-parameters...}`

If you do not specify any optional parameters, the SAF exit load module CACSX04 activates user ID and password authentication when a user connects to a Classic data server. The optional parameters provide additional security checking that the monitoring service, operator service, and query processor service perform.

All connections

The following optional parameter for validation of IP addresses applies to all connections:

NETACCESS=Y/N

Indicates whether the exit should validate the IP address of the connected client to authenticate access to the Classic data server.

Set the value to Y when the IP address of the connected client is known and the SERVAUTH parameter of the RACROUTE REQUEST=VERIFY invocation is supplied. The RACROUTE operation is successful when the associated user ID has at least READ-level access rights to the network security zone resource. If the security system indicates that it cannot make a decision in response to the request because a corresponding network security zone resource profile does not exist, the SAF exit regards the response as Access Denied.

A value of N indicates that the SERVAUTH parameter is omitted from the RACROUTE REQUEST=VERIFY invocation. This is the default.

Monitoring connections

The following optional parameters for the monitoring service activate security checking for client connections from the Classic Data Architect:

VALIDATE=Y/N

Indicates whether the SAF exit should perform resource class checking for each connected user. If Y is specified, resource access checking occurs when users make requests to retrieve metrics information. READ level access is checked. If N is specified, resource class checking is not performed. This is the default.

MONCLASS=*monitor-class-name*

Indicates the name of the security class that contains a profile that requires access authentication.

This parameter is valid if VALIDATE=Y on the monitoring service for the SAF exit. If this parameter is not specified, SERVAUTH is the default class name.

MONPROF=*monitor-profile-name*

Indicates the name of the resource profile that requires access authentication.

This parameter is valid if VALIDATE=Y on the monitor service for the SAF exit. If this parameter is not specified, CEC.MONITOR is the default profile name.

Console connections

The following optional parameters for the operator service activate security checking for client connections from the Classic Data Architect for issuing console commands:

VALIDATE=Y/N

Indicates whether the SAF exit should perform resource class checking for each connected user. If Y is specified, CONTROL level access is checked when users issue console commands through the remote operator. If N is specified, resource class checking is not performed. This is the default.

OPERCLASS=operator-class-name

Indicates the name of the security class that contains a profile that requires access authentication.

This parameter is valid if VALIDATE=Y on the operator service for the SAF exit. If this parameter is not specified, SERVAUTH is the default class name.

OPERPROF=operator-profile-name

Indicates the name of the resource profile that requires access authentication.

This parameter is valid if VALIDATE=Y on the operator service for the SAF exit. If this parameter is not specified, CEC.OPER is the default profile name.

Query processor connections

The following optional parameters for the query processor service activate authorization checks for data server connections.

VALIDATE=Y/N

Indicates whether the exit should validate that the user ID has authority to access a specified database, file, stored procedure, or PSB name. Use both SQL security and the SAF exit in conjunction with your site security package to restrict access to your data.

Set a value of Y to use RACF security and issue RACROUTE validation calls for specified resource names. Set a value of N to suppress validation processing for resources. The default value for VALIDATE is Y.

This parameter helps you to control access with greater precision:

- Ensure that only valid users can access resources by setting VALIDATE=Y against the QP or QPRRS service.
VALIDATE=Y authenticates each individual resource in the QP.
- Eliminate the overhead of verifying that the user has authority to access a resource by setting VALIDATE=N against the QP or QPRRS service.

Do this if you have elected to use SQL security to control access to tables and stored procedures. This setting is also useful in a test or development environment if you trust any user with a valid z/OS user ID to access the data. For example, you might not want to use DB2 privileges or use RACF to verify each resource.

IMS CLASS=Class Name

Specifies the name of the RACF resource class that is checked to determine whether the user has authority to schedule or access the PSBs associated

with the tables referenced in a query. The Class Name can be up to eight characters long. This sub-parameter is required when accessing IMS data.

PSB PREFIX=Prefix

Specifies a value to prefix to the PSB names before a RACF authorization call is issued. If specified, the PSB name is appended to the Prefix value, for example, IMSPPSB1 where the Prefix is IMSP and the PSB name is PSB1.

If you are planning to access IMS data, you might need to modify the IMS CLASS subparameter to define the RACF class where IMS PSBs are defined at your site.

To use a PSB, a user ID must have at least CONTROL access to that PSB's corresponding RACF profile within the class.

The combination of the length of the PSB name and the length of the prefix must be eight characters or less. This is a RACF restriction. If a larger PSB name or prefix combination is encountered, an error message is issued.

ADACLASS=Facility

Specifies the name of a class to be used to check for authorized use of ADABAS view names. An ADACLASS name can be up to eight characters long. In this example the IBM-supplied class of FACILITY is used. You can define an installation class specifically for ADABAS views. Use the FACILITY class as an example and specify the length of the resource name as 32. Replace FACILITY with the name of the new class. The security administrator must define each ADABAS view name as a resource in the class and grant CONTROL access to each user ID that uses that view name. If the ADABAS view name is not defined, or the user ID is not granted access, the following message is returned on the attempt to pull data from the ADABAS table defined with a view name:

Access Denied

If the ADABAS table is only defined with a file number (no Predict view name), you will receive the same error message as shown above, and the server log contains the following message:

CACL010E NO ADABAS VIEW NAME IN USE GRAMMAR

SPCLASS=FACILITY

Specifies the name of a class to be used to check for RACF-authorized use of stored procedure names that are defined in the metadata catalogs. These names are stored in the SYSIBM.SYSROUTINES system table. An SPCLASS name can be up to eight characters in length.

This example uses the IBM-supplied class of FACILITY. You can define an installation class specifically for stored procedures to RACF. Use the FACILITY class as an example and specify the length of the resource name as 39. Replace FACILITY with the name of the new class.

The RACF administrator must define each stored procedure as a resource in this class and grant ALTER access to each user ID that will invoke the stored procedure. If the stored procedure is not defined to RACF or the user ID is not granted access, -5046295 is returned on the attempt to use a stored procedure and message CACL006W is issued.

EXCLUDE=n

Indicates the query processor should not provide an ACEE address in commands sent to CA-Datacom. When the SAF exit is active, the address of an ACEE is obtained during SAF exit initialization. This ACEE address

is normally passed to CA-Datcom in each database request and CA-Datcom authenticates the request using information within the ACEE.

Whenever the SAF exit is active and you want to avoid database level security checking in CA-Datcom, you must indicate the query processor should exclude the ACEE from the database requests that are sent to CA-Datcom. Set the value of *n* to 2 (heterogeneous query processor CA-Datcom connector). This setting will not provide the ACEE address in the call parameters.

Specification

Use: Configuration parameter for the monitoring, operator, query processor, and two-phase commit query processor services.

Service classes: MAA, OPER, QP, QPRR

Service tasks: CECMAA, CACOPER, CACQP, CACQPRRS

Data type: CHAR

Default: None

SMFEXIT:

SMFEXIT reports clock time and CPU time for an individual user session with a query processor task.

Description

You can supply the following values for the SMFEXIT parameter:

RECTYPE=*nnn*

This is a required parameter that defines the SMF user record type. This parameter contains a numeric value between 128 and 255.

SYSID=*xxxx*

This is a required parameter that contains the primary JES subsystem ID. SYSID can be a maximum of four characters.

Specification

Use: Configuration parameter for the operator and query processor services.

Service classes: OPER, QP, QPRR

Service tasks: OPER, CACQP, CACQPRRS

Data type: CHAR(64)

Default value: None

SQLSECURITY:

The SQLSECURITY parameter determines the level of access privilege verification that will be performed on operator commands.

Description

When `SQLSECURITY` is set to `TRUE`, user access privileges are checked in the system catalog for `DISPLAY` and `SYSOPER` privileges. If no privileges are found, operator requests are not allowed in the Classic data server to which the user is connected.

Specification

Use: Configuration parameter for the command operator services.

Service class: `OPER`

Service task: `CACOPER`

Data type: `Boolean`

Default: `FALSE`

Valid values:

FALSE

Authenticated users can enter all valid commands. Valid commands include the `STOP` command that you can issue to stop the Classic data server. This setting also allows the Classic data server to run without defining a set of metadata catalogs.

TRUE Allows different levels of access level privileges for each user. For example, you can allow all users to issue the `DISPLAY` command but allow only a subset of users to issue the `STOP` command. If `TRUE` is specified, the server `JCL` must contain `DD` statements for the metadata catalogs.

STMTRETENTION:

The `STMTRETENTION` parameter defines the behavior of a prepared statement when a commit or rollback operation occurs.

Specification

Use: Configuration parameter for the query processor services.

Service classes: `QP`, `QPRR`

Service tasks: `CACQP`, `CACQPRRS`

Data type: `INT`

Default: `0`

Valid values: `0 - 2`

0 = SYNCPOINT

Release the statement whenever a `COMMIT` or `ROLLBACK` is issued.

1 = ROLLBACK

Release the statement only when a `ROLLBACK` syncpoint is issued.

2 = DISCONNECT

Release the statement only when the user disconnects from the Classic data server. All prepared statements are retained across both COMMIT and ROLLBACK calls.

STREAMNAME:

The STREAMNAME parameter identifies the name of the log stream that is defined in the z/OS system logger.

Description

The logger service writes log records to the z/OS log stream specified in the STREAMNAME parameter. The log stream is used for the diagnostic log that runs in the Classic data server.

Specifications

Use: Configuration parameter for the logger service.

Service class: LOG

Task name: CACLOG

Data type: CHAR(26)

Default: None

TEMPFILESPEC:

TEMPFILESPEC defines a temporary data set that is dynamically allocated by a Classic data server to store the intermediate result set. How you define the TEMPFILESPEC value varies by service.

Description: QP, QPRR

Temporary data set information is a set of parameters separated by commas. Parameters not specified are set to the defaults. Set this parameter so that the resulting file is large enough to hold any intermediate result sets that are generated from a typical query that runs on a particular Classic data server. If your site has a storage unit name for VIO storage, specify VIO.

Hiperspace places temporary data files, such as spill files, in expanded storage. Hiperspace improves performance and mainly affects complex queries, for example, queries that contain the ORDER BY clause. Hiperspace requires Authorized Program Facility (APF) authorization.

Specification

Use: Configuration parameter for the query processor services.

Service classes: QP, QPRR

Service tasks: CACQP, CACQPRRS

Data type: CHAR(64)

Default value: HIPERSPACE,INIT=8M,MAX=2048M,EXTEND=8M

Valid values:

ALCUNIT = BLOCK|TRK|CYL

Specifies a unit of space allocation in block, track, or cylinder units. The default value is TRK.

SPACE = bytes

Specifies the primary amount of space to allocate. The default value is 15.

EXTEND = bytes

secondary amount of space to allocate. The default value is 5.

VOL = VOLSER

Specifies the volume serial number. The default is the z/OS default for your site.

UNIT = unit name

Specifies a DASD allocation group name or the VIO group name, if it exists. The default unit name is the z/OS default for your site.

RECFM = F|V|U

Specifies the record format to allocate that corresponds to a z/OS Record Format (RECFM) of FB, VB, or U. The default is V.

RECLEN = nnn

Specifies the record length. For variable format records, z/OS LRECL (maximum record length) is set to the fixed record length RECLEN +4. Default is 255.

BLKSIZE = nnn

Specifies the block size. The default is 6144.

Example: DASD

Here are example entries for DASD:

TEMPFILESACE = ALCUNIT=TRK,SPACE=15,VOL=CACVOL

TEMPFILESACE = ALCUNIT=CYL,SPACE=2

TEMPFILESACE = ALCUNIT=CYL,SPACE=2,EXTEND=1,UNIT=VIO

Specification for hiperspace

Valid values:

INIT

Initial region size for the hiperspace

MAX

Maximum region size for the hiperspace.

EXTEND

Unit of growth when INIT is exceeded

In general, you should specify these sizes in megabytes (for example, 8M). You can use other units for the query processor.

The estimate for determining these values is related to system installation limits and expected query types. Roughly, make the maximum size equivalent to that of the regular temporary space file as described for the non-hiperspace TEMPFILESACE setting.

Example: hiperspace

To specify hiperspace, specify the `TEMPFILESSPACE` parameter as follows:

```
TEMPFILESSPACE = HIPERSPACE,INIT=16M,MAX=24M,EXTEND=8M
```

USERSUBPOOLMAX:

The `USERSUBPOOLMAX` parameter determines the maximum size of a user sub pool.

Description

Each user that connects to a Classic data source is assigned a user sub pool. A query processor task is assigned to a user connection. The `USERSUBPOOLMAX` parameter limits the size of the memory pool that a query processor task can use to optimize the use of system resources.

A user sub pool can grow to 256 times the `USERSUBPOOLMAX` value, resulting in the maximum user sub pool size in bytes. For example, if you set `USERSUBPOOLMAX` to the default value of 8192, the memory requirements for all of your queries for the current connection cannot exceed 2MB.

Recommendation: For complex queries, or in environments that process many queries simultaneously per user connection, set `USERSUBPOOLMAX` to a value greater than 8192. If you have many users simultaneously connecting to the data source, you might need to increase the value of the `MESSAGEPOOLSIZE` parameter because each user sub pool is allocated out of the main message pool.

Configure this parameter carefully to avoid using too much data server storage.

Specification

Use: Configuration parameter for the QP and QPRR service classes.

Data type: INT

Default value: 8192

Valid values: 4 - 4294967296

VSAMAMPARMS:

`VSAMAMPARMS` supplies VSAM buffer and memory-tuning parameters when a VSAM file is opened. The `VSAMAMPARMS` parameter specifies tuning parameters that are applied to all VSAM files that are opened when a single cursor is open.

Description

The `VSAMAMPARMS` parameter takes the form of a string of comma delimited parameters that are passed to the IBM C `afopen` call that accesses VSAM files.

Specification

Use: Configuration parameter for the query processor services.

Service classes: QP, QPRR

Service tasks: CACQP, CACQPRRS

Data type: CHAR(256)

Default value: None

Valid values:

BUFND

Specifies the number of data I/O buffers that VSAM uses. This parameter is equivalent to coding the BUFND value on a DD statement. A data buffer is the size of a control interval in the data component of a VSAM cluster. The default number of data buffers is the number of strings plus one. If you use the VSAM service, the default number of buffers is 11. If you do not use the VSAM service, the default number of buffers is two.

Generally with sequential access, the optimum value for the data buffers is six buffers or the size of the control area, whichever is less. When skip-sequential processing (random keyed read access) is performed, specifying two buffers is optimum. Specify a larger BUFND value when the VSAM file is scanned during query processing. The larger value generally yields performance improvements. In keyed access situations, specifying a larger BUFND might not show performance improvements or might degrade query performance by tying up large amounts of virtual storage and causing excessive paging.

BUFNI

Specifies the number of index I/O buffers that VSAM uses. This parameter is equivalent to coding the BUFNI value on a DD statement. An index buffer is the size of a control interval in the index component of a keyed VSAM cluster. If you use the VSAM service, the default number of index buffers is 10. If you do not use the VSAM service, the default number of index buffers is 1.

For keyed access, the optimum BUFNI specification is the number of high-level (non-sequence set) index buffers + 1. You can determine this number by subtracting the number of data control areas from the total number of index control intervals within the data set. You can accommodate most VSAM files having reasonable index control interval and data control area sizes using an upper bound BUFNI specification of 32. This BUFNI setting can accommodate cylinder-allocated data component sizes up to the 4 GB maximum. A large BUFNI value incurs little or no performance penalty, unless the value is excessively large.

BUFSP

Specifies the maximum number of bytes of storage that VSAM uses for file data and index I/O buffers. This parameter is equivalent to coding the BUFSP value on a DD statement. A data or index buffer is the size of a control interval in the data or index component.

A valid BUFSP specification generally overrides any BUFND or BUFNI specification. However, the VSAM rules for specifying an optimum BUFSP value are fairly complex. Consult the information about the ACB macro to determine the rules for specifying a BUFSP value.

Example

```
VSAMAMPARMS = BUFND=20,BUFNI=15
```

WLMUOW:

The WLMUOW parameter specifies the Workload Manager (WLM) unit-of-work activities and manages queries in WLM goal mode.

Description

Define a subsystem type and classification rules for the workload that is processed by the query processor. For an existing subsystem type, select which of these parameters fits that subtype. For example, the z/OS Started Task Control (STC) type supports user ID and TRANNAME. Job Entry Subsystem (JES) supports user ID, TRANNAME, and TRANCLASS.

For information about how to define service classes and classification rules, see the description of z/OS Workload Manager in the CICS product information for WLM goal mode. The priority for units of work need to be less than VTAM and IMS. The discretionary goal might result in very slow response times. Performance periods allow you to define a high number of service units for short transactions and a smaller number for long running transactions.

Specification

Use: Configuration parameter for the query processor services.

Service classes: QP, QPRR

Service tasks: CACQP, CACQPRRS

Data type: CHAR(64)

Default: None

Configuration parameters for clients

The client configuration parameters define settings for the JDBC, ODBC, and Call Level Interface (CLI) clients.

The following table lists the client configuration parameters and whether the parameter is required or optional.

Table 66. Summary of client configuration parameters

Parameter	Required/Optional
CLIENT CODEPAGE	Optional
DATASOURCE	Required
DEFLOC	Optional
FETCH BUFFER SIZE	Optional
MESSAGE POOL SIZE	Optional
NL CAT	Required
RESPONSE TIME OUT	Optional
SERVER CODEPAGE	Optional
SERVICE INFO ENTRY	Required
SHAPING	Optional

Table 66. Summary of client configuration parameters (continued)

Parameter	Required/Optional
SYMMETRIC SWAPPING	Optional
TASK PARAMETERS	Optional
TEXT ORIENTATION	Optional
TEXT PRESENTATION	Optional
TRACE LEVEL	Optional

CLIENT CODEPAGE

CLIENT CODEPAGE is an optional parameter that specifies the client code page value so that ICU4C can translate between the code pages for the server and client.

Description

This parameter corresponds to the code page converter names and aliases for the CCSID that are used on the client and on the server. ICU4C provides conversion between code pages.

Specifications

Use: UNIX CLI client

Example

```
CLIENT CODEPAGE = IBM-970
```

DATASOURCE

DATASOURCE is a required parameter for clients that specifies the name of the data source that a client attempts to connect to.

Description

- Field 1: The name of the remote data source that matches the service name of the service definition for the query processor.
- Field 2: The address identifies how this client connects to the named data source. This field consists of three parts separated by the forward slash (/) character. This value must match the service definition of the for the connection handler on the data server.

Specification

Maximum value: 18 characters for data source name, 64 characters for address field

Minimum value: 1 character for data source name, address field depends on the protocol

Default: None

Use: UNIX CLI client

Example: Address field for TCP/IP protocol

```
DATASOURCE = CACSAMP tcp/111.111.111.11/2222
```

This example defines an address field for TCP/IP protocol that uses the data source name CACSAMP. Field 2 contains the following parts:

- The first part of the field must be set to tcp.
- The second part of the field is the host name (string) of the server or the IP address of the server. If an IP address is specified, it must be defined in dot notation (123.456.789.10).
- The third part of the field is the port number (decimal value) or service name on which the server is listening for connection requests.

If you try to connect to the data server by using IPv6, you might need to provide a scope if you are using a link-local address. The scope is typically the network interface name that you specify following the IPv6 address. The format is ipv6 address%scope. For example:

```
DATASOURCE = CACSAMP tcp/fe80::xxxx:xxxx:xxxx:xxxx%eth0/9087
```

Example: Address field for cross-memory protocol

```
DATASOURCE = CACSAMP XM1/CAC/CAC
```

This example defines an address field for the cross-memory protocol that uses the data source name CACSAMP. field 2 contains these parts:

- The first part of the field must be set to XM1.
- The second part of the field is the data space name, CAC. The maximum length is four characters. This name must be the same as the data space name defined for the COMMSTRING parameter of the connection handler service.

If the data server was configured to directly connect to a query processor without going through a connection handler service, the protocol string is the service name for the query processor. In this case, the data space name must match the name of the data space specified in the service definition of the query processor.

- The third part of the field is the queue name, CAC. The maximum length is four characters. This name must be the same as the queue name defined for the COMMSTRING parameter of the connection handler service.

If the data server was configured to directly connect to a query processor without going through a connection handler service, the protocol string is the service name for the query processor. In this case, the queue name must match the name of the queue specified in the service definition of the query processor.

DEFLOC

DEFLOC is an optional parameter that specifies the default data source if a SELECT statement or a CONNECT statement does not specify where the data resides.

Specification

Maximum value: 18 characters

Minimum value: 1 character

Default: None

Use: Client

Example

```
DEFLOC = CACSAMP
```


FETCH BUFFER SIZE

FETCH BUFFER SIZE is an optional parameter that specifies the size of the result set buffer that is returned to a client application. You specify this parameter in the configuration file for the client application.

Description

Regardless of the specified size of the fetch buffer, federation always returns a complete row of data in this buffer. When you set the fetch buffer size to 1, single rows of data are returned to the client application.

An appropriate FETCH BUFFER SIZE depends upon the average size of the result set rows that are sent to the client application and the optimum communication packet size. To improve performance, pack as many rows as possible into a fetch buffer. The default fetch buffer size is generally adequate for most queries.

If FETCH BUFFER SIZE is set smaller than a single result set row, the size of the actual fetch buffer that is transmitted is based on the result set row size. The size of a single result set row in the fetch buffer depends on the number of columns in the result set and the size of the data that is returned for each column.

You can use the following calculations to determine the size of a result set row in the buffer:

$$\text{fetch buffer row size} = (\text{number of data bytes returned}) \times (\text{number of columns} * 6)$$

Each fetch buffer has a fixed overhead. You can compute the overhead as follows:

$$\text{fetch buffer overhead} = 100 + (\text{number of columns} * 8)$$

If your applications routinely retrieve large result sets, contact your network administrator to determine the optimum communication packet size. Then, set the FETCH BUFFER SIZE to a size that accommodates large result sets.

Specifications

Maximum value: 1 - 524288

Minimum value: 1

Default: 32000

Use: Windows and UNIX CLI client.

Example

```
FETCH BUFFER SIZE = 64000
```

MESSAGE POOL SIZE

MESSAGE POOL SIZE is an optional parameter for ODBC and CLI clients that specifies the size of the memory region in bytes for all memory allocation.

Description

Set the actual workable maximum value to 2 MB less than the region size. If the specified value is less than 1 MB, 1 MB is used. If the amount of storage that can be obtained is less than the specified value, the maximum amount available is obtained.

Recommendation: Set the value of MESSAGE POOL SIZE to at least 4 MB for 32-bit clients and to at least 8 MB for 64-bit clients.

Specification

Maximum value: 1 GB or more, depending on operating system.

Minimum value: 1048576 (1 MB)

Default value: 1048575 (1 MB)

Use: Data server, ODBC and UNIX CLI clients

Example

```
MESSAGE POOL SIZE = 16777216
```

NL CAT

NL CAT is a required parameter that points to a language catalog that contains messages in a specified language and encoding.

Description

You typically define NL CAT on the data server and the native z/OS client by using a data definition (DD) statement in startup procedures. You can also define NL CAT on the client by specifying one of the following:

- A path
- A data set name

On a USS client, you typically specify a data set name.

The following table describes the location of the NL CAT statement in supported environments.

Table 67. Location of NL CAT statement by environment

Environment	Location of NL CAT statement
Linux and UNIX	/opt/IBM/ISClassic113/CLI/lib/cac.ini
Windows	The Windows registry. Do not specify NL CAT on Windows.
z/OS	CACINIZ in SCACCONF
USS	CACINIU in SCACCONF

Localization functions native to the operating system supply a localization code that identifies the language locale to the database driver. The driver accesses the language catalog based on the locale. If the functions do not return a code that the

driver recognizes, the driver accesses the US English version of the language catalog. The following table describes supported language locales and localization codes.

Table 68. Language locales and localization codes

Language locale	Localization codes returned (Linux, UNIX, Windows)
English (United States)	us en_US en_US.UTF-8
Japanese	ja ja_JP ja_JP.UTF-8
Simplified Chinese	zh zh_CN zh_CN.UTF-8 zh_SG zh_SG.UTF-8
Traditional Chinese	zh zh_TW zh_TW.UTF-8 zh_MO zh_MO.UTF-8 zh_HK zh_HK.UTF-8

The database drivers access the message catalogs for your locale and encoding by referencing a fixed filename you cannot change. In some cases, you can have more than one version of the message catalog, each supporting a different encoding. If you are using CLI or ODBC drivers and have multiple choices of encoding for your locale, you must rename the message catalog for the encoding you are using to the fixed filename.

Example: The Japanese message catalog file is `cacmsg_ja_JP.cat`. The Japanese catalogs are distributed in SJIS and eucJP encodings. If you are running Shift-JIS, rename `cacmsg_ja_SJISJP.cat` to `cacmsg_ja_JP.cat`. If you are running eucJP, rename `cacmsg_ja_eucJP.cat` to `cacmsg_ja_JP.cat`. In a UNIX environment, you can create a link to the appropriate file that uses the fixed filename.

The following table describes supported encodings and related file names.

Table 69. Encodings and language catalogs

Language	Encoding	Encoded file name	Fixed file name
English	Latin	Not applicable	<code>cacmsg_us_EN.cat</code>

Table 69. Encodings and language catalogs (continued)

Language	Encoding	Encoded file name	Fixed file name
	UTF-8 ¹	cacmsg_UTF8.cat	cacmsg_us_EN_UTF8.cat
Japanese	Shift-JIS	cacmsg_ja_JP.cat	cacmsg_ja_JP.cat
	EUCJP	cacmsg_ja_eucJP.cat	cacmsg_ja_JP.cat
	UTF-8 ¹	Driver loads fixed filename directly	cacmsg_ja_JP_UTF8.cat
Simplified Chinese	EUC-CN	Not applicable	cacmsg_zh_CN.cat
	UTF-8 ¹	Driver loads fixed filename directly	cacmsg_zh_CN_UTF8.cat
Traditional Chinese	Big-5	cacmsg_zh_BIG5TW.cat	cacmsg_zh_TW.cat
	EUC-TW	cacmsg_zh_eucTW.cat	cacmsg_zh_TW.cat
	UTF-8 ¹	Driver loads fixed filename directly	cacmsg_zh_TW_UTF8.cat

¹ UTF-8 is supported on only Linux and UNIX.

Exception: If you are using JDBC drivers, you do not need to rename message catalog files. The JAR file contains all messages.

Specification

Valid values:

/Path

Where *Path* is the path of the language catalog file (Linux, UNIX, Windows).

DD:DDName

Where *DDName* is the name of a data definition, typically MSGCAT (data server, native z/OS client).

//dsn:DatasetName

Where *DatasetName* is the name of the message catalog data set SCACMENU (data server, USS client).

Examples

Linux

NL CAT = /opt/ibm/isclassic113/cli/lib

UNIX

NL CAT = /opt/IBM/isclassic113/cli/lib

Windows

NL CAT = C:\\Program Files\\IBM\\ISClassic113\\ODBC\\lib

Data server, native z/OS client

NL CAT = DD:MSGCAT

USS client

NL CAT = //dsn:CAC.V11R3M00.SCACMSGS

RESPONSE TIME OUT

RESPONSE TIME OUT is an optional parameter that specifies the maximum amount of time that the ODBC client waits for an expected response before the client terminates a connection.

Specification

Maximum value: 1000MS, 60S, and 60M respectively

Minimum value: 0MS

Default: 6M

Use: ODBC client

Formats

Valid formats:

*n*MS Number of milliseconds

*n*S Number of seconds

*n*M Number of minutes

Example

```
RESPONSE TIME OUT = 10M
```

SERVER CODEPAGE

SERVER CODEPAGE is an optional parameter that specifies the server code page. ICU4C uses this parameter to translate between code pages for the client and server.

Description

This parameter corresponds to the code page converter names and aliases for the CCSID that is used on the client and on the server. ICU4C provides conversion between code pages.

Specification

Use: UNIX CLI client

Example

```
SERVER CODEPAGE = IBM-933
```

SHAPING

SHAPING is an optional parameter for bidirectional language transformation that indicates if text shaping is required when text is rendered.

Description

The SHAPING option is for the server code page IBM-420, which encodes characters in their shaped forms. When Arabic text contains these kinds of characters in shaped forms, the shaping API is called to replace them with abstract characters when the server converts to the client code page or to replace them with shaped forms when converting from the client to the server.

If the text on the server contains Arabic letters in abstract form, set the SHAPING option to OFF to improve performance.

Specification

Valid values: ON and OFF

Default: ON

Use: ODBC and CLI clients

SYMMETRIC SWAPPING

SYMMETRIC SWAPPING is an optional parameter for bidirectional language transformation that ensures text is preserved in a logical order.

Description

Some characters, such as the greater-than sign or a left parenthesis, have an implied directional meaning. Within a text segment that is presented from right to left, these characters must be replaced to ensure that the correct meaning is preserved. This replacement is called *symmetrical swapping*. When SYMMETRIC SWAPPING is set to ON, these characters are replaced by their mirror image. Symmetric swapping is not performed for text that is in logical order.

Specifications

Valid values: ON and OFF

Default: ON

Use: ODBC and CLI clients

TASK PARAMETERS

TASK PARAMETERS is an optional parameter that specifies IBM C runtime options that are passed to system child tasks through the z/OS ATTACH macro.

Description

One common use of this parameter is to pass TCP/IP information to the Communications Interface task.

Specifications

Default: None

Valid values:

You can specify as values any valid variables that are preceded by the equal (=) sign.

TCPIP_PREFIX

This variable sets the high-level qualifier (hlq) for finding the TCP/IP system data sets. It can be set to use the installation-defined data sets or a user-defined data set.

The default value is TCPIP.

TCPIP_MACH

This variable sets the address space name/subsystem name of the TCPIP stack for Interlink. For IBM's TCP/IP system utilizing the Berkeley Socket interface, this parameter can also be specified in the hlq.TCPIP.DATA file under the parameter TCPIPUSERID.

The default value is TCPIP.

TZ The Time Zone environment variable must be set for each job on z/OS. The variable sets the time zone in which the task will start, for example Pacific Standard Time (PST).

For information about other valid TZ settings, see the IBM C compiler documentation.

Examples

```
TASK PARAMETERS= =TCPIP_PREFIX=TCPIP =TCPIP_MACH=TCPIP
TASK PARAMETERS = =MI =TZ=PST8PDT
```

This example sets the time zone to PST plus 8 hours from Greenwich mean time (8) and Pacific daylight time (PDT).

Using the same example for Eastern standard time (EST), enter the following information:

```
TASK PARAMETERS = =MI =TZ=EST5EDT
```

TEXT ORIENTATION

TEXT ORIENTATION is an optional parameter for bidirectional language transformation that specifies where bidirectional text begins.

Description

When data moves from the client to the server, text in SQL statements is always marked as Left-To-Right (LTR). To ensure that text is processed correctly, encode text into a host variable by binding parameters with the SQLBindParameter API. The following example shows a statement with a bound parameter:

```
SELECT * FROM BIDITAB WHERE NAME = ?;
```

Specifications

Valid values:

LTR

Marks the text as left-to-right.

RTL

Marks the text as right-to-left.

DLTR

The direction of the paragraph is set to the direction of the first strong character found. If no strong character exists, the direction is set to LTR.

DRTL

The direction of the paragraph is set to the direction of the first strong character found. If no strong character exists, the direction is set to RTL.

Default: LTR

Use: ODBC and CLI clients

TEXT PRESENTATION

TEXT PRESENTATION is an optional parameter for bidirectional language transformation that determines the text type on the server.

Specification

Valid values:

LOGICAL

No reordering occurs when this option is set. The values of TEXT ORIENTATION and SYMMETRIC SWAPPING are ignored. LOGICAL is the default setting for bidirectional code pages.

VISUAL

Text that is stored in visual order is the same as text that displays on a screen. Visual text is transformed in logical order by setting the TEXT PRESENTATION parameter to the VISUAL attribute. The output depends on the TEXT ORIENTATION parameter that must be set to the DLTR value.

VISUALLTR

Transforms text in visual typing order from left to right. The output depends on the TEXT ORIENTATION parameter that must be set to the DRTL or the DLTR value. DRTL is the default. In addition, the SYMMETRIC SWAPPING parameter must be set to ON.

Default: LOGICAL

Use: ODBC and CLI clients

TRACE LEVEL

The TRACE LEVEL is an optional parameter that regulates the amount of information that data server tasks record in the trace log.

Specification

Valid values: 0 - 20:

- 20 No trace information generated
- 16 Identify fatal error conditions
- 8 Identify all recoverable error conditions
- 4 Generate warning messages
- 3 Generate debugging information
- 1 Generate function call information
- 0 Trace all

Default: 4

Use: ODBC client

Important: Change this parameter only at the request of IBM Software Support. Settings lower than 4 cause response time degradation.

USERID

USERID is an optional parameter that is the default SQL ID. The default SQL ID is needed if no ID is present on a CONNECT statement or if a dynamic CONNECT statement is issued because the client application does not issue a CONNECT statement.

Description

The USERID value is used when the first line in the SQL input file is blank.

Specification

Maximum value: 7 characters with no spaces. If more than 7 characters are specified, only the first 7 are used.

Default: None

Use: UNIX CLI client configuration

Example

```
USERID = CACUSER
```

USERPASSWORD

USERPASSWORD is an optional parameter that is the default SQL ID password. The default SQL ID password is needed if no password is present on a CONNECT statement or if a dynamic CONNECT statement is issued because the client application did not issue a CONNECT statement.

Description

The USERPASSWORD value is used when the first line in the SQL input file is blank.

Specification

Maximum value: 8 characters with no spaces

Default: None

Use: UNIX CLI client configuration

Example

```
USERPASSWORD = CACPWD
```

Command reference

Look up syntax and explanations for commands that help you to manage subscriptions, Classic data servers, and configurations.

Classic data server administration commands

Use these commands to start and stop Classic data servers and services, list connected users, cancel user sessions, and view or print log messages.

Starting a data server

When you start a Classic data server, you start all of the services defined in the configuration file for the Classic data server.

About this task

You can perform either of the steps described in the following procedure to start a Classic data server. All services start if the value of the INITIALTHREADS configuration parameter is greater than 0.

Procedure

- Issue a console command to start the JCL procedure for the Classic data server:
S procname
where *procname* is the 1-8 character PROCLIB member name to be started. When you issue commands from the SDSF product, prefix all operator commands with the forward slash (/) character.
- Submit a batch job.

STOP command

Stopping a Classic data server stops all of the services that are running within it.

About this task

The purpose of the STOP,ALL command is to shutdown the data server. The data server stops after the services running in the data server complete their required processing.

If the shutdown process does not complete after issuing a STOP,ALL command, you can issue the STOP,ALL,IMMEDIATE command. For example, the shutdown process might not complete if a service encounters a problem and cannot complete its quiesce processing. In this case, you can issue the STOP,ALL,IMMEDIATE command to bypass service quiesce processing and stop the data server.

Procedure

- To stop a Classic data server, issue the following command in an MTO interface:
F name,STOP,ALL
name The name of the started task or batch job for the Classic data server.
- To stop a Classic data server immediately, issue the following command in an MTO interface:
F name,STOP,ALL,IMMEDIATE
name The name of the started task or batch job for the Classic data server.

START,SERVICE command

You can start an instance of a service that is defined in the configuration for the Classic data server.

About this task

You can use this command when you want to start a service without stopping and restarting the Classic data server. The service instance starts if the number of instances already active is less than the value of the MAXTHREADS configuration parameter.

Procedure

Issue the following command in an MTO interface, where *name_of_job* is the name of the started task for the Classic data server:

```
F name_of_job,START,SERVICE=name_of_service
```

STOP,SERVICE command

You can stop an instance of a non-critical service that is defined in the configuration for the Classic data server.

About this task

Important: You should not issue the STOP,SERVICE command regularly. When a data server is configured, you typically do not need to start and stop the services that run in the data server individually.

The STOP command cancels any user activity in a service and disconnects all active users from that service.

Restriction: You cannot stop a critical service. The logger service is critical to the operations of a Classic data server. If you attempt to stop a critical service by issuing a STOP,SERVICE command or a STOP,TASKID command, a warning message is issued.

Procedure

- To stop a service by means of its task ID, issue this command:

```
F server_name,STOP,TASKID=task_ID
```

server_name

The name of the task or batch job started by the Classic data server. This name is CACDS for Classic federation.

- To stop a service by means of its name, issue this command:

```
F server_name,STOP,SERVICE=name_of_service
```

server_name

The name of the task or batch job started by the Classic data server. This name is CACDS for Classic federation.

DISPLAY,ALL command

The **DISPLAY,ALL** command outputs a formatted list of the current usage information about a data server.

Procedure

To display current usage information about services, users, configurations, and the memory pool, issue the DISPLAY,ALL command:

```
F name,DISPLAY,ALL
```

name The name of the task or batch job started by the data server.

DISPLAY,MEMORY command

The **DISPLAY,MEMORY** command outputs a formatted list of usage information about data server memory.

Procedure

To display the current use of the memory pool in the data server, issue the `DISPLAY,MEMORY` command:

`F name,DISPLAY,MEMORY`

name The name of the task or batch job started by the data server.

The following information is displayed about overall data server memory usage:

TOTAL MEMORY

The total size in kilobytes of the message pool that was allocated.

USED The amount of memory that is currently being used out of the message pool. This value is expressed in kilobytes followed by the percentage of the current message pool that is being used.

MAX USED

The maximum amount of the message pool that was ever used. This value is expressed in kilobytes followed by the percentage of the message pool that was ever used.

DISPLAY,SERVICES command

The `DISPLAY,SERVICES` command outputs a formatted list of information about the services running in a data server.

Procedure

To display a list of all running services in the data server, issue this command:

`F name,DISPLAY,SERVICES`

name The name of the task or batch job started by the data server.

When information is requested about the services that are active within a data server, a WTO display message is generated for each service that is active. For each service, the following information is displayed:

SERVICE

Service name.

TASKID

TCB address (in decimal notation) of the service instance that is displayed.

TASKNAME

Same as TYPE.

STATUS

One of the values that is displayed in the table below.

USER The user ID that is currently being serviced. Generally, this value is blank.

The following table lists the most common statuses:

Table 70. States and descriptions

Status	Description
QUIESCE	Unused.
READY	Idle and waiting for requests.
RECEIVING	Receiving a request.
RESPONDING	Sending a response.

Table 70. States and descriptions (continued)

Status	Description
STOP	Processing a STOP,ALL request.

DISPLAY,USERS command

You can list all of the users that are connected to a Classic data server.

Procedure

To list users, issue this command in an MTO interface:

```
F name,DISPLAY,USERS
```

name The name of the started task or batch job for the Classic data server.

Results

When information is requested about active users, a WTO display message is generated for each user that is connected to the Classic data server. For each user, the following information is displayed:

USER The user ID that was supplied when the client application connected to the server.

TASKID

The TCB address of the query processor service that the client connected to.

SERVICE

The first 8 characters of the data source name that the client application connected to. This value corresponds to one of the names of a query processor service definition.

SESSIONID

The address (in decimal format) of a control block that is used to track the user.

HOSTNAME

The name of the host computer where the connection originated.

PROCESSID

The process ID on the host computer that the client application is executing in.

THREADID

The thread identifier on the host computer that the client application is executing in.

STMTS

This is a two-part value that is separated by a forward slash (/). The first value identifies the number of statements that the user currently has active. The second number identifies the maximum number of statements that were active.

MEMORY

This is a two-part value that is separated by a forward slash (/). The values are in kilobytes. The first value identifies the amount of memory that is currently being used to process the active statement. The second value identifies the maximum amount of memory that was used to process all statements that were issued by the user.

CANCEL command

You can cancel particular user sessions or all sessions for particular users.

About this task

You can find out which users are currently connected by issuing the `DISPLAY,USERS` command. The output of this command gives you the IDs of users and the IDs of the user sessions.

The cancel command takes effect when the query processor is not actively processing a query for the user that is being canceled. Users are notified upon completion of any action of theirs that follows the issuing of the command.

To disconnect users and user sessions, issue either of the following commands by using the MTO interface.

Procedure

- To cancel a particular user session, issue this command:
`F name,CANCEL,SESSIONID=session ID`
- To cancel all sessions for a particular user, issue this command:
`F name,CANCEL,USER=user ID`

DISPLAY,QUERIES command

You can list all of the queries that the query processors in a Classic data server are currently processing.

About this task

With the information provided in the output, you can issue the `CANCEL,QUERY` command to cancel a particular query.

Procedure

To list queries, issue this command in an MTO interface:

```
F name,DISPLAY,QUERIES
```

name The name of the started task or batch job for the Classic data server.

Results

When information is requested about the queries that are being tracked by the Classic data server, a WTO display message is generated for each query. For each query, the following information is displayed:

QUERY

The SQL statement name that is assigned by the client application or JDBC/ODBC Driver to track the query.

USER The user ID that issued the query.

SESSIONID

The user control block address (in decimal format) of the user that issued the query.

SERVICE

The first 8-characters of the service name that is processing the query.

TASKID

The TCB address (in decimal format) of the query processor that is handling the query.

TYPE One of the following values that indicates the type of query that is being processed:

XQRY

SELECT statement

XJOI SELECT statement that contains a join condition

XUNI SELECT statement that contains a union

XINS INSERT statement

XUPD

UPDATE statement

XDEL DELETE statement

CALL CALL statement

STATE

One of the following values that indicates the state that the query is in:

INITIAL

Statement is created and not yet prepared.

PREPARED

Statement is prepared.

OPENED

Statement is opened.

EXECUTING

Statement is being executed.

FETCHED

Result sets are being fetched.

WAITING

Statement is waiting to be executed.

SUSPENDED

Statement is suspended.

CLOSED

Statement is closed.

MEMORY

This is a two-part value that is separated by a forward slash (/). The values that are displayed are in kilobytes. The first value identifies how much memory the query is currently using. The second value identifies the maximum amount of memory that has been allocated to process the statement.

CANCEL,QUERY command

You can cancel queries that are running on a Classic data server or enterprise server.

Before you begin

Before issuing this command, issue the `DISPLAY,QUERIES` command to get the name and session ID for the query that you want to cancel.

About this task

The cancel command takes effect when the query processor is not actively processing the query to be canceled. Users are notified upon completion of any action of theirs that follows the issuing of the command.

To cancel a query, issue the following command using the MTO interface:

```
F name,CANCEL,QUERY=name,SESSIONID=session ID
```

Classic data server configuration commands

After a Classic data server is created and running as a result of the installation customization process, you can modify the configuration for the Classic data server as needed by using a set of operator commands.

You use configuration commands for the following actions:

- Updating and displaying configuration data for a Classic data server.
The commands include ADD, SET, DELETE, and DISPLAY.
- Importing and exporting configuration data for a Classic data server.
The commands include EXPORT and IMPORT.

You can issue the configuration-related commands by using the master terminal operator (MTO) interface. You can also use the Classic Data Architect to update your configuration for a Classic data server. With both the Classic Data Architect and the MTO interface, you make configuration updates against a running Classic data server.

The configuration migration and maintenance utility, CACCFGUT, also supports the EXPORT and IMPORT commands and provides a REPORT command. You can use the CACCFGUT utility to issue the commands offline, when the Classic data server is not running. For example, the utility supports the EXPORT and IMPORT commands that enable you to restore a configuration environment to a previous point in time.

Commands for updating and displaying configurations for Classic data servers

You use the ADD, SET, DELETE, and DISPLAY configuration commands to update configuration information for services, service lists, global configuration parameters, and user-specific configurations.

When you add a service with the ADD command, you add the service to the configuration for the Classic data server. Adding the service does not automatically start the service. A service starts automatically during the next startup of the Classic data server if the value of the INITIALTHREADS configuration parameter is set to a value of 1 or greater. Otherwise, if you do not restart the Classic data server, you must issue a START,SERVICE command to start the service.

After a service is added, you cannot update the name of the service. To change a service name, you must delete the existing service and then add a new service with the new service name.

Basic validation occurs when you modify configuration parameters with the SET command. These validations are limited to general parameter data type and numeric range checks. The individual services validate specific configuration parameters to verify parameter content and relationships during service startup.

In addition to services configurations, you can add and modify service lists. A service list is a type of parameter that represents list values.

You use the ADD, SET, and DELETE commands to maintain these parameters.

To update a service list entry, you must delete the existing entry and then add a new entry.

ADD, DELETE, and DISPLAY service list commands:

You can use the ADD and DELETE commands to add and remove service list entries. You can use the DISPLAY command to display service lists.

Description

A service list is a type of parameter that represents list values. A single service list can contain an unlimited number of entries.

A service list is always associated with a specific service.

You use the ADD and DELETE commands to maintain these parameters. To update a service list entry, you must delete the existing entry, and then add a new entry. You cannot use the SET command to update a service list entry.

Adding service list entries

You can add a service list entry with the following ADD command:

```
ADD,CONFIG,SERVICELIST=listname,SERVICE=servicename,VALUE=value
```

Deleting service list entries

You can delete entries from a service list with the following DELETE command:

```
DELETE,CONFIG,SERVICELIST=listname,SERVICE=servicename,VALUE=value
```

Displaying service list entries

Service list information is displayed when you display configuration information for the related service.

Parameters

listname

The service list name.

servicename

The service name associated with the service list name.

value

List value.

ADD configuration command:

You can use the ADD command to add new service definitions, user-specific definitions, and service lists to a configuration.

Adding services

Adding a new service typically occurs during the installation customization process. Otherwise, you can add a service by using the ADD configuration command.

When you add a service, you specify the service name and select the service class for the new service. The service class is predefined.

The following table lists the service classes and the service type that each service class defines.

Table 71. Service classes.

Service class	Service type
CAF	IBM DB2 call attachment facility (CAF) initialization service
CNTL	Region controller service
DCI	CA-Datacom access service
DRA	IBM IMS database resource adapter (DRA) access service
GLOB	Global service
IMSI	IMS BMP/DBB access service
INIT	Client connection handler service
LOG	Logger service
MAA	Monitoring service
ODBA	IMS open database access (ODBA) access service
OPER	Remote operator command service
QP	Single-phase commit query processor service
QPRR	Two-phase commit query processor service
SPS	Stored procedure service
VSMS	VSAM resource management service
WLM	Workload Manager (WLM) service

You can add services with the following ADD command:

```
ADD,CONFIG,SERVICE=servicename,SERVICECLASS=serviceclass
```

Example

To add a new database resource adapter service named DRA2 to the configuration, issue the following command:

```
ADD,CONFIG,SERVICE=DRA2,SERVICECLASS=CACDRA
```

In this example, the configuration parameters for the DRA2 service are created with the default values for the CACDRA service class.

Adding user-specific configurations

When you add a user-specific configuration, you must specify a user ID. The user ID is associated with a defined service. You can override the default values for the configuration parameters for the service that you specify. In version 9.5, you can define user-specific configurations for query processor services only.

Command format:

```
ADD,CONFIG,USER=userid,SERVICE=servicename
```

Example

To add a new user record for TESTUSER to the configuration for service TESTV10, issue the following command:

```
ADD,CONFIG,CONFIG,USER=TESTUSER,SERVICE=TESTV10
```

Parameters

serviceclass

The name of the service class that the service is associated with.

servicename

When adding a service, specify the name of the service to create. When adding a user-specific configuration, specify the name of the service to override. The maximum length allowed for the name is 64 bytes.

userid

For a user-specific configuration, the user logon ID used to connect to the Classic data server. The maximum length allowed for the user ID is 8 bytes.

DELETE configuration command:

You can use the DELETE command to remove service definitions, service list entries, and user-specific definitions from a configuration.

Deleting services

To delete a service, you must specify the name of the service to delete. If a service is running, you must stop the service before you delete the service. When a service configuration is deleted, any associated user-specific configuration is also deleted.

You cannot delete the core default services — the controller service and the logger service. You also cannot delete the GLOBAL service name which represents global parameters.

You can remove a non-core service with the following DELETE command:

```
DELETE,CONFIG,SERVICE=servicename
```

Example

To delete the service TESTV10 from the configuration, issue the following command:

```
DELETE,CONFIG,SERVICE=TESTV10
```

Deleting user-specific configurations

To delete a user-specific configuration, you must stop the associated service. You specify the user logon ID that is used to connect to the Classic data server.

DELETE command format:

```
DELETE,CONFIG,USER=userid,SERVICE=servicename
```

Example

To delete the user configuration for user ID TESTUSER related to service CACLOG from the configuration, issue the following command:

```
DELETE,CONFIG,USER=TESTUSER,SERVICE=CACLOG
```

Parameters

servicename

When deleting a service, specify the name of the service to delete. When deleting a user-specific configuration, specify the name of the related service.

userid

The user ID of the user to delete.

DISPLAY configuration command:

You can use the DISPLAY command to display configuration information about services, service lists, user-specific configurations, and global configuration parameters.

Displaying all configuration information

You can display all configuration information for a Classic data server with the following DISPLAY command:

```
DISPLAY,CONFIG,ALL
```

During startup of the Classic data server, the contents of the configuration file are written to the first entry in the diagnostic log for the Classic data server.

Displaying service information

You can display configuration information about a specific service with the following DISPLAY command:

```
DISPLAY,CONFIG,SERVICE=servicename
```

Example

To display all configuration parameters in the service named TESTV10, issue the following command:

```
DISPLAY,CONFIG,SERVICE=TESTV10
```

Displaying user-specific configuration information

You can display user-specific configuration information about a specific user configuration, a specific user, or all users with the following DISPLAY commands.

- Display a user-specific configuration for the specified service:

```
DISPLAY,CONFIG,USER=userid,SERVICE=servicename
```

Example

To display all parameters in user configuration for USER1 and service CACSAMP, issue the following command:

```
DISPLAY,CONFIG,USER=USER1,SERVICE=CACSAMP
```

- Display all user-specific configurations across all services that contain the specified user ID:

```
DISPLAY,CONFIG,USER=userid,SERVICE=ALL
```

Example

To display all parameters in all user configurations for USER1, issue the following command:

```
DISPLAY,CONFIG,USER=USER1,SERVICE=ALL
```

- Display all user-specific configurations across all services:

```
DISPLAY,CONFIG,USER=ALL
```

Parameters

servicename

The name of the service to display. To display global parameters, specify GLOBAL as the service name.

userid

The user ID of the user-specific configuration.

SET configuration command:

You can use the SET command to modify parameter values defined for an existing service, a user-specific configuration, and global parameters.

Modifying service information

You can use the following SET commands to modify parameters in a single service, modify all services within a given service class, and modify all services across all service classes.

You can also use the SET command to reset the values of a configuration parameter value to the parameter default value.

- For a specific service, use the following SET command:

```
SET,CONFIG,SERVICE=servicename,parm=value
```

Example

To set the TRACELEVEL parameter in service TESTV10 to 8, issue the following command:

```
SET,CONFIG,SERVICE=TESTV10,TRACELEVEL=8
```

- To reset a parameter value to the default for specific service, use the following SET command:

```
SET,CONFIG,SERVICE=servicename,parm=DEFAULT
```

Example

To set the TRACELEVEL parameter in service TESTV10 to the default value (4), issue the following command:

```
SET,CONFIG,SERVICE=TESTV10,TRACELEVEL=DEFAULT
```

- For services within a given service class, use the following SET command:

```
SET,CONFIG,SERVICECLASS=serviceclass,parm=value
```

Example

To set the TRACELEVEL parameter in all monitoring services to 3, issue the following command:

```
SET,CONFIG,SERVICECLASS=MAA,TRACELEVEL=3
```

- For all services, use the following SET command:

```
SET,CONFIG,SERVICE=ALL,parm=value
```

Example

To set the value of the TRACELEVEL parameter in all services to 2, issue the following command:

```
SET,CONFIG,SERVICE=ALL,TRACELEVEL=2
```

Modifying user-specific configuration information

You can set the value of a specific parameter in a user-specific configuration with the following SET command.

```
SET,CONFIG,USER=userid,SERVICE=servicename,parm=value
```

Example: Set the value of the TRACELEVEL parameter in the user-specific configuration for USER1 for TEST95 to 2.

```
SET,CONFIG,USER=USER1,SERVICE=TEST95,TRACELEVEL=2
```

You can also use the DEFAULT keyword on the SET command to return a parameter to its default value.

Parameters

parm

The name of the configuration parameter to modify.

serviceclass

The service class of the service to modify.

servicename

The name of the service that contains the configuration parameter to modify. For global parameters, specify GLOBAL as the service name. The maximum length allowed for the name is 64 bytes.

userid

The name of the user logon ID. The maximum length allowed for the user ID is 8 bytes.

value

The new parameter value.

Usage notes

The following rules apply to specifying lowercase and uppercase character values for the SET command:

- Character strings that contain embedded spaces or special characters must be enclosed in either single or double quotation marks.
 - For double quotation marks (“), the character string is set to uppercase.
 - For single quotation marks ('), values specified in lowercase are saved as lowercase.

Commands for importing and exporting configurations for a Classic data server

An export does not affect an existing configuration for a Classic data server. The EXPORT command creates a snapshot of the configuration in a command file that contains ADD and SET commands. The command file can be used as input to the IMPORT command.

You can use the IMPORT and EXPORT commands to perform the following functions:

- Back up and restore a current configuration environment. The EXPORT command creates a command file that consists of ADD and SET commands

based on the current environment that can later be imported into a Classic data server with newly initialized configuration files to complete the restore process.

Important: Frequently back up configuration files by copying them or by using the EXPORT command.

- Apply updates to the current configuration environment. You can use the IMPORT command to apply updates to the current configuration environment in multiple updates or single updates. You can use the EXPORT command to create a command file or create a command file manually.
- Save different versions of a configuration environment. You can use exported configuration output to create a clone of the existing configuration for a Classic data server. For example, the EXPORT command is useful in a test environment where you can rebuild the configuration required for a specific test scenario by importing the saved configuration. The EXPORT and IMPORT commands can also be an effective mechanism for cloning Classic data servers.

Example

You can use the EXPORT and IMPORT process to restore a specific configuration environment to a previous point in time. By using the EXPORT command, you can create a command file that is based on the configuration environment of a running Classic data server. You can then use this command file to update a different configuration file by using the IMPORT command.

For example, you can use the EXPORT command to generate a command file, and then IMPORT that command file on a server that is running with a newly initialized configuration file.

To build a new configuration environment that is identical to an existing configuration environment, export the source configuration to the desired target file. Then run the configuration migration and maintenance utility CACCFGUT to import the original source configuration to the new target environment.

You can also issue the IMPORT command while a Classic data server is running to update parameter settings in the current configuration environment.

EXPORT configuration command:

The EXPORT command is useful for multiple purposes, such as backing up configuration information and cloning a configuration for a Classic data server.

Description

You can use the EXPORT and IMPORT process to restore a specific configuration environment to a previous point in time. By using the EXPORT command, you can create a command file that is based on the configuration environment of a running Classic data server. You can then use this command file to update a different configuration file by using the IMPORT command.

The target of the EXPORT command is a PDS member or a sequential file. If the file or member that you specify in the EXPORT command already exists, it is rewritten. If the file or member does not exist, it is created.

If you predefine the EXPORT target:

- The minimum record length is 80 bytes.
- The format can be either fixed or variable length records.

The EXPORT command does not support GDGs.

The owner ID associated with the Classic data server job must have authorization with an external security manager (ESM), such as the Resource Access Control Facility (RACF), to create or access the EXPORT target data set (sequential file or PDS member).

When a PDS member is specified as the target of the EXPORT command, an attempt to run the EXPORT command fails if another user or job accesses the PDS member at the same time. To avoid this situation, you create a PDS member to use for the IMPORT and EXPORT process only.

Exporting configuration information

EXPORT command format:

```
EXPORT,CONFIG,FILENAME=DSN:dsname | DSN:dsname(member) | DDN:ddname  
| DDN:ddname(member)
```

Example

The following example shows sample file contents of an EXPORT file:

```
--SET,CONFIG,SERVICE=GLOBAL,MESSAGEPOOLSIZ=67108864;  
--SET,CONFIG,SERVICE=GLOBAL,DATACONVERRACT=1;  
--SET,CONFIG,SERVICE=GLOBAL,FETCHBUFSIZE=32000;  
--SET,CONFIG,SERVICE=GLOBAL,DECODEBUFSIZE=8192;  
--SET,CONFIG,SERVICE=GLOBAL,STATICCATALOGS=0;  
--SET,CONFIG,SERVICE=GLOBAL,TASKPARM='';  
--SET,CONFIG,SERVICE=GLOBAL,DATAVALIDATEACT=0;  
--SET,CONFIG,SERVICE=GLOBAL,REPORTLOGCOUNT=0;  
ADD,CONFIG,SERVICE=CNTL,SERVICECLASS=CNTL;  
--SET,CONFIG,SERVICE=CNTL,INITIALTHREADS=1;  
--SET,CONFIG,SERVICE=CNTL,MAXTHREADS=1;  
--SET,CONFIG,SERVICE=CNTL,MAXUSERS=100;  
--SET,CONFIG,SERVICE=CNTL,TRACELEVEL=4;  
--SET,CONFIG,SERVICE=CNTL,RESPONSETIMEOUT=5M;  
--SET,CONFIG,SERVICE=CNTL,IDLETIMEOUT=5M;  
SET,CONFIG,SERVICE=CNTL,SEQUENCE=1;
```

In the example, the export operation produces a command file of all overridden parameters as active SET commands. SET commands are also generated for all parameters that use default values. These particular SET commands are generated as comments. The prefix "--" in the first two columns of the command identifies comments.

Parameters

ddname

The DD statement defined in the JCL that starts the Classic data server. The DD name points to the target of the EXPORT command.

dsname

The name of the EXPORT data set.

Usage notes

The following rules apply to the format of command files:

- Commands must end with a semicolon.
- A comment begins with two dashes ("--") in columns 1 and 2.

- A single command can span multiple lines. You can break a line after a comma or a space. You cannot break a keyword or value across multiple lines. For example:

```
SET,CONFIG,SERVICE=XM1/CRAR/QRAR/2048/8,TEMPFILESPEACE=
'HIPERSPACE,INIT=2048M,MAX=2048M,EXTEND=0M';
```

Example

This EXPORT command creates the file USER.TEST.FILE if the file does not already exist.

```
EXPORT,CONFIG,FILENAME=DSN:USER.TEST.FILE
```

IMPORT configuration command:

You can use the IMPORT command to apply multiple updates or single updates to the configuration file for a running Classic data server. You can also use the IMPORT command to perform recovery operations to restore configuration information.

Description

The updates that the IMPORT command processes must reside in an existing IBM z/OS PDS member or sequential file. You can use either of the following methods to build the input file:

- Manually create the file and populate it with a defined set of commands.
 - When you manually create a command file, you can specify any valid format of the ADD, SET, or DELETE configuration commands.
- Issue the EXPORT command to generate an IMPORT command file.

All commands in the file are processed whether or not a single command fails. Any errors encountered during the IMPORT process are displayed on the operator console. For example, unknown commands or incorrect command syntax can cause errors. Attempting to add a service that already exists can also cause an error. A status message is displayed when the process is complete.

You can use the IMPORT command at any time to change configuration parameter settings for existing services on a running Classic data server.

Importing configuration command files

IMPORT command format:

```
IMPORT,CONFIG,FILENAME=DSN:dsname | DSN:dsname(member) | DDN:ddname
| DDN:ddname(member)
```

The data that you import must be in the command file format as described for the EXPORT command.

Example:

The following sample shows the contents of a sample IMPORT command file:

```
ADD,CONFIG,SERVICE=TEST10,SERVICECLASS=QP;
SET,CONFIG,SERVICE=IMSLRS,TRACELEVEL=3;
```

When this command file is imported, these changes occur:

- A new service named TEST10 is added

- The value of TRACELEVEL is set to 3 for the QP service

Parameters

ddname

The DD statement defined in the JCL that starts the Classic data server. The DD name points to the source IMPORT command file.

dsname

The name of the IMPORT data set.

Example

This IMPORT command imports the file USER.PDS.FILE.

```
IMPORT,CONFIG,FILENAME=DSN:USER.PDS.FILE(USER1)
```

IMPORT,CONFIG,MIGRATE configuration command:

You can use the IMPORT,CONFIG,MIGRATE commands to migrate the master configuration member, query processor override members, and user configuration members from a previous release to a new version 11.3 configuration.

Description

Recommendation: Use the configuration migration and maintenance utility, CACCFGUT, to migrate an existing configuration when the Classic data server is not running. The utility issues the commands described below.

You must issue the commands to migrate the master configuration before you migrate the override members and user configurations.

Importing the master configuration and query processor override members

These formats of the IMPORT command convert the master configuration member, and all query processor override members that it references, to the new version 9.5 configuration.

```
IMPORT,CONFIG,MIGRATE,ALL,FILENAME=DSN:pds (member) | DDN:ddname (member) | member
IMPORT,CONFIG,MIGRATE,FILENAME=DSN:pds (member) | DDN:ddname (member) | member
```

All configuration members related to the existing configuration must reside in the same PDS.

Importing the master configuration

This format of the IMPORT command migrates the master configuration only. Query processor definitions that refer to query processor override members are migrated along with master configuration values.

```
IMPORT,CONFIG,MIGRATE,MASTERCONFIG,FILENAME=DSN:pds (member) | DDN:ddname (member)
| member
```

Importing a query processor override member

This format of the IMPORT command migrates a query processor override member. You can migrate the override member for a specific query processor or for all query processors that are defined in the current configuration. For example, this command migrates all query processors.

```
IMPORT,CONFIG,MIGRATE,OVERRIDEMEMBER= DSN:pds(member) | DDN:ddname(member) |  
member, SERVICE=servicename | ALL
```

Importing a user-specific configuration

This format of the IMPORT command migrates a user-specific configuration. For this command, query processors must already be added or migrated to a new version 9.5 configuration file. The user configuration can either be applied to a specific query processor task or to all query processor tasks in the new configuration.

```
IMPORT,CONFIG,MIGRATE,USER= DSN:pds(member) |  
DDN:ddname(member) | member, SERVICE=servicename | ALL
```

Parameters

ddname

The DD statement defined in the JCL that starts the Classic data server. The DD card points to the configuration file from the previous version.

member

The source PDS member name of the master configuration, query processor override member, or user override member.

pds

The name of the configuration PDS from the previous version.

servicename

The name of the query processor service for the related user override.

Usage note

You can specify FILENAME, OVERRIDEMEMBER, or USER values as a DSN, a DD statement, or a PDS member name. When you specify a PDS member name, the migration process defaults to a PDS referred to by the VHSCONF DD card from the previous version. All configuration members related to the existing configuration must reside in the same PDS.

Example

This example shows how to use the member name only when a VHSCONF DD statement is defined.

- IMPORT,MIGRATE command:
F CACDS, IMPORT, CONFIG, MIGRATE=CACDSCF

Where CACDS is the job name for the Classic data server and CACDSCF is the name of the configuration member that will be migrated

- VHSCONF DD statement in the job control language (JCL) for the Classic data server:

```
//VHSCONF DD DISP=SHR,DSN=CAC.V11R3M00.SCACCONF
```

- The data is migrated from here:
CAC.V11R3M00.SCACCONF(CACDSCF)

Utilities reference

To manage the metadata catalog on a Classic data server, use the catalog initialization and maintenance utility (CACCATUT) and the z/OS metadata utility. To work with configurations from a previous version, monitor data server configurations, or for backup and recovery purposes, use the configuration migration and maintenance utility.

The catalog initialization and maintenance utility (CACCATUT)

The catalog initialization and maintenance utility (CACCATUT) is a z/OS batch job that creates or performs operations on a metadata catalog when the data server is stopped.

The catalog-related JCL is customized during the installation customization process. You can configure the standalone CACCATUT JCL to perform one of the following operations when it runs:

- Create and initialize a version 11.3 zFS metadata catalog
- Create and initialize a version 11.3 sequential metadata catalog
- Create and initialize a version 11.3 linear metadata catalog
- Upgrade an existing pre-version 11.3 sequential metadata catalog to a version 11.3 zFS or sequential metadata catalog.
- Upgrade an existing pre-version 11.3 linear metadata catalog to a version 11.3 zFS or sequential metadata catalog.
- Report on space utilization, the contents of a zFS or sequential metadata catalog, and any corruptions that might exist within the metadata catalog.
- Reorganize the contents of a zFS or sequential metadata catalog to reclaim unused space and, where possible, correct any corruptions that might exist.
- Create a version 11.3 zFS or sequential copy of a version 11.3 zFS, sequential, or linear metadata catalog.
- Load system objects into the metadata catalog.

You can set up your metadata catalogs manually. See the guidelines for estimating the size of the metadata catalog for detailed information.

The catalog initialization and maintenance utility is distributed in library SCACLOAD as member name CACCATUT.

Estimating the size of the catalog initialization and maintenance utility

You can use the guidelines provided with the formula in this topic to estimate the size of the metadata catalog.

During the installation customization process you provide input to estimate the size of the metadata catalog. This estimate is used to allocate the file space that will store the metadata catalogs. The estimate is based on the formula below.

You can also use the formula if you set up your metadata catalogs manually. This is a generous estimate because an accurate size determination is not feasible and there is no guarantee to fully store a catalog. In the unlikely event that you run out of file space, you will receive an explicit error message providing further instructions.

ESTIMATED CATALOG OBJECT SIZE in Bytes = 314572800

+ <number of expected tables> * 3710

+ <number of expected tables> * <maximum or average number of expected columns per table> * 776
 + <number of expected views> * 3086
 + <number of expected views> * <maximum or average number of expected columns per view> * 776
 + <number of expected indexes> * 1278
 + <number of expected indexes> * <maximum or average number of expected columns per index> * 264
 + <number of expected stored procedures> * 1543
 + <number of expected stored procedures> * <maximum or average number of parameters per stored procedure> * 264
 TOTAL ESTIMATED CATALOG SIZE in Bytes = ESTIMATED CATALOG OBJECT SIZE
 + <user-defined reserved space in bytes>
 TOTAL ESTIMATED CATALOG SIZE in Megabytes = TOTAL ESTIMATED CATALOG SIZE in Bytes / 1048576

Notes:

- You supply the number in brackets based on your planned usage of the catalog.
- A conservative user would use the maximum number instead of the average number in the formula.
- Minimally, you should reserve an additional 1/3 of the calculated catalog object size.
- Indexes are typically used in Classic Federation for IMS, VSAM, DB2, and Adabas data sources only.
- Because stored procedures are rarely used, it is likely that you can ignore providing input for them.
- A reserve for GRANT authorities is included in the formula which should be sufficient, even for the largest catalog.

Creating and initializing zFS metadata catalogs

The zSeries File System (zFS) metadata catalog is initialized as part of the installation customization process.

Before you begin

The zFS format of the metadata catalog is the default. This format is the recommended configuration for the metadata catalog that provides significant performance and capacity capabilities in comparison to the sequential or linear data set formats used in releases prior to version 11.3. The sequential and linear formats continue to be supported for compatibility.

Using the zFS format of the metadata catalog requires the definition of a z/FS file system. The data set that contains the zFS file systems is called a zFS *aggregate*. This data set will contain the files and directories for the file system and is mounted into the z/OS UNIX hierarchy.

You must create and mount the file system before trying to create the metadata catalog. To mount the file system, the user issuing the MOUNT command must have the following Unix privileges:

- SAF READ-access level authorization is required for the SUPERUSER.FILESYS.PFSCCTL resource in the UNIXPRIV class to run the zFS administration command, IOEZADM.
- SAF READ-access level authorization is required for the SUPERUSER.FILESYS.MOUNT resource in the UNIXPRIV class to perform MOUNT and UNMOUNT operations against USS file systems

For more information about zFS files, see *Distributed File Service, SMB, and zFS* and *z/OS UNIX System Services* in the z/OS product documentation.

The installation customization provides steps to setup the files system and an example for mounting the file system in addition to the steps included below.

About this task

zFS metadata catalogs provide improved performance and capacity compared to sequential and linear metadata catalogs. You can update a zFS metadata catalog and eliminate the need to maintain both an updatable sequential catalog and a non-updatable linear catalog. With zFS support, catalogs can be greater than 4GB in size in contrast to the 2GB limitation for sequential and linear catalogs.

Recommendation: Use the sizing formula during the installation customization process to estimate the required catalog size. See the topic 'Estimating the size of the catalog initialization and maintenance utility.

Procedure

1. Define and format a zFS aggregate by using `USERHLQ.USERSAMP(CECCRZCT)`.
2. Mount the file system using the sample MOUNT command provided in `USERHLQ.USERSAMP(CECCRZCT)`. Ensure that the user issuing the MOUNT command has the necessary privileges.
3. Follow the installation customization process. During the customization process, the CACCATLG member in the SCACSAMP data set is customized. The zFS metadata catalog is initialized as part of the installation customization process.
4. Optional: To initialize additional catalogs as needed, customize and run JCL member CACCATLG. Member CACCATLG in the SCACSAMP data set contains JCL to run CACCATUT with the INIT option. The INIT operation of the catalog initialization and maintenance utility (CACCATUT) initializes USS file system files for a version 11.3 zFS metadata catalog and creates the SYSIBM and SYSCAC system tables for the metadata catalog.
 - a. Provide a job card that is valid for your site.
 - b. Change the value of the CAC parameter to your high-level qualifier.
 - c. Change the value of the DISKU parameter to a valid DASD unit type for your site.
 - d. Change the value of the DISKVOL parameter to identify the volume where you want to locate the system catalog.

Results

When the catalog initializes, it grants SYSADM privileges to the user ID that creates the catalog.

Creating and initializing sequential metadata catalogs

The sequential metadata catalog is initialized as part of the installation customization process.

Before you begin

Important: The zFS format of the metadata catalog is the default. This format is the recommended configuration for the metadata catalog that provides significant performance and capacity capabilities in comparison to the sequential data set format.

To grant ownership privileges for the catalog, ensure that each TSO user ID is assigned a unique OMVS UID. Set an automatically generated UID for the user IDs that you are using in RACF:

```
ALTUSER USER01 OMVS(NUUID)
ALTUSER USER01 OMVS(AUTOUID)
```

Procedure

- Follow the installation customization process. The CACCATLG member in the SCACSAMP data set is customized. The sequential metadata catalog is initialized as part of the installation customization process.
- Optional: To initialize additional catalogs as needed, customize and run JCL member CACCATLG. Member CACCATLG in the SCACSAMP data set contains JCL to run CACCATUT with the INIT option. The INIT operation of the catalog initialization and maintenance utility (CACCATUT) initializes data sets for a version 11.3 sequential metadata catalog and creates the SYSIBM and SYSCAC system tables that make up the metadata catalog.
 1. Provide a job card that is valid for your site.
 2. Change the value of the CAC parameter to your own high level qualifier.
 3. Change the value of the DISKU parameter to a valid DASD unit type for your site.
 4. Change the value of the DISKVOL parameter to identify the volume where you want the system catalog located.

Results

When the catalog initializes, it grants SYSADM privileges to the user ID that creates the catalog automatically.

Creating and initializing linear metadata catalogs

To use a linear metadata catalog, you first create a sequential metadata catalog, populate it with the tables and other objects that you create with Classic Data Architect or the metadata utility, and then copy the content to a linear metadata catalog.

Before you begin

A sequential metadata catalog must exist on the z/OS LPAR where the Classic data server is located. To create a sequential metadata catalog, see “Creating and initializing sequential metadata catalogs” on page 373.

The sequential metadata catalog must contain the final versions of all of the mapped tables and other objects that you plan to use. You cannot directly update the content of a linear metadata catalog. To modify a mapped table or any other object in a linear metadata catalog, you must update the object in a sequential metadata catalog and then copy the content of the sequential metadata catalog into your linear metadata catalog.

About this task

Prior to version 11.1, you could use linear metadata catalogs to improve performance. However, they are dependent on a sequential metadata catalog because linear metadata catalogs cannot be updated.

Important: The zFS format of the metadata catalog is the default. A zFS metadata catalog is updatable and its performance matches, if not exceeds, the performance

of a linear metadata catalog. zFS also accommodates a catalog size greater than 4GB, making it the most flexible metadata catalog option.

Procedure

1. Stop the Classic data server.
2. Customize and run member CACLCAT in the SCACSAMP data set.
 - a. Provide a job card that is valid for your site.
 - b. Change the data set names and volsers in the IDCAMS definition to match your requirements.
3. Copy the content of your sequential metadata catalog into the linear metadata catalog. See “Copying metadata catalogs” on page 376.
4. In the configuration file for the Classic data server, set the value of the configuration parameter STATICCATALOGS to 1.
5. Update the JCL for the Classic data server to point to the linear metadata catalog.
6. Start the Classic data server.

Upgrading metadata catalogs

You can upgrade a sequential or zFS metadata catalog to a new version of the metadata catalog.

About this task

You can upgrade metadata catalogs from:

- A previous version of a sequential metadata catalog to a new version of a sequential or zFS metadata catalog.
- A current version of a sequential metadata catalog to a new version of a zFS metadata catalog.
- A previous version of a zFS metadata catalog to a new version of a zFS metadata catalog.

The UPGRADE operation copies an existing metadata catalog to a new version of the metadata catalog. For an UPGRADE operation, the CACCAT and CACINDX DD statements refer to the new USS file system path of the zFS metadata catalog or to sequential metadata catalog data sets. These statements identify the target metadata catalog. The INCAT and ININDEX DD statements refer to the old version of the metadata catalog.

During the UPGRADE operation, the user objects in the source metadata catalog are copied to the target metadata catalog. All user tables, indexes, views, and stored procedure definitions are copied to the target metadata catalog. Also, all security authorizations that exist in the source metadata catalog, including all security authorizations that apply to the system tables, are copied to the target metadata catalog.

After the UPGRADE operation completes, the CACCATUT utility generates a summary analysis report that identifies the contents of the metadata catalog and current space utilization.

Note: When you upgrade a metadata catalog that contains table mappings for CA-IDMS that are flagged for data capture, CACCATUP verifies that these tables have an explicit database name defined. If such a table does not have a database

name defined, CACCATUP changes the value of the DATA CAPTURE flag to a space and issues warning message 0x00760022.

Procedure

1. Follow the installation customization and migration process. The CACCATUP member in the SCACSAMP data set is customized. The sequential metadata catalog is upgraded as part of the installation customization and migration process. See Customizing installation environments.
2. To update the catalog as needed after the installation customization and migration process, customize and run member CACCATUP.
 - a. Provide a job card that is valid for your site.
 - b. Change the CAC parameter to installed high level qualifier.
 - c. Change the OLDCAT parameter to identify the high level qualifier of the input metadata catalogs.
 - d. Change the DISKU parameter to a valid DASD unit type for your site.
 - e. Change the VOLSER parameter to identify the volume where you want the metadata catalog located.
3. Update the JCL for the Classic data server to point to the new metadata catalog.

Copying metadata catalogs

You can create a copy of a metadata catalog.

About this task

The COPY operation copies the contents of an existing metadata catalog into a new metadata catalog for the same release as the Classic Federation data server.

The INCAT and ININDEX DD statements are required and identify the metadata catalog that is to be copied into the target metadata catalog identified by the CACCAT and CACINDEX DD statements. The INCAT and ININDEX DD statements cannot refer to the same data sets names that CACCAT and CACINDEX DD statements refer to.

During the COPY operation, you can modify the size and organization of the target metadata catalog so that it differs from that of the input metadata catalog. For example, you can change the data set organization from sequential to linear or vice versa. You can also increase or decrease the size of the target metadata catalog.

After processing finishes, a summary report is generated. This report identifies the contents of the metadata catalog and current space utilization of the created metadata catalog.

Note: If the target metadata catalog is not large enough, CACCATUT issues informational message 0x00710401.

Procedure

1. Customize and run member CACCATUT of the SCACSAMP data set.
 - a. Provide a job card that is valid for your site.
 - b. Change the CAC parameter to installed high level qualifier.
 - c. Change the OLDCAT parameter to identify the high level qualifier of the input system catalogs.
 - d. For the PARM keyword of the EXEC statement, specify COPY.

2. Update the JCL for the Classic data server to point to the target metadata catalog.

Reorganizing metadata catalogs

You can reorganize a zFS or sequential metadata catalog to reclaim wasted disk space.

About this task

You can reorganize zFS and sequential metadata catalogs. You cannot reorganize linear metadata catalogs.

The REORG operation compresses the contents of an existing metadata catalog by moving the metadata catalog. The new version of the metadata catalog does not contain the fragmented space that occurs as tables and other objects are dropped from the metadata catalog.

In this operation, the INCAT and ININDEX DD statements reference the metadata catalog that is being reorganized and the CACCAT and CACINDEX DD statements refer to the new metadata catalog that the REORG operation will create. If these statements reference an existing metadata catalog, the REORG operation replaces the content of this catalog.

If corruptions are detected in the source metadata catalog, the REORG operation attempts to minimize the loss of data when transferring the corrupted definitions from the source metadata catalog into the target metadata catalog.

After processing finishes, a summary analysis report is generated. This report identifies the contents of and the current space utilization of the new metadata catalog.

Note: If the target metadata catalog is not large enough, CACCATUT issues informational message 0x00710401.

Procedure

1. Customize and run member CACCATUT of the SCACSAMP data set.
 - a. Provide a job card that is valid for your site.
 - b. Change the CAC parameter to installed high level qualifier.
 - c. Change the OLDCAT parameter to identify the high level qualifier of the input metadata catalogs.
 - d. Change the NEWCAT parameter to identify the high level qualifier of the output metadata catalogs.
 - e. For the PARM keyword of the EXEC statement, specify REORG.
2. Update the JCL for the Classic data server to point to the new metadata catalog.

Loading system objects into metadata catalogs

You can load system objects into the metadata catalog.

About this task

The METALOAD operation populates new system objects in the catalog such as stored procedures and table definitions.

The catalog that the CACCAT and CACINDX DD statements point to is updated with the most current object definitions.

Procedure

1. Run member CACCATMD. The customized CACCATMD member provides an example of how to run the METALOAD command. CACCATMD is available in the SCACSAMP data set.
2. For the PARM keyword of the EXEC statement, specify METALOAD.

Results

Message 0x00760205 is written to SYSPRINT for each system object that is successfully loaded.

Generating reports about metadata catalogs

Use member CACCATRP in the SCACSAMP data set to generate an analysis report that identifies the contents of a sequential metadata catalog that is referenced by the CACCAT and CACINDX DD statements. You can specify an additional command line parameter that identifies the level of detail to include in the report and the amount of validation to be performed on the contents of the metadata catalog.

About this task

These options are available for the reports that you generate:

SUMMARY

Generates a summary report that identifies general information about the metadata catalogs, space usage within the metadata catalog and summary information about the number and types of objects stored in the metadata catalog. This is the default type of report.

DETAIL

Generates a summary report and produces a detail report that identifies all objects stored in the metadata catalog and identifies the structural linkages between the different metadata catalog objects.

VALIDATE

Generates a detail report and validates the structural linkages for each metadata catalog object.

- For more information on summary reports, see “Summary reports.”
- For more information on detail reports, see “Object detail reports” on page 382.

Procedure

Run member CACCATRP. The customized CACCATRP member is available in the SCACSAMP data set.

For the PARM keyword of the EXEC statement, specify either SUMMARY, DETAIL, or VALIDATE.

Summary reports:

After the catalog initialization and maintenance utility (CACCATUT) performs UPGRADE, REORG, REPORT, and COPY operations, it generates system catalog analysis reports. All such reports include a summary report.

Summary information is printed on a single page. The following example shows the contents of the summary report that is written to SYSPRINT when an UPGRADE, REORG or COPY operation is performed, or when any kind of REPORT is requested.

```
Date: yyyy/mm/dd      metadata catalog Analysis Report      Page 1
Time: hh:mm:ss              Summary
```

Index Component Information

```
Dataset name:          Data-Set-Name
Version identifier:    Version
Date created:          yyyy/mm/dd hh:mm:ss
Last updated:          yyyy/mm/dd hh:mm:ss
Last copied:           yyyy/mm/dd hh:mm:ss
Space used:            number-of-bytes
Maximum node ID:      number
Deleted objects:      number
Data file size used:  number
```

Data Component Information

```
Dataset name:          Data-Set-Name
Version identifier:    Version
Catalog identifier:    Identifier
Date created:          yyyy/mm/dd hh:mm:ss
Last updated:          yyyy/mm/dd hh:mm:ss
Last copied:           yyyy/mm/dd hh:mm:ss
End-of-file:           number-of-bytes
Highest valid RID:    number
```

Space Utilization Summary Information

Object Type	Records	Space
Tables	189	314496
Columns	5245	4028160
Fragments	282	134144
Indexes	447	457728
Keys	552	141312
Views	2	768
View dependents	2	512
Routines	32	32768
Parameters	127	32512
DB authorizations	10	1280
Table authorizations	183	46848
Routine authorizations	32	8192
User authorizations	15	1920
Indexing	33	135168
Catalog identifier	1	128
Free space	2	14976
Total	7134	5350912

The summary report has two main categories:

- Information about the index component of the metadata catalog.
- Information about the data component and the different kinds of objects that are stored in the data component.

The following fields give information about the index component of the metadata catalog:

Dataset name

The name of the data set for the index component.

Version identifier

The version of the metadata catalog. For a V11.3 metadata catalog, the version identifier is reported as 11.03.00.

Date created

The date and time that the index component of the metadata catalog was created. The date and time are displayed in US English format. The creation date and time displayed for the data component should match the date and time that the index component was created.

Last updated

The date and time the index component was last updated, which corresponds to the date and time the last DDL statement was successfully executed for this metadata catalog. The date and time are displayed in US English format. The last update date and time displayed for the data component should match the date and time that the index component was last updated.

Last copied

The date and time that the index components contents was last replaced by a copy operation, or N/A if the metadata catalog has never been the target of a copy operation. The date and time are displayed in US English format. The value displayed for the data component should match the value displayed for the index component.

Space used

Identifies how much of the index component has been used, in bytes. The number is not related to the data set allocation size or the current physical size of the index component (that is, primary space allocation and extents). The space used is computed based on the number of logical records that exist within the index component. Each logical record is 64-bytes long, so that total spaced used should be divisible by 64.

Maximum node ID

The maximum node ID value identifies how many logical records exist in the index component. Each logical record is referred to as a node.

Deleted objects

The index component is used to track the number of metadata catalog objects (tables, views, indexes, and so on) that have been deleted from the metadata catalog. The deleted objects count identifies how many deleted metadata catalogs objects exist in the metadata catalog. You can use the REORG function to physically reclaim this unused space.

Data file size used

The data file size used number identifies how many bytes in the data component are currently being used. Used in this context means the space is either being actively used for an existing object, or represents “free space” because the object has been deleted. Like the space used value, the data file size used value has no relationship to the current physical DASD allocation size of the data component.

The following fields give information about the data component of the metadata catalog:

Dataset name

The name of the data set for the data component.

Version identifier

The version of the metadata catalog. For a V11.3 metadata catalog, the version identifier is reported as 11.03.00.

Catalog identifier

A string value that is stored in the first logical record of the data component that identifies which version of the software was used to create the metadata catalog.

The following values can appear:

- V11.3 metadata catalog – IBM InfoSphere Classic V11.3 *build-date*
- V11.1 metadata catalog – IBM InfoSphere Classic V11.1 *build-date*
- V10.1 metadata catalog – IBM InfoSphere Classic V10.1 *build-date*
- V9.5 metadata catalog – IBM InfoSphere Classic V9.5 *build-date*
- V9.1 metadata catalog – IBM WebSphere Classic V9.1 *build-date*

The build-date takes the form *mmdyyy* and is updated for major releases and roll-up PTFs.

Date created

The date and time that the data component of the metadata catalog was created. The date and time are displayed in US English format. The creation date and time displayed for the data component should match the date and time that the index component was created.

Last updated

The date and time the data component was last updated, which corresponds to the date and time the last DDL statement was successfully executed for this metadata catalog. The date and time are displayed in US English format. The last update date and time displayed for the data component should match the date and time that the index component was last updated.

Last copied

The date and time that the data components contents was last replaced by a copy operation, or N/A if the metadata catalog has never been the target of a copy operation. The date and time are displayed in US English format. The value displayed for the data component should match the value displayed for the index component.

End-of-file

Number of physical bytes stored in the data component. For V9.1 and later releases of the system catalogs this value should match the Data file size used value displayed in the index component section of the report. If it does not this implies the catalog has been corrupted. For a V8.x version of the metadata catalog the end-of-file value must not match the value for *Data file size used*. The End-of-file value is used to compute the value of the highest valid RID value.

Highest valid RID

Identifies the highest valid record identifier (RID) that can be used to reference a record in the metadata catalog. RIDs are used to establish inter-record relationships within the metadata catalog. When a RID reference value is larger than the number *Highest valid RID number* that RID is identified as being corrupted. It is not valid because it references a record that does not exist in the metadata catalog.

Space Utilization Summary Information

The space utilization summary information section of the report displays a table listing the different kinds of objects that are stored in the system catalog. For each object type, this section displays the number of records that exist and the total space used for each object type. A summary line is also printed that identifies the number of records that exist in the data

component and the total space that is currently being used in the data component. The total size should match the *Data file size used* number printed in the index component section of the summary report.

Object type	Description	Code
Tables	System tables and user tables created by a CREATE TABLE statement.	TAB
Columns	Columns associated with a table or view definition.	COL
Fragments	Objects that exist within a table to manage record arrays; or for CA-IDMS tables the records referenced by the table and for IMS tables the segments referenced by the table definition.	FRG
Indexes	Indexes automatically created for the system tables and user indexes created by a CREATE INDEX statement.	IDX
Keys	SYSIBM.SYSKEYS rows created when a column is referenced in a CREATE INDEX statement.	KEY
Views	View definitions created using the CREATE VIEW statement.	VEW
View dependents	SYSIBM.SYSVIEWDEP rows created to manage a CREATE VIEW statement.	VDP
Routines	System stored procedure definitions automatically created in the metadata catalog and user stored procedure definitions created by a CREATE PROCEDURE statement.	RTN
Parameters	SYSIBM.SYSPARMS rows created when a parameter is defined in a CREATE PROCEDURE statement.	PRM
DB authorizations	SYSIBM.SYSDBAUTH rows that were created due to DBMS GRANT/REVOKE statements.	ADB
Table authorizations	SYSIBM.SYSTABAUTH rows that were created due to table GRANT/REVOKE statements.	ATB
Routine authorizations	SYSIBM.SYSROUTINEAUTH rows that were created due to routine GRANT/REVOKE statements.	ART
User authorizations	SYSIBM.SYSUSERAUTH rows that were created due to user GRANT/REVOKE statements.	AUS
Indexing	Internally created index records used to optimize access to the contents of the metadata catalog.	SIX
Catalog identifier	Catalog identifier record. This is the first record in the data component and is used to record creation and last updated information.	IRD
Free space	Free space records that are available for reuse.	FRE

Object detail reports:

When you run a REPORT operation and add to the PARM parameter the DETAIL, VALIDATE, or DEBUG keywords, the metadata catalog analysis report also includes an object detail report.

The report lists two lines of information for each logical record that is stored in the data component of the metadata catalog. Page breaks occur after 50 lines of information corresponding to about 25 metadata catalog objects.

The following example shows the format of the object details report:

Date: yyyy/mm/dd metadata catalog Analysis Report Page 2
 Time: hh:mm:ss Object Detail

Record	RID	Size	Type	Name	Authorization		Column	Table	Frag.		
					Next	Prev	Start	End / Parm	Index	Other	
1	0	128	IRD	IBM InfoSphere Classic V11.3	N/A	N/A	N/A	N/A	N/A		
2	1	1664	TAB	SYSIBM.SYSTABLES	200	41135	0	0	14	2596	0
3	14	768	COL	SYSIBM.SYSTABLES.NAME	20	1	N/A	N/A	1	N/A	N/A
4	20	768	COL	SYSIBM.SYSTABLES.CREATOR	26	14	N/A	N/A	1	N/A	N/A
5	26	768	COL	SYSIBM.SYSTABLES.TYPE	32	20	N/A	N/A	1	N/A	N/A
6	32	768	COL	SYSIBM.SYSTABLES.DBNAME	38	26	N/A	N/A	1	N/A	N/A

The data component of the metadata catalog is organized as a set of logical records. Each logical record has a record ID (RID) associated with it and a logical record number. The minimum logical record size is 128-bytes and all logical records are an integral multiple of 128. RIDs start at zero and represent 128-byte increments in the data file. Internally, for inter-object relationships the RID is used to identify the “source” or “target” record. Therefore, given a RID the physical starting offset and the logical record can be computed in the data component.

For each logical record two lines of information are displayed. The following information is displayed on the first line:

Record

Identifies the logical record number for the data component object.

RID

Identifies the logical records computed record identifier (RID.) A ‘C’ can be appended after the RID to indicate that corruption has been detected in the metadata catalog for the record being listed.

Size

Identifies the length of the logical record in bytes. This value must be an integral multiple of 128.

Type

Identifies the type of logical record. One of the values shown in the Detail Type column above in Table 4.12-7.

Name

Identifies the external or internal name of the object. The following table identifies the different types of object names that can be displayed and their formats.

Object type	Name
TAB	Qualified table name: <i>owner-name.table-name</i>
COL	Qualified column name: <i>owner-name.table-name.column-name</i>

Object type	Name
FRG	Depends on the type of fragment, as follows: <ul style="list-style-type: none"> Record Array Fragment Definitions – Fragment ID and level number CA-IDMS tables – record name IMS tables – segment name Table-level fragments for tables other than CA-IDMS or IMS – Fragment ID and level number System fragments – 'SYSTEM'
IDX	Qualified index name: <i>owner-name.index-name</i>
KEY	Qualified key name: <i>owner-name.index-name.column-name</i>
VEW	Qualified view name: <i>owner-name.view-name</i>
VDP	Qualified view name and view dependent number: <i>owner-name.view-name(number)</i>
RTN	Qualified stored procedure name: <i>owner-name.routine-name</i>
PRM	Qualified parameter name: <i>owner-name.routine-name.parameter-name</i>
ADB	User and database class: <i>user-name.database-class</i>
ATB	Qualified table name and user: <i>owner-name.table-name.authorization-ID</i>
ART	Qualified stored procedure name and user: <i>owner-name.routine-name.authorization-ID</i>
AUS	Qualified object name and user: <i>owner-name.table-name.authorization-ID</i>
SIX	Internal information
IRD	Catalog identifier display in Summary section of the report
FRE	N/A

The second line of information displayed for a metadata catalog object consists of structural RID information that is stored in the record. The report lists RID information that is associated with most types of catalog objects. When there is no corresponding RID information for the object, the value N/A is displayed. A 'C' can be appended to the end of a RID to indicate that the RID value is invalid or a corruption has been detected in the referenced object.

This information is primarily for use by support personnel. The following RID information is displayed for each logical record:

Next The next logical RID number that that the logical record points to. Most of the metadata catalog objects are maintained as some form of linked list structure.

Prev The previous logical RID number that that the logical record points to. Most of the metadata catalog objects are maintained as some form of linked list structure.

Authorization Start & End

Identifies the first and last authorization record RIDs that are used to manage access to the object.

Column / Parm

Identifies the first RID of the list of columns, keys, or parameters that make up the owning object.

Table or Index

Identifies the RID of a related object. For example, the first index associated with a table or the first view dependent record associated with a view.

Frag or Other

Identifies the RID of a related object. For example, the first fragment definition associated with a table, or the first indexing record (SIX) associated with a system index definition.

The metadata utility

The metadata utility is a z/OS CLI-based application that connects to a Classic data server and updates the metadata catalogs with the contents of DDL statements that are read from a SYSIN input stream.

The metadata utility accepts as input the DDL that is generated by Classic Data Architect.

Running the metadata utility

The metadata utility executes as a z/OS batch job. Sample JCL to execute the metadata utility is distributed as member name CACMETAU in the SCACSAMP library. The name of the load module for the metadata utility is CACMETA.

Before you begin

You must know the name of the query processor that you want the metadata utility to connect to. You must also know the user ID and password required for a connection.

The query processor that you want to connect to must be running.

About this task

The names of the sample JCL and load modules distributed in V11.3 are the same as those used in prior releases of the metadata utility. You cannot use an earlier release of the metadata utility JCL to run V11.3 of the utility without modification. Use the *USERHLQ.USERSAMP(CACMETAU)* sample for V11.3.

MSGCAT DD

The metadata utility JCL must contain an MSGCAT DD statement that references the message catalog. The message catalog is accessed by the CLI component and the metadata utility to retrieve the text for error messages reported by the Classic data server and error conditions detected by CLI or by the metadata utility.

SYSOUT DD

The SYSOUT DD statement is used to record a summary of the processing performed by the metadata utility.

SYSPRINT DD

As the metadata utility reads input records and executes each statement, the metadata utility echoes each statement and the execution status of each statement out to the SYSPRINT DD.

SYSIN DD

The sample JCL uses data set concatenation on the SYSIN DD statement to provide the CONNECT TO SERVER statement. The metadata utility uses the statement to configure the run-time environment so that the CLI

interface can connect to the correct Classic data server to process the DDL statements. The DDL statements are referenced in the second data set referenced by the SYSIN DD statement and referenced by the DDLIN substitution variable. A sample CONNECT TO SERVER statement is provided in SCACCONF sample member CACMUCON.

The file referenced by the SYSIN DD statement is treated as a text input stream, and can be in fixed length or variable length format. There is no restriction on the record length.

DDLOUT DD

The DDLOUT DD statement can be used to support GENERATE DDL statements. If you specify this statement, the metadata utility will write DDL to the specified DDLOUT file for any successful GENERATE DDL statements that are received from SYSIN. If the statement is not specified, any generated DDL is written to SYSPRINT.

Procedure

1. Open for editing member CACMETAU in the SCACSAMP data set and make these changes:
 - a. Provide a job card that is valid for your site.
 - b. Change the CAC parameter to the installed high-level qualifier.
 - c. If you are importing DB2 definitions, uncomment the DB2 parameter and change the parameter to the correct high-level qualifier.
 - d. Customize the connection statement in member CACMUCON to point to the Classic data server with the metadata catalog that you want to update. See “CONNECT TO SERVER statement for the metadata utility” on page 391.
 - e. Change the DDLIN parameter to the member that contains the DDL statements to process.
 - f. If you intend to run GENERATE DDL statements and you want to write the output to a file, add a DDLOUT DD statement that points to the file that will contain the generated DDL.
 - g. If you need to process large DDL statements, change the RGN parameter to change the region size. Increase the region size in increments of two megabytes.
2. Submit the job.

Results

As the metadata utility reads input records, it echoes them out to the SYSPRINT DD statement.

A header is printed for each statement in the SYSIN input file. Each input record read from SYSIN is echoed in the SYSPRINT file. The line number appears in front of the statement text. The line numbers increase monotonically. When the SYSIN data set has a record length that is greater than 80-bytes, the SYSIN input is wrapped in 80-character increments. Wrapped lines do not have line numbers.

After the statements, the output specifies whether the statements ran successfully or failed. Next, any error, warning, or information messages associated with the execution of the statements appear.

Example

The following example shows sample SYSPRINT output:

```
LINE NO.    STATEMENT
   74    CONNECT TO SERVER "CACSAMP" "TCP/0.0.0.0/9087"
   75    USERID USER01;
CACM001I  SQLCODE = 0, INFO: Statement execution was successful.
LINE NO.    STATEMENT
   77    DROP TABLE "CAC"."EMPLOYEE";
CACM001I  SQLCODE = 0, INFO: Statement execution was successful.
LINE NO.    STATEMENT
   78
   79    CREATE TABLE "CAC"."EMPLOYEE" DBTYPE VSAM
   80    DS "USER01.EMPLOYEE"
   81    (
   82
   83    "ENAME" SOURCE DEFINITION
   84    DATAMAP OFFSET 0 LENGTH 20
   85    DATATYPE C
   86    USE AS CHAR(20),
   87    "PHONE" SOURCE DEFINITION
   88    DATAMAP OFFSET 20 LENGTH 4
   89    DATATYPE UF
   90    USE AS INTEGER,
   91    "MAILID" SOURCE DEFINITION
   92    DATAMAP OFFSET 25 LENGTH 6
   93    DATATYPE C
   94    USE AS CHAR(6),
   95    "SALARY" SOURCE DEFINITION
   96    DATAMAP OFFSET 31 LENGTH 4
   97    DATATYPE P
   98    USE AS DECIMAL(7 , 2),
   99    "JOBID" SOURCE DEFINITION
  100    DATAMAP OFFSET 35 LENGTH 4
  101    DATATYPE D
  102    USE AS REAL,
  103    "EMPID" SOURCE DEFINITION
  104    DATAMAP OFFSET 39 LENGTH 4
  105    DATATYPE UF
  106    USE AS INTEGER,
  107    "DEPTID" SOURCE DEFINITION
  108    DATAMAP OFFSET 43 LENGTH 2
  109    DATATYPE UH
  110    USE AS SMALLINT,
  111    "DEPARTMENT" SOURCE DEFINITION
  112    DATAMAP OFFSET 47 LENGTH 15
  113    DATATYPE C
  114    USE AS CHAR(15) );
CACM001I  SQLCODE = 0, INFO: Statement execution was successful.
LINE NO.    STATEMENT
   115   GRANT DELETE, INSERT, UPDATE, SELECT ON TABLE "CAC"."EMPLOYEE" TO
        PUBLIC;
CACM001I  SQLCODE = 0, INFO: Statement execution was successful.
```

Figure 17. Sample SYSPRINT output

The sample output shows that the metadata utility ran four statements. These statements existed in the SYSIN input stream.

Summary reports that are generated by the metadata utility:

The metadata utility writes error and summary information to the SYSOUT data set.

During normal processing, the output written to SYSOUT consists of a two line “header” that identifies the format of the information displayed. For each statement processed by the metadata utility, the starting and ending line numbers from the SYSIN file are reported. The starting line number identifies the actual line on which the statement started and does not include any comments that might have been encountered before the statement. The last line of the statement is the line where the ; is detected and does not take into account any comment lines that might exist after the statement. Therefore, it is possible to see “jumps” between ending and starting line numbers.

The third column in the summary report output identifies the statements processing status. For example, in the SYSPRINT output, the processing status is 0 if no errors were reported, a negative SQL error code value or a hexadecimal form of a system error message. Additionally, N/A is displayed for generated statements that were not part of the SYSIN input stream, such as DISCONNECT statements.

After the processing status is the name of the statement that was encountered followed by the object name that the statement is referencing. Table 72 identifies the statement types that are supported by the metadata utility. For each entry in the table, the statement name is followed by the type of object name that is extracted from the statement. For each entry the statement class is also identified, which determines how the statement is processed by the metadata utility.

Table 72. Statement types that are supported by the metadata utility

Types of statement	Names of objects	Class
CREATE TABLE	Table name	DDL
CREATE INDEX	Index name	DDL
CREATE PROCEDURE	Procedure name	DDL
CREATE VIEW	View name	DDL
COMMENT ON	Table name, index name, column name, or stored procedure name	DDL
ALTER TABLE	Table name	DDL
DROP INDEX	Index name	DDL
DROP TABLE	Table name	DDL
DROP PROCEDURE	Procedure name	DDL
DROP VIEW	View name	DDL
GRANT	N/A	DDL
REVOKE	N/A	DDL
CONNECT TO DB2	DB2 subsystem name	Connect
CONNECT TO SERVER	Data source name	Connect
IMPORT DB2 TABLE	DB2 table name	Import
IMPORT DB2 VIEW	DB2 view name	Import
IMPORT DB2 INDEX	DB2 index name	Import
DISCONNECT FROM DB2	DB2 subsystem name	Connect
DISCONNECT FROM SERVER	Data source name	Connect

DB2 object names can be longer than Classic object names. The SYSOUT output is designed to display in 80-column format. If the object name is greater than an 80-character line, the name is wrapped to the next line of output until the entire contents of the name displays. For these continued object name lines, the Processing Status contains a + (plus) sign. The DISCONNECT FROM statement is a pseudo statement generated by the metadata utility. The DISCONNECT FROM statement is automatically generated during the metadata termination process when a connection exists with DB2 or a Classic data server, or when the SYSIN input stream contains multiple CONNECT TO DB2 or data server statements.

If a statement is encountered that is not in Table 72 on page 388, then the metadata utility issues error 0x00760002 (Unsupported statement encountered by the metadata utility: start-of-statement). The first 25 characters of the statement are displayed in the summary report, and the object name is set to UNKNOWN. The line numbers that display for this unknown/unsupported statement start with the first line of the statement. The ending line number is where a ; is encountered or end-of-file is reached. The contents of this statement are also echoed in SYSPRINT followed by the 0x00760008 error message text and the first 70 characters of the statement.

For supported statements, a single line is generated in the SYSOUT summary report for each statement that was identified in the SYSIN input stream. A single line is also generated in the SYSOUT summary report for any DISCONNECT or COMMENT ON statements generated by the metadata utility. For generated statements, the starting and ending line numbers are identified as N/A. Following each statement summary line, for statements that either reported errors when they were executed or if the statement reported one or more warning messages, information messages, or both, the SYSOUT output echoes the same error, warning, and information messages that display in the SYSPRINT output stream.

Note: COMMENT ON statements are generated when processing DB2 IMPORT statements

Start Line #	End Line #	Processing Status	Statement	Object
1	1	0	CONNECT TO SERVER	CACSAMP
CACM0001I SQLCODE = 0, INFO: Statement execution was successful.				
2	2	-204	DROP TABLE	CAC.EMPLOYEE
CACM0002I SQLCODE = -204, ERROR: CAC.EMPLOYEE is an undefined name.				
3	22	0	CREATE TABLE	CAC.EMPLOYEE
CACM0001I SQLCODE = 0, INFO: Statement execution was successful.				
N/A	N/A	0	DISCONNECT FROM SERVER	CACSAMP
CACM0001I SQLCODE = 0, INFO: Statement execution was successful.				

Figure 18. Sample SYSOUT summary report

Return codes for the metadata utility:

The metadata utility uses the z/OS return code to identify the overall success or failure of its execution.

The following table identifies the return codes that the metadata utility can issue.

Table 73. List of return codes for the metadata utility

Return code	Meaning
20	Environmental setup error detected. Processing is not attempted.
16	Errors detected in the configuration member or while initializing the run-time environment. Processing is not attempted.
12	Error reported connecting to the Classic data server. Processing is not attempted.
8	Error reported by the server, CLI, or metadata utility while processing a statement. Processing continues with the next statement in the SYSIN input stream.
4	Warning messages issued by the server or metadata utility. Statements were processed successfully.
0	No errors were reported.

CONNECT statements for the metadata utility

The metadata utility supports two different types of CONNECT statements.

The metadata utility does not check that CICS connection statements are valid. When it encountered one of these statements, the metadata utility echoes the statement in the SYSOUT output and displays message CACM010I with the following information:

0x00760200 - CONNECT TO CICS statement no longer used.

CONNECT TO DB2 statement for the metadata utility:

The CONNECT TO DB2 statement for the metadata utility uses the following syntax:

```

▶▶CONNECT TO DB2—SUBSYSTEM—subsystem-name—USING PLAN—plan-name—————▶
▶[ACCESS CATALOG PREFIX—prefix] ;—————▶▶▶

```

CONNECT TO DB2

These keywords identify the statement. All subsequent parameters describe the DB2 system and table owner name that the metadata utility queries for information during an IMPORT operation. The identified DB2 system and catalog are used until it is explicitly changed by another CONNECT TO DB2 statement or until the metadata utility terminates.

SUBSYSTEM *subsystem-name*

This clause identifies the DB2 subsystem where the DB2 objects which are referenced in subsequent DB2 IMPORT statements, exist. The subsystem name identifies the DB2 subsystem to which the connection will be made. The name cannot exceed 4 characters in length. The characters permitted in the subsystem name vary with the version of DB2.

USING PLAN *plan-name*

This clause identifies the DB2 application plan that is used to access the target DB2 subsystem identified by *subsystem-name*. *plan-name*. It cannot exceed 8 characters in length, and the first character must be alphabetic. There is no default value.

Accessing DB2 data requires binding an application plan for use by the DB2 Call Attach Facility (CAF) that is used by the metadata utility to access DB2 data. InfoSphere Classic Federation Server for z/OS includes the DB2 database request module (DBRM) that is required for binding the necessary plan. SAMPLIB member CACBCNTL contains a sample DB2 bind statement. SAMPLIB member CACBIND contains sample JCL to process the CACBCNTL input and bind the plan to the Classic supplied DBRM. The Classic DBRM is supplied in SAMPLIB member CACP2EC. To bind a plan, follow these steps:

1. Edit CACBCNTL to change the plan name and DB2 DSN name.
2. Edit CACBIND by making the following changes:
 - Provide a valid job card.
 - Change the CAC variable to the high-level qualifier of your Classic installation.
 - Change the DB2 variable to the high-level qualifier of your DB2 installation.
 - Run the CACBIND JCL.

ACCESS CATALOG PREFIX *prefix*

This optional clause identifies the prefix of the DB2 catalog. Do not enter this keyword phrase if you want to accept the default value SYSIBM as the prefix. If importing a table, view, or alias, the metadata utility uses the prefix to access the DB2 catalog tables 'prefix.SYSTABLES' and 'prefix.SYSCOLUMNS'. If importing an index, the metadata utility uses the prefix to access the DB2 catalog tables 'prefix.SYSINDEXES' and 'prefix.SYSKEYS'.

CONNECT TO SERVER statement for the metadata utility:

The metadata utility uses the information in this statement to connect to a query processor and update the content of a metadata catalog.

The CONNECT TO SERVER statement uses the following syntax:

```

▶▶CONNECT TO SERVER—datasource-name—connection-URL—USERID—user-ID
PASSWORD—password;

```

Parameters

datasource-name

A 1- to 16-character native identifier that names the query processor (data source) to connect to in the Classic data server identified by *connection-URL*. *datasource-name* is used until it is explicitly changed by another CONNECT TO SERVER statement or until the metadata utility terminates.

connection-URL

A native identifier that provides information used to connect to a Classic data server. The URL must be delimited and match the connection information of an active TCP/IP connection handler in the Classic data server.

USERID *user-ID*

This optional clause specifies the user ID for connecting to the Classic data server. The default value is the TSO user ID associated with the metadata utility job. The user ID is a 1- to 7-character short-native identifier.

PASSWORD *password*

This clause is required when the Classic data server has the SAF exit active. *password* provides authentication information when the connection with the Classic data server is established. By default, the connection is established without a password.

If the PASSWORD clause is specified, the password is a 16-character DES encrypted character string specified in hexadecimal format. This password needs to be generated using the password generator utility on Windows or on z/OS.

IMPORT DB2 statements for the metadata utility in Classic federation

The metadata utility uses the IMPORT DB2 statements to extract the definition of one or more DB2 objects from a DB2 system, which is identified by a preceding CONNECT TO DB2 statement. Based on the information that it retrieves, the metadata utility builds a CREATE TABLE or CREATE INDEX statement that defines the DB2 object in a metadata catalog.

For each table or index that is created, the metadata utility generates a single COMMENT ON statement that identifies the name of the DB2 source object. Individual column level comment information is not created because the DB2 and Classic column names are identical.

IMPORT DB2 TABLE or VIEW statement for the metadata utility:

Use this statement for importing alias, table, materialized query tables, or view definitions.

The IMPORT DB2 TABLE or VIEW statement uses the following syntax:

```

▶▶—IMPORT DB2—

|       |
|-------|
| TABLE |
| VIEW  |

—[—DB2-owner-name—.]—DB2-table-name—————▶

```



```

▶

|                                      |              |
|--------------------------------------|--------------|
| [—RENAME AS—owner-name—.]—table-name | WITH INDEXES |
|--------------------------------------|--------------|

—————▶

```

Parameters

[DB2-owner-name.]DB2-table-name

After identifying the type of DB2 object to import, you specify the name of that DB2 object. You can specify the name with or without delimiters.

You can qualify the DB2 object with *DB2-owner-name*. If you do not qualify the object, the default qualifier is the TSO user ID associated with the metadata utility job.

DB2-owner-name and *DB2-table-name* can each be 128-characters long because the metadata utility supports DB2 Version 8 “long” names. However, if the DB2 subsystem identified in the CONNECT TO DB2 statement is a version prior to V8, the maximum length that you can specify for *DB2-owner-name* is 8 characters. The maximum length you can specify for *DB2-table-name* is 18 characters. If you supply a long name but are

using a DB2 subsystem earlier than V8, the IMPORT operation returns no information for the DB2 object and fails.

RENAME AS [*owner-name*].*table-name*

The RENAME AS clause specifies the name of the table to define in the metadata catalog for the DB2 object. The name can be up to 18 characters long. You can specify *table-name* as a delimited or undelimited identifier.

You can also qualify the name or leave it unqualified. The qualifier can be up to 8-characters long. The default qualifier is the user ID obtained from the CONNECT TO SERVER statement that was used to establish the connection with the Classic data server. If you do not specify the user ID, the default value is the TSO user ID that is associated with the metadata utility job. Because the DB2 default owner is based on the TSO user ID that executes the metadata utility, it is possible for the owner name in this clause to differ from the DB2 default owner name.

The owner name cannot be SYSIBM or SYSCAC.

You must use the RENAME AS clause when either of the two following conditions is true:

- The name of the DB2 object already exists in the metadata catalog and does not refer to a DB2 object, or refers to a different DB2 object.
- The IMPORT DB2 statement refers to a DB2 object with a long name. The metadata catalog does not support long names.

WITH INDEXES

This clause requests that DB2 index definitions automatically be defined for the DB2 table in the metadata catalog. Indexes are defined with the DB2 index name. They are associated with the table name that was specified in the RENAME AS clause or the DB2 table name, if a RENAME AS clause was not specified. The automatic creation of indexes is attempted only if the execution of the CREATE TABLE statement is successful.

If execution of the CREATE TABLE statement is successful, the metadata utility internally generates a DB2 IMPORT INDEX statement for each DB2 index that is defined.

If the INDEXES that are associated with the DB2 table contain long names, you cannot use the WITH INDEXES clause. You must use a separate IMPORT DB2 INDEX statement with a RENAME clause to import indexes with long names.

When processing a DB2 IMPORT table statement, the metadata utility access the DB2 catalog and reads the SYSIBM.SYSTABLES and SYSIBM.SYSCOLUMNS tables to gather information for the table to be imported. It also reads the SYSIBM.SYSINDEXES table if the WITH INDEXES option is specified. From this information, the metadata utility generates a CREATE TABLE statement and submits that to the Classic data server to be processed. If the processing is successful, the metadata utility generates a COMMENT ON statement to document the original DB2 table name and submits that statement to the Classic data server for processing.

If the WITH INDEXES option is specified, the metadata utility generates an IMPORT DB2 INDEX statement for each index that is defined on the original DB2 table. Generating this statement results in a CREATE INDEX and COMMENT ON statement being generated for each index that is defined on the original table.

Sample table import SYSPRINT output using a table named DNS8710.EMP:

```
LINE NO.    STATEMENT

          1    CONNECT TO SERVER CACSAMP TCP/0.0.0.0/5026
CACM0001I  SQLCODE = 0, INFO: Statement execution was successful.

LINE NO.    STATEMENT

          2    CONNECT TO DB2 SUBSYSTEM DSN1 USING PLAN CACPLAN;
CACM0001I  SQLCODE = 0, INFO: Statement execution was successful.

LINE NO.    STATEMENT

          3    IMPORT DB2 TABLE DNS8710.EMP RENAME AS CAC_DB2.EMP WITH INDEXES;
CACM0001I  SQLCODE = 0, INFO: Statement execution was successful.

LINE NO.    STATEMENT

          CREATE TABLE "CAC_DB2"."EMP" TABLE "DNS8710"."EMP"
          SUBSYSTEM "DSN1" USING PLAN "CACPLAN"
          ("EMPNO" SOURCE DEFINITION COLUMN "EMPNO"
           COLTYPE CHAR(6) USE AS CHAR(6) NOT NULL,
           "FIRSTNAME" SOURCE DEFINITION COLUMN "FIRSTNAME"
           COLTYPE VARCHAR(12) USE AS VARCHAR(12) NOT NULL,
           "MIDINIT" SOURCE DEFINITION COLUMN "MIDINIT"
           COLTYPE CHAR(1) USE AS CHAR(1) NOT NULL,
           "LASTNAME" SOURCE DEFINITION COLUMN "LASTNAME"
           COLTYPE VARCHAR(15) USE AS VARCHAR(15) NOT NULL,
           "WORKDEPT" SOURCE DEFINITION COLUMN "WORKDEPT"
           COLTYPE CHAR(3) USE AS CHAR(3),
           "PHONENO" SOURCE DEFINITION COLUMN "PHONENO"
           COLTYPE CHAR(4) USE AS CHAR(4),
           "HIREDATE" SOURCE DEFINITION COLUMN "HIREDATE"
           COLTYPE DATE USE AS DATE,
           "JOB" SOURCE DEFINITION COLUMN "JOB"
           COLTYPE CHAR(8) USE AS CHAR(8),
           "EDLEVEL" SOURCE DEFINITION COLUMN "EDLEVEL"
           COLTYPE SMALLINT USE AS SMALLINT,
           "SEX" SOURCE COLUMN DEFINITION "SEX"
           COLTYPE CHAR(1) USE AS CHAR(1),
           "BIRTHDATE" SOURCE COLUMN DEFINITION "BIRTHDATE"
           COLTYPE DATE USE AS DATE,
           "SALARY" SOURCE COLUMN DEFINITION "SALARY"
           COLTYPE DECIMAL(9,2) USE AS DECIMAL(9,2),
           "BONUS" SOURCE COLUMN DEFINITION "BONUS"
           COLTYPE DECIMAL(9,2) USE AS DECIMAL(9,2),
           "COMM" SOURCE COLUMN DEFINITION "COMM"
           COLTYPE DECIMAL(9,2) USE AS DECIMAL(9,2),
           PRIMARY KEY("EMPNO"));
CACM0001I  SQLCODE = 0, INFO: Statement execution was successful.

LINE NO.    STATEMENT

          COMMENT ON TABLE "CAC_DB2"."EMP"
          IS "Source DB2 table is DSN8710.EMP";
CACM0001I  SQLCODE = 0, INFO: Statement execution was successful.

LINE NO.    STATEMENT

          IMPORT DB2 INDEX DSN8710.XEMP1 DEFINED ON TABLE CAC_DB2.EMP;
CACM0001I  SQLCODE = 0, INFO: Statement execution was successful.

LINE NO.    STATEMENT

          CREATE UNIQUE INDEX "DSN8710"."XEMP1"
```

```

        ON "CAC_DB2"."EMP"
        ("EMPNO" ASC);
CACM0001I SQLCODE = 0, INFO: Statement execution was successful.
LINE NO.    STATEMENT

        COMMENT ON INDEX "DSN8710"."XEMP1"
        IS 'Source DB2 index is DSN8170.XEMP1';
CACM0001I SQLCODE = 0, INFO: Statement execution was successful.

LINE NO.    STATEMENT

        IMPORT DB2 INDEX DSN8710.XEMP2 DEFINED ON TABLE CAC_DB2.EMP;
CACM0001I SQLCODE = 0, INFO: Statement execution was successful.

LINE NO.    STATEMENT

        CREATE INDEX "DSN8710"."XEMP2"
        ON "CAC_DB2"."EMP"
        ("WORKDEPT" ASC);
CACM0001I SQLCODE = 0, INFO: Statement execution was successful.

LINE NO.    STATEMENT

        COMMENT ON INDEX "DSN8710"."XEMP2"
        IS 'Source DB2 index is DSN8170.XEMP2';
CACM0001I SQLCODE = 0, INFO: Statement execution was successful.

LINE NO.    STATEMENT

        DISCONNECT FROM DB2 DSN1
CACM0001I SQLCODE = 0, INFO: Statement execution was successful.

LINE NO.    STATEMENT

        DISCONNECT FROM SERVER CACSAMP;
CACM0001I SQLCODE = 0, INFO: Statement execution was successful.

```

IMPORT DB2 INDEX statement for the metadata utility:

When you import a DB2 table definition, you can use this statement to request that the indexes defined on the table in DB2 also be defined within a metadata catalog.

The IMPORT DB2 INDEX statement uses the following syntax:

```

▶▶—IMPORT DB2 INDEX—[—DB2-owner-name—.]—DB2-index-name—————▶
|
|_____DEFINED ON TABLE—table-name—————▶
|
|—RENAME AS—index-name—|

```

Parameters

[DB2-owner-name.]DB2-index-name

You can specify the name of that DB2 index with or without delimiters.

You can also qualify the DB2 index with *DB2-owner-name*. If you do not qualify the object, the default qualifier is the TSO user ID associated with the metadata utility job.

DB2-owner-name and *DB2-table-name* can each be 128-characters long because the metadata utility supports DB2 Version 8 “long” names. However, if the DB2 subsystem identified in the CONNECT TO DB2 statement is a version prior to V8, the maximum length that you can

specify for *DB2-owner-name* is 8 characters. The maximum length that you can specify for *DB2-table-name* is 18 characters. If you supply a long name but are using a DB2 subsystem earlier than V8, the IMPORT operation returns no information for the DB2 index and fails.

RENAME AS *index-name*

This clause specifies the name of the index to define in the metadata catalog. The name can be up to 18-characters long. You can specify *index-name* as a delimited or undelimited identifier.

You can also qualify the name or leave it unqualified. The qualifier can be up to 8-characters long. The default qualifier is the user ID obtained from the CONNECT TO SERVER statement that was used to establish the connection with the Classic data server. If you do not specify the user ID, the default value is the TSO user ID that is associated with the metadata utility job. Because the DB2 default owner is based on the TSO user ID that executes the metadata utility, the owner name in this clause can differ from the DB2 default owner name.

The owner name cannot be SYSIBM or SYSCAC.

You must use the RENAME AS clause when either of the following conditions is true:

- The name of the DB2 index already exists in the metadata catalog and does not refer to a DB2 index, or refers to a different DB2 index.
- The IMPORT DB2 INDEX statement refers to a DB2 index with a long name. The metadata catalog does not support long names.

DEFINED ON TABLE [*owner-name*].*table-name*

This clause specifies the name of the logical table that is referenced in the CREATE INDEX statement. You can specify the table name as a delimited or undelimited identifier.

You can also qualify the name or leave it unqualified. The qualifier can be up to 8-characters long. The default qualifier is the user ID obtained from the CONNECT TO SERVER statement that was used to establish the connection with the Classic data server. If you do not specify the user ID, the default value is the TSO user ID that is associated with the metadata utility job. Because the DB2 default owner is based on the TSO user ID that executes the metadata utility, the owner name in this clause can differ from the DB2 default owner name.

GENERATE DDL statement for the metadata utility

To generate DDL from the catalog, you can use Classic Data Architect. You can also use the metadata utility directly on the z/OS system. The metadata utility can generate the same kinds of objects that Classic Data Architect can generate.

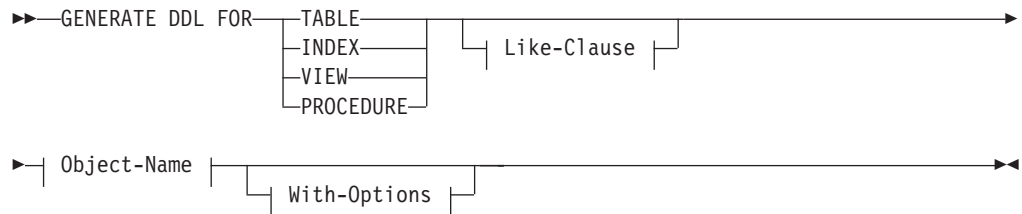
Table 74. Generated Statements supported for each object type

Generated DDL	Object
DROP statements	TABLE, VIEW, INDEX, PROCEDURE
CREATE statements	TABLE, VIEW, INDEX, PROCEDURE
COMMENT ON statements	TABLE, VIEW, INDEX, PROCEDURE
GRANT statements	TABLE, VIEW, INDEX, PROCEDURE
ALTER statements	TABLE, VIEW
Indexes	TABLE

One situation where you might choose to create DDL from the metadata utility is in a migration. For example, you could generate DDL for objects on a test system and import those objects onto a production system. The DDL is output to the media that you specify for subsequent processing. The generated DDL is formatted for direct input into the metadata utility for creating or modifying catalog entries.

To generate DDL, the metadata utility first connects to an appropriate Classic data server specified in a `CONNECT TO SERVER` statement. Once connected, the catalog is accessible for queries necessary to gather data for the `GENERATE DDL` statements. The generated DDL is printed to `SYSPRINT` or recorded in the file associated with the `DDLOUT DD` in the `CACMETAU JCL`.

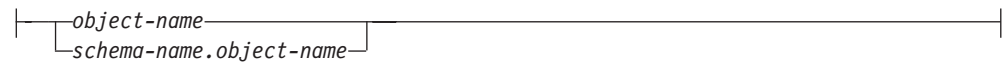
Syntax



Like-Clause:



Object-Name:



With-Options:



Parameters

Like-Clause:

LIKE

LIKE ESCAPE *char* NOT LIKE ESCAPE *char*

Specify this clause when the object name contains wildcard characters. When the object name contains an underscore name character, an escape character must be defined to preserve the context. Once defined, place the escape character immediately preceding the underscore character to indicate that it is not a wildcard character.

```
GENERATE DDL FOR TABLE LIKE ESCAPE '!' "USER1"."IMS!_TAB%"  
WITH DROP,INDEX;
```

Object-Name:

object-name
schema-name.object-name

Specify the object with or without a schema name. If schema name is not specified, the schema defaults to the user ID assigned to the metadata utility job. When you specify objects after a like clause, you can use wildcard characters in the schema or object name. The supported wildcard characters are the same as those characters that are supported by the Query Processor (underscore and percent sign).

System catalog objects with a schema name of SYSIBM or SYSCAC cannot be generated using the GENERATE DDL statement. When the like clause is used, all system objects that qualify for the like processing are ignored. If you attempt to generate a specific object in the SYSIBM or SYSCAC schema, the following error is returned:

```
CACM004I Non-SQLCODE = 0x0071001E, ERROR: You cannot create or  
generate a catalog object with an owner of SYSIBM or SYSCAC.
```

With-Options:

WITH *option*

The with options clause is used to describe additional DDL statement types to be included with the CREATE *object* statement that is generated. You can specify additional statement types individually or you can specify ALL, which generates all additional statement types that are applicable for the CREATE *object* statement. The Table 74 on page 396 table shows the valid additional statement types that can be generated.

A redundant or duplicate specification in the with options clause returns an error during parsing (for example, if other options are specified with ALL). A specification of an invalid option returns an error during parsing (for example, when an index is specified on a view).

Considerations for the DDL output

- By default, the generated DDL is written to SYSPRINT in the standard output format for the metadata utility report.

If you want to generate the DDL to a file, allocate the file to the DDLOUT DD in the CACMETAU JCL. When the metadata utility is run, the DDL is written to the data set referenced by the DDLOUT DD instead of SYSPRINT. And, SYSPRINT contains the following informational message for each statement:

```
CACM001I SQLCODE = 0, INFO: The generated DDL was successfully written to  
DDLOUT.
```

- As with other metadata utility statements, you can choose to set up the SYSIN files for CACMETAU as two concatenated LRECL 80 files. The first file contains the CONNECT TO SERVER statement and the second contains the statements to run. For example:

```
//SYSIN DD DISP=SHR,DSN=&USRHLQ..SCACCONF(&CONNECT)  
// DD DISP=SHR,DSN=&USRHLQ..SCACSAMP(&DDLIN)
```

Note: When concatenating with SYSIN, ensure that the files are consistent in their LRECL definitions.

The metadata utility can handle a SYSIN of 80 bytes or greater. Cases where you might require more than 80 bytes:

- If you specify DB2 tables and indexes using DB2 long names. (DB2 long names could require as much as 132 bytes.)
- If you specify COMMENT ON statements with long text strings.

Example 1

Generate DDL for a specific table named SEQ_TABLE1 that was created by USER1. Include a DROP statement in the generated DDL. The generated DDL is recorded in the standard SYSPRINT output file for z/OS.

Provide the following input to SYSIN:

```
GENERATE DDL FOR TABLE "USER1"."SEQ_TABLE1" WITH DROP;
```

SYSOUT contains the following metadata utility output:

```
***** TOP OF DATA *****
Start      End Processing
Line #    Line #      Status Statement          Object
   1         1         0 CONNECT TO SERVER   CACDAS
CACM001I  SQLCODE = 0, INFO: Statement execution was successful.
   2         2         0 GENERATE DDL FOR   "USER1"."SEQ_TABLE1"
CACM001I  SQLCODE = 0, INFO: Statement execution was successful.

   N/A       N/A         0 DISCONNECT FROM SERVER CACDAS
CACM001I  SQLCODE = 0, INFO: Statement execution was successful.
***** BOTTOM OF DATA *****
```

SYSPRINT contains the following metadata utility output:


```

***** TOP OF DATA *****
LINE NO.   STATEMENT

      1   CONNECT TO SERVER CACDAS "TCP/9.30.136.90/5002";
CACM001I  SQLCODE = 0, INFO: Statement execution was successful.

LINE NO.   STATEMENT

      2   GENERATE DDL FOR TABLE "USER1"."SEQ_TABLE1" WITH DROP;
CACM001I  SQLCODE = 0, INFO: Statement execution was successful.

          DROP TABLE "USER1"."SEQ_TABLE1";

CREATE TABLE "USER1"."SEQ_TABLE1" DBTYPE SEQUENTIAL
  DS "USER1.SEQUENTIAL.SEQ.EMPFILE"
(
"ENAME" SOURCE DEFINITION
  DATAMAP OFFSET 0 LENGTH 20
  DATATYPE C
  USE AS CHAR(20),
"PHONE" SOURCE DEFINITION
  DATAMAP OFFSET 20 LENGTH 4
  DATATYPE UF
  USE AS INTEGER,
"MAILID" SOURCE DEFINITION
  DATAMAP OFFSET 25 LENGTH 6
  DATATYPE C
  USE AS CHAR(6),
"SALARY" SOURCE DEFINITION
  DATAMAP OFFSET 31 LENGTH 4
  DATATYPE P
  USE AS DECIMAL(7 , 2),
"JOBID" SOURCE DEFINITION
  DATAMAP OFFSET 35 LENGTH 4
  DATATYPE F
  USE AS REAL,
"EMPID" SOURCE DEFINITION
  DATAMAP OFFSET 39 LENGTH 4
  DATATYPE UF
  USE AS INTEGER,
"DEPTID" SOURCE DEFINITION
  DATAMAP OFFSET 43 LENGTH 2
  DATATYPE UH
  USE AS SMALLINT,
"DEPARTMENT" SOURCE DEFINITION
  DATAMAP OFFSET 47 LENGTH 15
  DATATYPE C
  USE AS CHAR(15) );
CACM001I  SQLCODE = 0, INFO: Statement execution was successful.

***** BOTTOM OF DATA *****

```

Example 2

Generate DDL for table names belonging to USER1 and beginning with the characters IMS_TAB. Include indexes defined on the resultant tables. Generate DROP statements for the resultant tables and indexes. Record the generated DDL in the specified z/OS data set associated with the DDLOUT DD.

Provide the following input to SYSIN, with a DDLOUT DD specified in the CACMETAU JCL:

```

GENERATE DDL FOR TABLE LIKE ESCAPE '!' "USER1"."IMS!_TAB%"
  WITH DROP,INDEX;

```

SYSOUT contains the following metadata utility output:

```
***** TOP OF DATA *****
  Start   End Processing
  Line #  Line #   Status Statement           Object
    1      1         0 CONNECT TO SERVER      CACDAS
CACM001I  SQLCODE = 0, INFO: Statement execution was successful.

    2      3         0 GENERATE DDL FOR        LIKE
CACM001I  SQLCODE = 0, INFO: The generated DDL was successfully written to DDLOUT.

  N/A     N/A         0 DISCONNECT FROM SERVER  CACDAS
CACM001I  SQLCODE = 0, INFO: Statement execution was successful.
***** BOTTOM OF DATA *****
```

SYSPRINT contains the following metadata utility output:

```
***** TOP OF DATA *****
LINE NO.   STATEMENT

    1      CONNECT TO SERVER CACDAS "TCP/9.30.136.90/5002";
CACM001I  SQLCODE = 0, INFO: Statement execution was successful.

LINE NO.   STATEMENT

  2  GENERATE DDL FOR TABLE LIKE ESCAPE '!' "USER1"."IMS!_TAB%"
  3  WITH DROP,INDEX TO DSN:USER1.GENERATE.OUTPUT;
CACM001I  SQLCODE = 0, INFO: The generated DDL was successfully written to DDLOUT
***** BOTTOM OF DATA *****
```

Only one table created by USER1 fulfilled the search criteria for a name beginning with the characters IMS_TAB. The z/OS data set associated with the DDLOUT DD contains the following generated DDL:

```

***** TOP OF DATA *****
DROP INDEX "USER1"."IMS_TABLE_IDX1";

DROP TABLE "USER1"."IMS_TABLE";

CREATE TABLE "USER1"."IMS_TABLE" DBTYPE IMS
  DS "USER1.IMS.IMSFILE"
(
  "NAME" SOURCE DEFINITION
  DATAMAP OFFSET 0 LENGTH 20
  DATATYPE UC
  USE AS CHAR(20),
  "ADDRESS1" SOURCE DEFINITION
  DATAMAP OFFSET 20 LENGTH 20
  DATATYPE UC
  USE AS CHAR(20),
  "ADDRESS2" SOURCE DEFINITION
  DATAMAP OFFSET 40 LENGTH 20
  DATATYPE UC
  USE AS CHAR(20),
  "CITY" SOURCE DEFINITION
  DATAMAP OFFSET 60 LENGTH 12
  DATATYPE UC
  USE AS CHAR(12),
  "STATE" SOURCE DEFINITION
  DATAMAP OFFSET 72 LENGTH 2
  DATATYPE UC
  USE AS CHAR(2),
  "ZIP" SOURCE DEFINITION
  DATAMAP OFFSET 74 LENGTH 5
  DATATYPE UC
  USE AS CHAR(5),
  "PLUS4" SOURCE DEFINITION
  DATAMAP OFFSET 79 LENGTH 4
  DATATYPE UC
  USE AS CHAR(4) );

CREATE INDEX "USER1"."IMS_TABLE_IDX1" ON "USER1"."IMS_TABLE" ("STATE"
  ASC, "ZIP" ASC, "NAME" ASC);

***** BOTTOM OF DATA *****

```

Encrypting passwords for connecting to Classic data servers when the SAF exit is active

If the Classic data server that the metadata utility communicates with has an active SAF exit, the CONNECT TO SERVER statement used by the metadata utility must include both a USERID and PASSWORD clause.

About this task

The password must be encrypted according to the Data Encryption Standard (DES). Use the password encryption utility for Windows or the password encryption utility for z/OS to encrypt the password according to this standard.

- The password generator utility for Windows, cacencr.exe, is a Windows-based command-line utility.
- The password generator utility for z/OS runs as a batch job.

The SAF exit does not use encryption. If your site uses SAF exits you can connect to a server by using the Classic Data Architect. This GUI tool masks your password, but does not use encryption. Supply your z/OS user ID and password.

User-written JDBC and ODBC applications do not use encrypted passwords. If your site uses these applications, you can connect by using a z/OS user ID and password.

Running the password generator utility for Windows:

The password generator utility for Windows verifies the password that you supply, encrypts the password, and writes the encrypted password to the password.txt file in the current directory.

About this task

The utility is installed in the ODBC\bin directory where you installed the Classic ODBC client.

Procedure

1. Update the PATH variable to include the following references to the ODBC installation directories: `PATH=ODBC\lib;ODBC\bin;...`
2. Edit the cac.ini file.
 - a. In a Windows command window, navigate to the ODBC\lib directory.
 - b. Edit the cac.ini file to provide the following required settings based on the location of your Classic ODBC client installation and your locale information:

```
NL CAT = c:\Program Files\IBM\ISClassic113\ODBC\lib
NL = US English
```

3. Set the CAC_CONFIG environment variable to point to the full path of the cac.ini file that you updated in the previous step.

```
SET CAC_CONFIG= c:\Program Files\IBM\ISClassic113\ODBC\lib\cac.ini
```

4. In a Windows command window, navigate to the ODBC\bin directory.
5. Type **cacencr** and press **Enter**.
6. At the prompt, type the password that you want to encrypt. The password encryption utility creates a file named **password.txt** that contains the password in a 16-byte hexadecimal string format.

```
ENCRYPTED=x'<16-byte hexadecimal value>'
```

7. Open the password.txt file in a file editor and copy the hexadecimal string (`x'<16-byte hexadecimal value>`) to the Windows clipboard.
8. In a mainframe session, edit the data set that contains the CONNECT TO SERVER statement used for the metadata utility. Paste the encrypted password after the PASSWORD keyword.

Running the password generator utility for z/OS:

The password generator utility for z/OS verifies the password that you supply, encrypts the password, and writes the encrypted password to the file that the DD PASSWD points to.

About this task

Sample member CACENCRP that runs the utility is distributed in the USERHLQ.SCACSAMP data set.

Procedure

1. Create a new file or a PDS member that only users who should know the encrypted password can access. You must create the file or PDS with the following attributes: RECFM=FB and LRECL=80.
2. Edit the new file or PDS member. Enter the password that you want to encrypt using the following keyword value pair:

```
PASSWORD=<value>
```

The keyword does not have to start in column one.

3. Edit the *userhlq*.SCACSAMP(CACENCRP) JCL member. Change the PASSWD DD statement to point to the file that you updated in the previous step.
4. Submit the *userhlq*.SCACSAMP(CACENCRP) JCL member to generate the encrypted password. The encrypted password is written back to the input file that the DD PASSWD points to. The file will contain the original PASSWORD keyword=value pair and the ENCRYPTED keyword=value:

```
PASSWORD=<value> ENCRYPTED=x'<16-byte hexadecimal value>'
```

5. Browse the file that the DD PASSWD points to. Copy the 16-byte hexadecimal string (x'<16 byte hexadecimal value>') from that file.
6. Edit the data set that contains the CONNECT TO SERVER statement used for the metadata utility. Member *userhlq*.SCACCONF(CACMUCON) contains an example CONNECT TO SERVER statement with a USER ID and PASSWORD.
7. Paste the 16-byte hexadecimal string password that you copied in step 5 after the PASSWORD keyword.

The configuration migration and maintenance utility

The configuration migration and maintenance utility, CACCFGUT, runs as an IBM z/OS batch job that manages configurations for your Classic data servers. You can use the utility for backup and recovery, monitoring, and maintenance.

Features of the configuration migration and maintenance utility

You can use the configuration migration and maintenance utility to manage the configurations of your Classic data servers. The utility can monitor, back up, and recover your configurations.

The utility supports the following configuration-related Master Terminal Operator (MTO) commands:

- EXPORT
- IMPORT,CONFIG
- IMPORT,CONFIG,MIGRATE

The utility also provides a REPORT command.

To run the configuration migration and maintenance utility, enter one or more of the configuration commands on the SYSIN DD statement.

Backups

You can use the utility to run the EXPORT command at any time to create backups of configuration files.

Recovery

You can use the EXPORT and IMPORT commands to restore a configuration environment to a previous point in time. By using the EXPORT command, you can create a command file that is based on the configuration environment of a running Classic data server. You can then use this command file to update a different configuration file by using the IMPORT command.

To restore the full configuration, run the utility offline against empty configuration data sets when the Classic data server is not running. To avoid conflicts with default services, run the utility against empty configuration data sets.

Monitoring

You can use the utility to run the REPORT command at any time to monitor the configuration of the Classic data server.

Running the configuration migration and maintenance utility

To run the configuration migration and maintenance utility, complete the steps described in the procedure.

About this task

You must migrate the master configuration as a single file. If you previously used a concatenation of multiple master configuration members in your VHSCONF data set, consider merging those members into one file before running the utility. Otherwise, global configuration values from the previous configuration might not be migrated. You can use a utility such as IEBGENER to perform this operation. See the IBM z/OS product information for more details about the IEBGENER utility.

You must call the commands that migrate USER or OVERRIDEMEMBERS after the master configuration is migrated.

When you run the configuration migration and maintenance utility for migration purposes, run the utility as a stand-alone process before starting a Classic data server for the first time. The configuration data sets should be empty.

You can specify FILENAME, OVERRIDEMEMBER, or USER values as a DSN, a DD statement, or a PDS member name. When you specify a PDS member name, the migration process defaults to the VHSCONF DD card from the previous version. All configuration members related to the existing configuration must reside in the same PDS.

Sample stand-alone JCL for running the configuration migration and maintenance utility is provided in *USERHLQ.SCACSAMP* member CACCFGUT.

To run the configuration migration and maintenance utility to migrate a configuration from a previous version, enter one or more of the following configuration commands in the specified order on the SYSIN DD statement.

1. Import the master configuration file by using either of the following command formats:

```
IMPORT,CONFIG,MIGRATE,FILENAME= DSN:pds(member) | DDN:ddname(member) |  
member
```

```
IMPORT,CONFIG,MIGRATE,ALL,FILENAME= DSN:pds(member) | DDN:ddname(member)
```

| *member*

Converts the master configuration member, and all query processor override members that it references, to the new configuration. Include query processor override members if applicable.

2. Import only the master configuration with the following command.

```
IMPORT,CONFIG,MIGRATE,MASTERCONFIG,FILENAME= DSN:pds(member) |
```

```
DDN:ddname(member) | member
```

Migrates only the master configuration. Query processor definitions that refer to query processor override members are defined only with master configuration values.

3. Import query processor override members.

```
IMPORT,CONFIG,MIGRATE,OVERRIDE MEMBER= DSN:pds(member) |
```

```
DDN:ddname(member) | member,SERVICE=service name | ALL
```

Migrates a query processor override member. You can migrate the override member for a specific query processor or for all query processors that are defined in the current configuration.

4. Import user configurations.

```
IMPORT,CONFIG,MIGRATE,USER = DSN:pds(member) | DDN:ddname(member),  
member, SERVICE=service name | ALL
```

Migrates a specific user configuration. For this command, query processors must already be added or migrated to a new configuration file. The user configuration can either be applied to a particular query processor task or to all query processor tasks in the new configuration.

5. Verify the configuration migration. You can run the REPORT command at any time to verify the configuration. The following figure shows sample report output.

```
REPORT Reports all services and user configurations stored in the binary  
configuration data sets.
```

Parameters

ddname

The DD name points to the configuration file from the previous version.

member

The name of the PDS member.

pds

The name of the configuration PDS from the previous version.

service name

The name of the query processor service for the related user override. The maximum length allowed for the name is 64 bytes.

Examples

- Migrate user configuration USER1 from DSN CAC.CONFIG to query processor TEST95.

```
IMPORT,CONFIG,MIGRATE,USER=DSN:CAC.CONFIG(USER1),  
SERVICE=TEST95
```

- Migrate USER1 of DDN VSHCONF to all query processors that exist in the current configuration

```
IMPORT,CONFIG,MIGRATE,USER=USER1, SERVICE=ALL
```

The ALL keyword generally causes a query processor override member or a user configuration to be migrated to all query processors defined in the current configuration.

- The following example shows sample REPORT command output:

```

CAC00200I REPORT
CONFIGURATION
SERVICE NAME: GLOBAL
SERVICE CLASS: GLOB
    MESSAGEPOOLSIZE {DEFAULT}=67108864
    DATACONVERRACT {DEFAULT}=0
    FETCHBUFSIZE {DEFAULT}=32000
    DECODEBUFSIZE {DEFAULT}=8192
    STATICCATALOGS {DEFAULT}=0
        NLCAT {DEFAULT}=DD:MSGCAT
    TASKPARM {DEFAULT}=
CONFIGURATION
SERVICE NAME: CNTL
SERVICE CLASS: CNTL
TASK NAME: CACCNTL
    INITIALTHREADS {DEFAULT}=1
    MAXTHREADS {DEFAULT}=1
    MAXUSERS {DEFAULT}=100
    TRACELEVEL {DEFAULT}=4
    RESPONSETIMEOUT {DEFAULT}=5M
    IDLETIMEOUT {DEFAULT}=5M
    SEQUENCE {CHANGED}=1
CONFIGURATION
SERVICE NAME: LOG
SERVICE CLASS: LOG
TASK NAME: CACLOG
    INITIALTHREADS {DEFAULT}=1
    MAXTHREADS {DEFAULT}=1
    MAXUSERS {DEFAULT}=100
    TRACELEVEL {DEFAULT}=1
    RESPONSETIMEOUT {DEFAULT}=5M
    IDLETIMEOUT {DEFAULT}=5M
    SEQUENCE {CHANGED}=2
    LOGBUFSIZE {DEFAULT}=65536
    DISPLAYLOG {DEFAULT}=FALSE
    STREAMNAME {DEFAULT}=
    LOGURL {DEFAULT}=
CONFIGURATION
SERVICE NAME: OPER
SERVICE CLASS: OPER
TASK NAME: CACOPER
    INITIALTHREADS {DEFAULT}=1
    MAXTHREADS {DEFAULT}=1
    MAXUSERS {DEFAULT}=100
    TRACELEVEL {DEFAULT}=4
    RESPONSETIMEOUT {DEFAULT}=5M
    IDLETIMEOUT {DEFAULT}=5M
    SAFEXIT {DEFAULT}=
    SMFEXIT {DEFAULT}=
    SEQUENCE {CHANGED}=3
    SQLSECURITY {DEFAULT}=FALSE
CONFIGURATION
SERVICE NAME: CACTWA
SERVICE CLASS: QP
TASK NAME: CACQP
    INITIALTHREADS {DEFAULT}=5
    MAXTHREADS {DEFAULT}=10
    MAXUSERS {DEFAULT}=20
    TRACELEVEL {DEFAULT}=4
    RESPONSETIMEOUT {DEFAULT}=5M
    IDLETIMEOUT {DEFAULT}=5M

```



```

        BTREEBUFFS {DEFAULT}=4
        USERSUBPOOLMAX {DEFAULT}=8192
        TEMPFILESPEC {DEFAULT}=HIPERSPACE,INIT=8M,MAX=2048M,EXTEND=8M
        VSAMAMPARMS {DEFAULT}=
        SAFEXIT {DEFAULT}=
        SMFEXIT {DEFAULT}=
        SEQUENCE {CHANGED}=4
        JOINTABLES {DEFAULT}=4
        MAXROWSEXCACTION {DEFAULT}=1
        MAXROWSEXAMINED {DEFAULT}=0
        MAXROWSRETURNED {DEFAULT}=0
        STMTRETENTION {DEFAULT}=0
        CPUGOVERNOR {DEFAULT}=
        WLMUOW {DEFAULT}=
CONFIGURATION
SERVICE NAME: INIT
SERVICE CLASS: INIT
TASK NAME: CACINIT
        INITIALTHREADS {DEFAULT}=1
        MAXTHREADS {DEFAULT}=1
        MAXUSERS {DEFAULT}=100
        TRACELEVEL {DEFAULT}=4
        RESPONSETIMEOUT {DEFAULT}=5M
        IDLETIMEOUT {DEFAULT}=5M
        SEQUENCE {CHANGED}=5
        COMMSTRING {CHANGED}=TCP/1.1.1.1/9087

```

Viewing log messages with the log print utility (CACPRTLGL)

With the log print utility (CACPRTLGL), you can format and display messages that are written to a log. You can summarize the log messages or filter them. You can also format and print event messages.

About this task

Perform the steps in the following procedure to view log messages:

Procedure

1. Configure CACPRTLGL. See “Parameters for configuring the log print utility (CACPRTLGL).”
2. Create filters for the output. See “Filters for modifying output from the log print utility (CACPRTLGL)” on page 409.
3. Run CACPRTLGL. There are two ways to run this utility:
 - Run CACPRTLGL as a step in the same job used to run the Classic data server.
 - Run CACPRTLGL as a separate job from the Classic data server job or started task.

Parameters for configuring the log print utility (CACPRTLGL)

You supply values to the **PARM** parameter of the CACPRTLGL EXEC statement to determine which information CACPRTLGL displays and where CACPRTLGL extracts the information from.

Specify the **PARM** parameter in the JCL for the Classic data server. See the sample JCL for CACPRTLGL in the sample members for the Classic data servers found in *USERHLQ.SCACSAMP*, such as CACDS.

See also the sample JCL for CACPRTLGL in the sample member *USERHLQ.SCACSAMP(CACPRTLGL)*, which shows how to run CACPRTLGL against a system logger stream and print a log stream separately.

Recommendation: Use log streams for the diagnostic log or the event log (CACLOG) so that you can print the log while the data server is running. By using log streams, you do not need to set the logger service parameter `DISPLAYLOG=TRUE` to see logged information and can avoid the processing overhead costs related to formatting and displaying the logs.

The following list shows the possible values for the PARM parameter.

SUMMARY=N|Y

- N Displays all of the messages that are in the log if you configured the logger service to write to the CACLOG DD statement or system log streams.
- Y Displays a report about the contents of the log if you configured the logger service to write to the CACLOG DD statement or system log streams.

STREAM=log_stream

The *log_stream* value must be a valid log stream that contains data that was written by the logger service. If you use the STREAM keyword, remove the CACLOG DD statement from the JCL for the log print utility.

PURGE

Marks for deletion all of the log messages that are in the log stream and that are older than the value of the STARTTIME filter criterion for the log print utility.

PURGEALL

Marks for deletion all of the log messages that are in the log stream.

EVENTS=eventlog_stream_name

The name of the event log stream that was specified for the EVENTLOG configuration parameter for the logger service.

LOCALE=locale

The message locale to use when translating the event messages. If you do not specify the LOCALE parameter, the default value EN_US is used to translate event messages using the US English message catalog. Valid values:

EN_US

US English message catalog.

JA_JP Japanese message catalog.

KO_KR

Korean message catalog.

ZH_CH

Traditional Chinese message catalog.

ZH_TW

Simplified Chinese message catalog.

Filters for modifying output from the log print utility (CACPRTLG)

You can use SYSIN control cards to filter and display only a subset of the log messages. With these control cards, you can display messages for a specific time-frame, a specific task, a range of return codes, or any combination of the elements that are listed in the log summary report.

The format of the SYSIN filtering is exactly the same as the format of the summary report. So, you can run a summary report, find the criteria that would be relevant

for you to filter on, then submit a SYSIN control card with those criteria. You can find sample JCL to run a summary report in member CACPRTLS in the SCACSAMP data set.

The following list presents the available filtering criteria. Although the criteria are presented in uppercase, you can specify them in mixed case because the log print utility will fold the characters into uppercase. All filter criteria must be followed by an equal sign and a value.

STARTTIME='YYYY/MM/DD HH:MM:SS:*t h m i*'

Specifies the beginning of the duration of time that you want log information from. When you request the log information for a particular Classic data server, you might find it helpful to review the JES output for the Classic data server job to obtain the start time.

- *t* is tenths of a second
- *h* is hundredths of a second
- *m* is milliseconds
- *i* is ten-thousandths of a second

STOPTIME='YYYY/MM/DD HH:MM:SS:*t h m i*

Specifies the end of the duration of time that you want log information from.

- *t* is tenths of a second
- *h* is hundredths of a second
- *m* is milliseconds
- *i* is ten-thousandths of a second

MINRC

Specifies a numeric value that represents the lowest return code that you want to be reported.

FILTERS

Specifies tracing filters to use in the report. Use only in conjunction with IBM support.

EXFILTERS

Specifies tracing filters not to use in the report. Use only in conjunction with IBM support.

MAXRC

Specifies a numeric value that represents the highest return code that you want to be reported.

TASKS

Specifies a task number (service) to filter the log information by. Although this criterion is helpful if you are diagnosing a problem with a specific task, generally you should not use this criterion. If this criteria is used with multiple values, each separate line must start with the TASKS keyword, an equal sign, and the comma-delimited list of task numbers enclosed within parenthesis.

NODES

Specifies a specific node (address space or Classic data server) for which the log print utility should return information. This value is a comma-delimited list enclosed with parentheses. Each line of node filters must be preceded by the NODES keyword and an equal sign.

SPCRC

Specifies a list of specific return code values for which the log print utility should return log records. Use only in conjunction with IBM support.

SQL reference

The SQL reference information includes general information about SQL language elements, security, views, and SQL information about specific data sources.

General information

You can use the following language elements and syntax diagrams for any of the supported database management systems.

Language elements

The SQL language elements are characters, tokens, SQL statement format, identifiers, naming conventions, authorization IDs, authorization names, qualification of unqualified object names, data types, and constants.

The following topics apply only to the DDL SQL statements for dynamic catalog update operations and to DML SQL statements. See the DB2 SQL reference documentation for the full set of language elements.

Characters:

The basic symbols of SQL are characters from the EBCDIC syntactic character set.

These characters are classified as letters, digits, or special characters:

letter Any one of the uppercase alphabetic characters A through Z plus the three EBCDIC code points reserved as alphabetic extenders for national languages (the code points X'5B', X'7B', and X'7C', which display as \$, #, and @ in code pages 37 and 500).

digit Any one of the characters 0 through 9.

special character

Any character other than a letter or a digit.

SQL statements can also contain double-byte character set (DBCS) characters. You can use double-byte characters in SQL ordinary identifiers and graphic string constants when you enclose the necessary shift characters. You can also use double-byte characters in string constants and delimited identifiers.

In SQL applications you must contain double-byte characters within a single line. Therefore, you cannot continue a graphic string constant from one line to the next. You can continue a character string constant and a delimited identifier from one line to the next only if the break occurs between single-byte characters.

Tokens:

The basic syntactical units of the language are called tokens. A token consists of one or more characters, not including spaces, control characters, or characters within a string constant or delimited identifier.

Tokens are classified as ordinary or delimiter tokens:

Ordinary token

A numeric constant, an ordinary identifier, a host identifier, or a keyword.

```

1
.1
+2
SELECT
E
3

```

Figure 19. Examples of ordinary tokens

Delimiter token

A string constant, a delimited identifier, an operator symbol, or any of the special characters shown in the syntax diagrams. A question mark (?) is also a delimiter token when it serves as a parameter marker.

```

,
'string'
"fld1"
=
.

```

Figure 20. Examples of delimiter tokens

String constants and certain delimited identifiers are the only tokens that can include a space or control character. Any token can be followed by a space or control character. Every ordinary token must be followed by a delimiter token, a space, or a control character. If the syntax does not allow a delimiter token, a space or a control character must follow the ordinary token.

Spaces

A sequence of one or more blank characters. A space is represented as the value x'40'.

Control characters

A special character for string alignment. A control character is treated like a space and does not cause a particular action to occur. The following table identifies the control characters that are supported.

Table 75. Control characters

Name of control character	Hexadecimal value
Tab	05
Form feed	0C
Carriage return	0D
New line or next line	15
Line feed (new line)	25

Uppercase and lowercase

Any token can include lowercase letters, but a lowercase letter in an ordinary token is folded to uppercase. Delimiter tokens are never folded to uppercase.

For example, the first statement is equivalent to the second statement, after folding:

```

select * from DSN8710.EMP where lastname = 'Smith';
SELECT * FROM DSN8710.EMP WHERE LASTNAME = 'Smith';

```

SQL statement format:

An SQL statement is a complete instruction to the database manager that is written using Structured Query Language.

An SQL statement has a length attribute that identifies the physical length of the SQL statement in bytes. Either the client application explicitly identifies the length of the SQL statement, or the ODBC or JDBC driver determines the length of the SQL statement.

Typically, the SQL statement ends with a null byte, which has a hexadecimal value of zeros. In these cases, the length of the SQL statement is determined by scanning the SQL statement and counting the number of bytes up to the null byte.

SQL identifiers:

An identifier is a token that forms a name. An identifier in an SQL statement is an SQL identifier, a parameter marker, or a native identifier. SQL identifiers can be ordinary identifiers or delimited identifiers. They can also be short identifiers, medium identifiers, or long identifiers.

An SQL identifier can be in one of these categories: short ordinary, medium ordinary, long ordinary, short delimited, medium ordinary, or long delimited.

Ordinary identifiers

An ordinary identifier is a letter that is followed by zero or more characters, each of which is a letter, a digit, or the underscore character. An ordinary identifier with an EBCDIC encoding scheme can include Katakana characters.

Double byte character set (DBCS) characters are allowed in SQL ordinary identifiers. You can specify an SQL ordinary identifier, when it is the name of a table, column, view, or stored procedure, by using either DBCS characters or single-byte character set (SBCS) characters. However, an SQL ordinary identifier cannot contain a mixture of SBCS and DBCS characters.

The following rules show how to form DBCS SQL ordinary identifiers. These rules are EBCDIC rules because all SQL statements are in EBCDIC.

- The identifier must start with a shift-out (X'0E') and end with a shift-in (X'0F'). An odd-numbered byte between those shifts must not be a shift-out.
- The maximum length is 8, 18, or 30 bytes including the shift-out and the shift-in depending upon the context of the identifier. In other words, there is a maximum of 28 bytes (14 double-byte characters) between the shift-out and the shift-in.
- There must be an even number of bytes between the shift-out and the shift-in. DBCS blanks (X'4040') are not acceptable between the shift-out and the shift-in.
- The identifiers are not folded to uppercase or changed in any other way.
- Continuation to the next line is not allowed.

An ordinary identifier must not be identical to a keyword that is a reserved word in any context in which the identifier is used.

The following example is an ordinary identifier:

```
SALARY
```

Delimited identifiers

A delimited identifier is a sequence of one or more characters that are enclosed within escape characters. The escape character is the quotation mark (").

You can use a delimited identifier when the sequence of characters does not qualify as an ordinary identifier. Such a sequence, for example, can be an SQL reserved word, or it can begin with a digit. Two consecutive escape characters represent one escape character within the delimited identifier. A delimited identifier that contains double-byte characters also must contain the necessary shift characters.

When the escape character is the quotation mark, the following example is a delimited identifier:

```
"VIEW"
```

Short, medium, and long identifiers

SQL identifiers are also classified according to their maximum length. A long identifier has a maximum length of 30 bytes. A medium identifier has a maximum length of 18 bytes. A short identifier has a maximum length of 8 bytes. These limits do not include the escape characters of a delimited identifier.

Whether an identifier is long, medium, or short depends on what it represents.

Parameter marker

Parameter markers represent values that are supplied for the SQL statement when it is run. The question mark (?) identifies a parameter marker in an SQL statement.

Native identifiers

Native identifiers exist only in CREATE TABLE and CREATE INDEX statements and refer to a native database object. For example, objects can be an MVS data set name, an IMS segment name, a CA-IDMS record name, and so on. Native identifiers can be ordinary or delimited identifiers. If the native identifier represents a reserved word, then you must supply a delimited identifier.

The length and allowable characters in a native identifier are DBMS-specific.

Reserved words:

A number of words cannot be used as ordinary identifiers in some contexts because these words might be interpreted as SQL keywords.

For example, ALL cannot be a column name in a SELECT statement. Each word, however, can be a delimited identifier in contexts where the word otherwise cannot be an ordinary identifier. For example, the quotation mark (") is the escape character that begins and ends delimited identifiers. "ALL" can appear as a column name in a SELECT statement.

You must use the following words as delimited identifiers where referring to an SQL identifier or native identifier in any SQL statement:

```
ADD  
ALL  
ALTER  
AND  
ANY
```

AS
AUDIT
BETWEEN
BIND
BUFFERPOOL
BY
CALL
CAPTURE
CHAR
CHARACTER
CHECK
CLUSTER
COLLECTION
COLUMN
CONCAT
CONSTRAINT
COUNT
CURRENT
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURSOR
DATABASE
DAY
DAYS
DEFAULT
DELETE
DESCRIPTOR
DISTINCT
DOUBLE
DROP
EDITPROC
ERASE
ESCAPE
EXCEPT
EXECUTE
EXISTS
FIELDPROC
FOR
FROM
FULL
GO
GOTO
GRANT
GROUP
HAVING
HOUR
HOURS
IMMEDIATE
IN
INDEX
INNER
INOUT
INSERT
INTO
IS
JOIN
KEY
LEFT
LIKE
LOCKMAX
LOCKSIZE
MICROSECOND
MICROSECONDS
MINUTE
MINUTES
MONTH

MONTHS
NOT
NULL
NUMPARTS
OBID
OF
ON
OPTIMIZE
OR
ORDER
OUT
OUTER
PACKAGE
PART
PLAN
PRECISION
PRIQTY
PRIVILEGES
PROGRAM
REFERENCES
RIGHT
SECOND
SECONDS
SECQTY
SELECT
SET
SOME
STOGROUP
SUBPAGES
SYNONYM
TABLE
TABLESPACE
TO
UNION
UNIQUE
UPDATE
USER
USING
VALIDPROC
VALUES
VCAT
VIEW
VOLUMES
WHERE
WITH
YEAR
YEARS

Naming conventions:

The rules for forming a name depend on the object type that is designated by the name. The syntax diagrams use different terms for different types of names.

The following list defines terms that represent common SQL objects that are referenced in the various DDL statements.

authorization-name

A short identifier that designates a set of privileges. It can also designate a user or group of users.

column-name

A qualified or unqualified name that designates a column of a table or a view.

A qualified column name is a qualifier followed by a period and a long identifier. The qualifier is a table name or a view name.

An unqualified column name is a long identifier.

index-name

A qualified or unqualified name that designates an index.

A qualified index name is a short identifier followed by a period and a medium identifier. The short identifier is the authorization ID that owns the index.

An unqualified index name is a medium identifier with an implicit qualifier. The implicit qualifier is an authorization ID.

procedure-name

A qualified or unqualified name that designates a stored procedure.

A fully qualified procedure name is a two-part name. The first part is the authorization ID that designates the owner of the procedure. The second part is a medium identifier. A period must separate each of the parts.

A one-part or unqualified procedure name is a medium identifier with an implicit qualifier. The implicit qualifier is an authorization ID.

schema-name

An SQL identifier that designates a schema. A schema name that is used as a qualifier of that object name is often also an authorization ID. The objects that are qualified with a schema name are stored procedures and tables.

table-name

A qualified or unqualified name that designates a table.

A fully qualified table name is a two-part name. The first part is the authorization ID that designates the owner of the table. The second part is a medium identifier. A period must separate each of the parts.

A one-part or unqualified table name is a medium identifier with an implicit qualifier. The implicit qualifier is an authorization ID.

view-name

A qualified or unqualified name that designates a view.

A fully qualified view name is a two-part name. The first part is the authorization ID that designates the owner of the view. The second part is a medium identifier. A period must separate each of the parts.

A one-part or unqualified view name is a medium identifier with an implicit qualifier. The implicit qualifier is an authorization ID.

Authorization IDs and authorization names:

An authorization ID is a character string that designates a defined set of privileges. Client connections can successfully run SQL statements only if client connections have the authority to perform the specified functions. A client connection derives this authority from its authorization IDs. An authorization ID can also designate a user or a group of users, but federation does not control this property.

Authorization IDs provide this functionality:

- Authorization checking of SQL statements
- Implicit qualifiers for database objects like tables, views, and indexes

Connections and authorization IDs

Whenever a connection is established, an authorization ID is optionally passed to the server. If an authorization ID is not supplied, the connection is assigned the default authorization ID of PUBLIC. If external security is active by using the SAF EXIT configuration parameter, then as part of connection authentication, the authorization ID is passed to the SAF exit. If the authorization ID is allowed to access the system, the SAF exit can return a secondary authorization ID that is in the list of authorization IDs that are associated with the connection.

Every connection has exactly one primary authorization ID. All other IDs are secondary authorization IDs. A connection can have a maximum of three associated authorization IDs: the authorization ID on the connection request, possibly one secondary authorization ID that is returned by the SAF exit, and the generic authorization ID of PUBLIC.

An authorization name in an SQL statement is not the same as an authorization ID of a connection.

Example

For example, assume that SMITH is the user ID that is supplied during the connection, and you run the following statements:

```
CREATE TABLE TDEPT ...;  
GRANT SELECT ON TDEPT TO KEENE;
```

When the GRANT statement is prepared and run, the SQL authorization ID is SMITH. KEENE is an authorization name that is specified in the GRANT statement.

Authorization to run the GRANT statement is checked against SMITH, and SMITH is the implicit qualifier of TDEPT. The authorization rule is that the privilege set is designated by SMITH must include the SELECT privilege with the GRANT option on SMITH.TDEPT. There is no check involving KEENE.

If SMITH is the implicit qualifier for a statement that contains NAME1, then NAME1 identifies the same object as SMITH.NAME1. If the implicit qualifier is other than SMITH, then NAME1 and SMITH.NAME1 identify different objects.

Privileges

For statements other than an ALTER, CREATE, DROP, GRANT, or REVOKE statement, each privilege that is required for the statement can be a privilege that is designated by any authorization ID of the connection. Therefore, a privilege set is the union of the set of privileges that are held by each authorization ID.

If the SQL statement is an ALTER, CREATE, DROP, GRANT, or REVOKE statement, the only authorization ID for authorization checking is the SQL authorization ID. Therefore, the privilege set is the privileges that are held by the single authorization ID that corresponds to the user ID that is supplied on the connection request.

Qualification of unqualified object names:

Unqualified object names are implicitly qualified. The rules for qualifying a name differ depending on the type of object that the name identifies.

The implicit qualifier for an unqualified index, table, or view name is the authorization ID in the CURRENT SQLID special register. This authorization ID corresponds to the user ID of the connected user that runs the SQL statement.

Data types:

The smallest unit of data that can be manipulated in SQL is called a value. How values are interpreted depends on the data type of their source.

The sources of values are as follows:

- Columns
- Constants
- Expressions
- Host variables
- Special registers

All data types include the null value. Distinct from all non-null values, the null value is a special value that denotes the absence of a (non-null) value. Although all data types include the null value, some sources of values cannot provide the null value. For example, constants, columns that are defined as NOT NULL, and special registers cannot contain null values. The COUNT function cannot return a null value column as the result of a query.

Binary strings:

A binary string is a sequence of bytes. The length of a binary string is the number of bytes in the sequence.

Binary strings are not associated with any CCSID. The supported binary string data types are BINARY and VARBINARY (BINARY VARYING).

The following subtypes of binary strings are supported:

Fixed-length binary strings

The type of fixed-length binary strings is BINARY. When fixed-length binary string distinct types, columns, and variables are defined, the length attribute is specified, and all values have the same length. For a fixed-length binary string, the length attribute must be between 1 and 32704 inclusive.

Varying-length binary strings

All values of varying-length string columns have the same maximum length, which is determined by the length attribute.

The varying-length binary string is VARBINARY (BINARY VARYING).

When varying-length binary strings, distinct types, columns, and variables are defined, the maximum length is specified and this length becomes the length attribute. Actual length values might have a smaller value than the length attribute value. For varying-length binary strings, the actual length specifies the number of bytes in the string.

For a VARBINARY string, the length attribute must be between 1 and 32704. Like a varying-length character string, varying-length binary string could be an empty string.

A binary string column is useful for storing non-character data, such as encoded or compressed data, pictures, voice, and mixed media. Another

use is to hold structured data for exploitation by distinct types, user-defined functions, and stored procedures.

Character string:

A character string is a sequence of bytes. The length of the string is the number of bytes in the sequence. If the length is zero, the value is an empty string. An empty string is not the same thing as a null value.

The bytes of a character string can represent a mixture of characters from a single-byte character set (SBCS) and a double-byte character set (DBCS). Strings that might contain both SBCS and DBCS characters are called mixed data. EBCDIC mixed data might contain shift characters, which do not represent SBCS or DBCS data.

The following subtypes of character strings are supported:

Fixed-length character strings

All the values of a column with a fixed-length character-string data type have the same length, which is determined by the length attribute of the column. The length attribute must be between 1 and 255. Every fixed-length string column is a short string column. A fixed-length character-string column can also be called a CHAR or CHARACTER column.

Varying-length character strings

All values of varying-length string columns have the same maximum length, which is determined by the length attribute. If the length attribute is greater than 254, the column is a long-string column. Long-string columns cannot be referenced in these items:

- Function other than SUBSTR or LENGTH
- GROUP BY clause
- ORDER BY clause
- CREATE INDEX statement
- SELECT DISTINCT statement
- Subselect of a UNION without the ALL keyword
- Predicate other than LIKE

The VARCHAR column identifies a short, varying-length string column. A maximum-length attribute must be specified, and must be between 1 and 32704. The VARCHAR(*x*) syntax is preferred over LONG VARCHAR, which is a long-string column that does not have an explicit length attribute. The maximum length is also 32704. However, a smaller maximum length is explicitly specified or internally computed, based on the maximum physical size limits.

Graphic strings:

A graphic string is a sequence of DBCS characters. The length of the string is the number of characters in the sequence. Each character is assumed to be two-bytes long. Like character strings, graphic strings can be empty. An empty string is not the same thing as a null value.

The following subtypes of character strings are supported:

Fixed-length graphic strings

All the values of a column with a fixed-length graphic-string data type have the same length, which is determined by the length attribute of the column. The length attribute must be between 1 and 127. Every fixed-length graphic-string column is a short string column. A fixed-length, graphic-string column can also be called a GRAPHIC column.

Varying-length graphic strings

All values of varying-length string columns have the same maximum length, which is determined by the length attribute. If the length attribute is greater than 127, the column is a long-string column, and the same rules for LONG VARCHAR columns apply.

The VARGRAPHIC column identifies a short, varying-length graphic-string column, and a maximum-length attribute must be specified. The length attribute must be between 1 and 127. A long-graphic-string column is identified as a LONG VARGRAPHIC column and does not have an explicit length attribute. The maximum length is 16352; however, a smaller maximum length is explicitly specified or internally computed, based on the maximum physical-size limits.

Numbers:

The numeric data types are binary integer, floating-point, and decimal. Binary integer includes small integer and large integer. Floating-point includes single precision and double precision. Binary numbers are exact representations of integers; decimal numbers are exact representations of real numbers; and floating-point numbers are approximations of real numbers.

All numbers have a sign and a precision. When the value of a column or the result of an expression is a decimal or floating-point zero, its sign is positive. The precision of binary integers and decimal numbers is the total number of binary or decimal digits excluding the sign. The precision of floating-point numbers is either single or double, based on the number of hexadecimal digits in the fraction.

The types of numbers are as follows:

Small integer (SMALLINT)

A small integer is a binary integer with a precision of 15 bits. The range of small integers is -32768 to +32767.

Large integer (INTEGER)

A large integer is a binary integer with a precision of 31 bits. The range of large integers is -2147483648 to +2147483647.

Single precision floating-point (REAL or FLOAT)

A single-precision, floating-point number is a 32-bit approximation of a real number. The number can be zero or can range from -3.40282347E+38 to -1.17549435E-38, or from 1.17549435E-38 to 3.40282347E+38.

When the precision of a FLOAT is in the range of 1 to 21, the query processor treats the column as REAL.

The query processor uses standard 370 representation to process single precision floating point numbers. Individual databases might treat these numbers in different representations and support different limits. Likewise, when single precision floating point numbers are manipulated by the CLI, JDBC, and ODBC client components, the representation and limits are based on the platform and compiler that is used.

On the z/OS server, the 370 representation consists of a sign bit, a 7-bit biased hexadecimal exponent, and a 24-bit fractional part. The exponent bias is 64. All operations on single precision floating point numbers are normalized. The value that can be represented by a single precision floating point number is approximately 6 or 7 decimal digits of precision.

Double precision floating-point (DOUBLE or FLOAT)

A double precision, floating-point number is a 64-bit approximation of a real number. The number can be zero or can range from -1.797693134862315E+308 to -2.225073858507201E-308, or from 2.225073858507201E-308 to 1.797693134862315E+308.

When the precision of a FLOAT is in the range of 22 to 53, the query processor treats the column as DOUBLE PRECISION.

Like single precision numbers, double precision numbers that are manipulated by the z/OS server components use standard 370 representation with the caveat that the source database and client implementation might be different than that used by the data server.

On the z/OS server, the 370 representation consists of a sign bit, a 7-bit biased hexadecimal exponent, and a 56-bit fractional part. The exponent bias is 64. All operations on double precision floating point numbers are normalized. The value that can be represented by a single precision floating point number is approximately 16 or 17 decimal digits of precision.

Decimal (DECIMAL)

A decimal number is a packed decimal number with an implicit decimal point. The position of the decimal point is determined by the precision and the scale of the number. The scale, which is the number of digits in the fractional part of the number, cannot be negative or greater than the precision. The maximum precision is 31 digits.

All values of a decimal column have the same precision and scale. The range of a decimal variable or the numbers in a decimal column is $-n$ to $+n$, where n is the largest positive number that can be represented with the applicable precision and scale. The maximum range is $1 - 10^{31}$ to $10^{31} - 1$.

String representations of numbers

Values whose data types are small integer, large integer, floating-point, and decimal are stored in an internal form that is transparent to the user of SQL. But string representations of numbers can be used in some contexts. A valid string representation of a number must conform to the rules for numeric constants.

COBOL data types supported by the data server:

When you map user tables in the metadata catalogs on the data server, Classic Data Architect can convert COBOL data types to native types that the server supports and their corresponding SQL types.

Data types in a CREATE TABLE statement

When you map a new table, you can import data definitions in COBOL copybooks that describe the layout of the source database and the types associated with the data items. The information in the following table shows how Classic Data Architect converts COBOL data types to native and SQL types.

When you create a new user table in the metadata catalogs on a data server, you run a CREATE TABLE statement. Typically, this statement is found within Data Definition Language (DDL) that Classic Data Architect creates automatically when you run the Generate DDL wizard. The CREATE TABLE statement assigns a native data type to each column by using a DATATYPE keyword and assigns an SQL data type in a USE AS clause.

You can also use the information shown here to override the data types that are assigned to columns in a user table. One approach is to edit the DDL in an SQL Editor window. With VSAM, CICS VSAM, CA-Datcom, or sequential data sources, you can also change the native data type in the Classic Column Info properties window. Data types that you assign manually must be appropriate for the data in the source database.

Supported COBOL data types

The following table describes COBOL data types that the data server supports. The table also provides information about native and SQL types that correspond with each COBOL type.

COBOL element type	COBOL usage	Sign	Native data type	Length	SQL data type	Example
COBOLAlphabeticType COBOLAlphaNumericType			C	<= 254	CHAR scale = 0	PIC X(4)
COBOLNumericType	Packed decimal	T	P		DECIMAL	PIC S9(3) COMP-3
COBOLNumericType	Packed decimal	F	UP		DECIMAL	PIC 9(3) COMP-3
COBOLNumericType	Display	T	C		DECIMAL	PIC S99
COBOLNumericType	Display	F	UC		CHAR scale = 0 DECIMAL scale > 0	PIC 99
COBOLNumericType	Binary	T	H		SMALLINT	PIC S99 COMP-4
COBOLNumericType	Binary	F	UH		SMALLINT	PIC 99 COMP-4
COBOLNumericType	Binary	T	F		INTEGER	PIC S9(8) COMP-4
COBOLNumericType	Binary	F	UF		INTEGER	PIC 9(8) COMP-4
COBOLNumericType	Binary		D		DECIMAL	PIC 9(18) COMP-4
COBOLInternalFloatType	Float		F		REAL	COMP-1
COBOLInternalFloatType	Double		D		DOUBLE	COMP-2
COBOLDBCSType	DBCS		C	<= 127	GRAPHIC	PIC G (or PIC N if NSymbol parser setting to DBCS)

COBOL element type	COBOL usage	Sign	Native data type	Length	SQL data type	Example
COBOLAlphaNumericEditedType COBOLNumericEditedType COBOLExternalFloatType			C	<= 254	CHAR	PIC clause is a display format that might contain B,P,Z,,,+,-,CR,DB,E

Native data types

The following table describes Classic data types and their equivalent SQL types. A CREATE TABLE statement specifies the native data type by using the DATATYPE parameter.

The Other SQL data types column lists alternative SQL types that you can specify as overrides by editing DDL in the SQL Editor window.

Table 76. Native data types

DATATYPE value	Contents	Standard SQL data type	Other SQL data types
C	Mixed mode character data. When the SQL data type is DECIMAL, the data is assumed to consist wholly of numbers with the right most number identifying the sign.	CHAR DECIMAL	DECIMAL, VARCHAR, GRAPHIC, or VARGRAPHIC, BINARY, VARBINARY
P	Packed decimal data where the sign is stored in the far right aggregation of four bits.	DECIMAL	BINARY
D ¹	Floating point data. The columns length or precision determines whether the SQL data type is DOUBLE PRECISION or DECIMAL. 64-bit data is mapped as DECIMAL.	DOUBLE PRECISION	FLOAT(<i>precision</i>) DECIMAL, BINARY
F ²	32-bit signed binary value where the sign is in the high order bit.	DECIMAL INTEGER REAL	FLOAT(<i>precision</i>) BINARY
H	16-bit signed binary value where the sign is in the high order bit.	DECIMAL SMALLINT	BINARY

Table 76. Native data types (continued)

DATATYPE value	Contents	Standard SQL data type	Other SQL data types
V	Variable mixed-mode character data, where the actual data is preceded by a 16-bit signed binary number that identifies the actual length of the data.	VARCHAR	LONG VARCHAR, VARGRAPHIC, or LONG VARGRAPHIC, VARBINARY
UC	Unsigned zoned-decimal data where the last character does not identify the sign. The value is always a positive value.	DECIMAL	CHAR, BINARY, VARBINARY
UP	Packed decimal data where the sign bit is ignored. The value is always positive.	DECIMAL	BINARY
UF	Unsigned 32-bit binary value.	INTEGER	BINARY
UH	Unsigned 16-bit binary value.	SMALLINT	BINARY
B ³	Fixed length binary data.	BINARY	DECIMAL, INTEGER, SMALLINT, VARBINARY
VB ³	Variable length binary data, where the actual data is preceded by a 16-bit signed binary number that identifies the actual length of the data.	VARBINARY	N/A

¹The SQL data type is DOUBLE PRECISION or FLOAT. DOUBLE PRECISION is shorthand for an 8-byte floating point number. In the USE AS clause, you can identify this data type as DOUBLE PRECISION or FLOAT(precision). If the precision value is in the range 22–53, the column represents an 8-byte floating point number. If you assign FLOAT, specify the maximum precision. The column length is based on the precision.

²The SQL data type is REAL or FLOAT. REAL is shorthand for a 4-byte floating point number. In the USE AS clause, you can identify this data type as REAL or FLOAT(precision). If the precision value is in the range 1–21, the column represents a 4-byte floating point number. If you assign FLOAT, specify the maximum precision. The column length is based on the precision.

³ Although Cobol does not support native binary types, you can manually edit the DDL to use binary data types.

PL/I data types supported by the data server:

When you map user tables in the metadata catalogs on the data server, Classic Data Architect can convert PL/I data types to native types that the server supports and their corresponding SQL types.

Data types in a CREATE TABLE statement

When you map a new table, you can import data definitions in PL/I include files that describe the layout of the source database and the types associated with the data items. The information in the following tables shows how the data server converts PL/I data types to native and SQL types.

When you create a new user table in the metadata catalogs on a data server, you run a CREATE TABLE statement. Typically, this statement is found in the Data Definition Language (DDL) that Classic Data Architect creates automatically when you run the Generate DDL wizard. The CREATE TABLE statement assigns a native data type to each column using a DATATYPE keyword and assigns an SQL data type in a USE AS clause.

You can also use the information shown here to modify the data types that are assigned to columns in a user table. One approach is to edit the DDL in an SQL Editor window. With VSAM, CICS VSAM, CA-Datcom, or sequential data sources, you can also change the native data type in the Classic Column Info properties window. Data types that you assign manually must be appropriate for the data in the source database.

Supported PL/I data types

The following table describes PL/I data types that the data server supports. The table also provides information about native and SQL types that correspond with PL/I.

Computational data type	Precision and range	Scale	Native data type	SQL data type
CHAR(n)	n <= 254		C	CHAR(n)
CHAR(n) VARYING	n <= 32704		V	VARCHAR(n)
GRAPHIC(n)	n <= 127		C	GRAPHIC(n)
GRAPHIC(n) VARYING	n <= 16352		V	VARGRAPHIC(n)
PICTURE (non-numeric)	size <= 254		C	CHAR(n)
PICTURE (V,9) ¹			UC	DECIMAL(p,q)
PICTURE (all other numeric formats) ¹	size <= 254		C	CHAR(n)
BIN FIXED(p,q) UNSIGNED	9 <= p <= 16	q=0	UH	SMALLINT
BIN FIXED(p,q) UNSIGNED	17 <= p <= 32	q=0	UF	INTEGER
BIN FIXED(p,q) UNSIGNED	33 <= p <= 64	q=0	D	DECIMAL(21,0)
BIN FIXED(p,q) SIGNED ²	8 <= p <= 15	q=0	H	SMALLINT

Computational data type	Precision and range	Scale	Native data type	SQL data type
BIN FIXED(p,q) SIGNED ²	16 <= p <= 32	q=0	F	INTEGER
BIN FIXED(p,q) SIGNED ²	32 <= p <= 63	q=0	D	DECIMAL(19,0)
DECIMAL FIXED(p,q)		q>=0	P	DECIMAL(p,q)
DECIMAL FLOAT(p)	1 <= p <= 6		F	REAL
DECIMAL FLOAT(p)	7 <= p <= 16		D	DOUBLE
BIN FLOAT(p)	1 <= p <= 21		F	REAL
BIN FLOAT(p)	22 <= p <= 53		D	DOUBLE
¹ PL/I supports many different types of arithmetic character data, such as 9, V, S, +, -, Z, /, R, CR, DB, T, R, K, and F. Values that contain character strings are typically used for report formatting purposes, and the data server maps them as character data. With PL/I picture strings, S, +, and - denote a character, not a sign bit. The data server does not support them as DECIMAL. ² The SIGNED attribute is the default.				

Native data types

The following table describes the contents of native data types and their equivalent SQL types. A CREATE TABLE statement specifies the native data type by using the DATATYPE parameter.

The Other SQL data types column lists alternative SQL data types that you can specify as overrides by editing DDL in the SQL Editor window.

Table 77. Native data types

DATATYPE value	Contents	Standard SQL data type	Other SQL data types
C	Mixed mode character data. When the SQL data type is DECIMAL, the data is assumed to consist wholly of numbers with the right most number identifying the sign.	CHAR DECIMAL	DECIMAL, VARCHAR, GRAPHIC, or VARGRAPHIC, BINARY, VARBINARY
P	Packed decimal data where the sign is stored in the far right aggregation of four bits.	DECIMAL	BINARY

Table 77. Native data types (continued)

DATATYPE value	Contents	Standard SQL data type	Other SQL data types
D ¹	Floating point data. The columns length or precision determines whether the SQL data type is DOUBLE PRECISION or DECIMAL. 64-bit data is mapped as DECIMAL.	DOUBLE PRECISION	FLOAT(<i>precision</i>) DECIMAL, BINARY
F ²	32-bit signed binary value where the sign is in the high order bit.	INTEGER REAL	FLOAT(<i>precision</i>) BINARY
H	16-bit signed binary value where the sign is in the high order bit.	SMALLINT	BINARY
V	Variable mixed-mode character data, where the actual data is preceded by a 16-bit signed binary number that identifies the actual length of the data.	VARCHAR	LONG VARCHAR, VARGRAPHIC, or LONG VARGRAPHIC, VARBINARY
UC	Unsigned zoned-decimal data where the last character does not identify the sign. The value is always a positive value.	DECIMAL	CHAR, BINARY, VARBINARY
UP	Packed decimal data where the sign bit is ignored. The value is always positive.	DECIMAL	BINARY
UF	Unsigned 32-bit binary value.	INTEGER	BINARY
UH	Unsigned 16-bit binary value.	SMALLINT	BINARY
B ³	Fixed length binary data.	BINARY	DECIMAL, INTEGER, SMALLINT, VARBINARY
VB ³	Variable length binary data, where the actual data is preceded by a 16-bit signed binary number that identifies the actual length of the data.	VARBINARY	N/A

Table 77. Native data types (continued)

DATATYPE value	Contents	Standard SQL data type	Other SQL data types
	¹ The SQL data type is DOUBLE PRECISION or FLOAT. DOUBLE PRECISION is shorthand for an 8-byte floating point number. In the USE AS clause, you can identify this data type as DOUBLE PRECISION or FLOAT(precision). If the precision value is in the range 22–53, the column represents an 8-byte floating point number. If you assign FLOAT, specify the maximum precision. The column length is based on the precision.		
	² The SQL data type is REAL or FLOAT. REAL is shorthand for a 4-byte floating point number. In the USE AS clause, you can identify this data type as REAL or FLOAT(precision). If the precision value is in the range 1–21, the column represents a 4-byte floating point number. If you assign FLOAT, specify the maximum precision. The column length is based on the precision.		
	³ Although PL/I does not support native binary types, you can manually edit the DDL to use binary data types.		

Mapping PL/I complex data:

You can support complex data found in PL/I include files by mapping complex numbers as two separate numbers.

About this task

Complex data consists of a real number followed by an imaginary number. Classic Data Architect maps complex data by mapping two fields. One field contains the real number, and the other contains the imaginary number.

Tip: PICTURE data types with a numeric picture clause can have the COMPLEX attribute. If the PICTURE clause contains the supported characters of 9 and V, it will be mapped as two DECIMAL variables. If the PICTURE clause contains non-supported arithmetic characters, it will be mapped as two CHAR variables.

To map complex data found in a PL/I include file:

Procedure

1. When you map a table in the New table wizard, select a PL/I include file containing a complex number.
No Classic or SQL data type can represent complex numbers, such as the one shown in the first example.
2. When the wizard interprets a complex number, it appends the element name with `_REAL` and `_IMAG`, as shown in the second example.

Example

Example: PL/I construct containing a complex number.

```
DCL 1 MYSTRUCT,
      2 COMPLEXNUM BIN FIXED(15) COMPLEX;
```

Example: DDL describing two mapped fields, one containing the real number and the other containing the imaginary number.

```
DCL 1 MYSTRUCT,
      2 COMPLEXNUM_REAL BIN FIXED(15),
      2 COMPLEXNUM_IMAG BIN FIXED(15);
```

Constants:

A constant (also called a literal) specifies a value. Constants are classified as string constants or numeric constants. Numeric constants are further classified as integer, floating-point, or decimal. String constants are classified as character or graphic. All constants have the attribute NOT NULL. A negative sign in a numeric constant with a value of zero is ignored.

The types of constants are as follows:

Integer constants

Specifies a binary integer as a signed or unsigned number that has a maximum of 10 significant digits and no decimal point. If the value is not within the range of a large integer, the constant is interpreted as a decimal constant. The data type of an integer constant is large integer.

In syntax diagrams, the term *integer* is used for an integer constant that must not include a sign.

Floating-point constants

Specifies a floating-point number as two numbers separated by an E. The first number can include a sign and a decimal point. The second number can include a sign but not a decimal point. The value of the constant is the product of the first number and the power of 10 that is specified by the second number. The value must be within the range of floating-point numbers. The number of characters in the constant must not exceed 30. Excluding leading zeros, the number of digits in the first number must not exceed 17, and the number of digits in the second must not exceed 2. The data type of a floating-point constant is double precision floating-point.

15E1 2.E5 -2.2E-1 +5.E+2

Figure 21. Floating-point constants that represent the numbers 150, 200000, -0.22, and 500

Decimal constants

Specifies a decimal number as a signed or unsigned number of no more than 31 digits and either includes a decimal point or is not within the range of binary integers. The precision is the total number of digits, including any to the right of the decimal point. The total includes all leading and trailing zeros. The scale is the number of digits to the right of the decimal point, including trailing zeros.

025.50 1000. -15. +37589333333333333333.33

Figure 22. Decimal constants that have precisions and scales of 5 and 2, 4 and 0, 2 and 0, 23 and 2

Binary string constants

Specifies a varying-length binary string. The binary constant is valid for columns BINARY or VARBINARY data types.

A binary-string constant is formed by specifying a BX followed by a sequence of characters that starts and ends with a string delimiter. The characters between the string delimiters must be an even number of hexadecimal digits. The number of hexadecimal digits must not exceed 254.

A hexadecimal digit is a digit or any of the letters A through F (upper case or lower case). Under the conventions of hexadecimal notation, each pair

of hexadecimal digits represents one byte. Note that this representation is similar to the representation of the character-constant that uses the X" form; however binary-string constant and character-string constant are not compatible and the X" form can not be used to specify a binary-string constant, just as the BX" form cannot be used to specify a character-string constant.

Examples of binary-string constants:

```
BX'0000'   BX'C141C242'   BX'FF00FF01FF'
```

Character string constants

Specifies a varying-length character string. Character string constants have these forms:

A sequence of characters that starts and ends with an apostrophe (').

Specifies the character string that is contained between the string delimiters. The number of bytes between the delimiters must not be greater than 255. Two consecutive string delimiters are used to represent one string delimiter within the character string.

An X followed by a sequence of characters that starts and ends with a string delimiter.

Also called a hexadecimal constant. The characters between the string delimiters must be an even number of hexadecimal digits. The number of hexadecimal digits must not exceed 254. A hexadecimal digit is a digit or any of the letters A through F (uppercase or lowercase). Under the conventions of hexadecimal notation, each pair of hexadecimal digits represents a character. A hexadecimal constant allows you to specify characters that do not have a keyboard representation.

```
'12/14/1985'   '32'   'DON'T CHANGE'   X'FFFF'   ''
```

Figure 23. Examples of character string constants

The last string in the example (") represents an empty character string constant, which is a string of zero length.

A character string constant is classified as mixed data if it includes a DBCS substring. In all other cases, a character string constant is classified as SBCS data.

Graphic string constants

Specifies a varying-length graphic string.

In EBCDIC environments, the forms of graphic string constants are:

- G'0x0e[dbcs-string]0x0f'
- N'0x0e[dbcs-string]0x0f'

In SQL statements, graphic string constants cannot be continued from one line to the next. The maximum number of DBCS characters in a graphic string constant is 124.

General syntax diagrams

The syntax for the DROP and COMMENT ON statements is the same for all the DBMS types.

DROP statement:

The DROP statement deletes an object from the metadata catalog.

Syntax

```
►► DROP INDEX index-name ;  
      TABLE table-name  
      PROCEDURE procedure-name  
      VIEW view-name
```

Parameters

INDEX *index-name*

Deletes an index definition from the metadata catalog.

Following the INDEX keyword is the index name. If qualified, the name is a two-part name, and the authorization ID that qualifies the name is the index owner. If an unqualified name is supplied, the owner name is the authorization ID from the CURRENT SQLID special register.

One of the following permissions is required to run the statement:

- SYSADM
- DBADM for the database type that is in the DBNAME column for the table that is referenced by the index
- Ownership of *index_name* or the owner of table that is referenced by the index.

TABLE *table-name*

Deletes a table from the metadata catalog. The table name can be a qualified or unqualified table name. If an unqualified name is supplied, the table owner is from the CURRENT SQLID special register.

When a table is deleted, all dependent indexes are deleted, in addition to any views that reference the table. All authorization information that is associated with the table is also deleted from the metadata catalog.

One of the following permissions is required to run the DROP TABLE statement:

- SYSADM
- DBADM for the database type that is in the DBNAME column for the table referenced by the index
- Ownership of the table that you are dropping

PROCEDURE *procedure-name*

Deletes a stored procedure definition from the metadata catalog. The procedure name can be a qualified or unqualified name. If an unqualified name is supplied, the stored procedure owner is from the CURRENT SQLID special register.

All authorization information that is associated with the procedure is also deleted from the metadata catalog.

One of the following permissions is required to run the DROP PROCEDURE statement:

- Ownership of the procedure
- SYSADM authority

VIEW *view-name*

Identifies the name of the view to be deleted. *view-name* can be a qualified or unqualified view name. If an unqualified name is supplied, the view owner is obtained from the CURRENT SQLID special register.

When a view is deleted, all authorization information that is associated with the view is also deleted from the metadata catalog. DROP VIEW deletes dependent views along with those specified in the DROP VIEW statement.

One of the following permissions is required to execute the DROP VIEW statement:

- SYSADM
- Ownership of the view being dropped

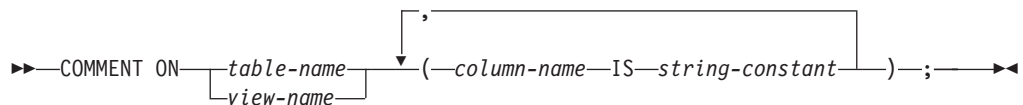
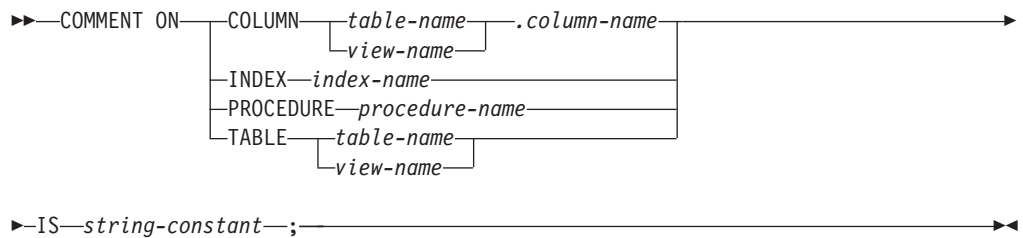
COMMENT ON statement:

The COMMENT statement adds or replaces comments in the descriptions of objects in the metadata catalog.

The COMMENT ON statement updates the REMARKS column in the SYSIBM.SYSTABLES, SYSIBM.SYSCOLUMNS, SYSIBM.SYSINDEXES, or SYSIBM.SYSROUTINES table, depending on the form of the statement.

The COMMENT ON statement has two syntax diagrams. The first syntax diagram updates the REMARKS column in a single metadata catalog table. The second syntax diagram updates the REMARKS column in the SYSIBM.SYSCOLUMNS table for a table or view definition.

Syntax



Parameters

COLUMN

Specifies the column that the comment applies to.

The name must identify a column of a table or view that exists in the system catalog. The column names must not be qualified. The comment is placed into the REMARKS column of the SYSIBM.SYSCOLUMNS system table, for the row that describes the column.

Do not use the keywords TABLE or COLUMN to comment on more than one column in a table or view. Give the table or view name and then, in parentheses, a list in the form:

column-name IS string-constant,
column-name IS string-constant,...

One of the following permissions is required to run the COMMENT ON statement to update a column:

- Ownership of the table or view
- SYSADM authority
- DBADM authority for the database class (only when a table is referenced)

table-name.column-name

Name of the table column that the comment applies to.

view-name.column-name

Name of the view column that the comment applies to.

INDEX *index-name*

Updates the REMARKS column in SYSIBM.SYSINDEXES for an index definition in the metadata catalog.

Following the INDEX keyword is the index name. If qualified, *index-name* is a two-part name, and the authorization ID that qualifies the name is the owner of the index. If an unqualified index name is supplied, the owner name is the authorization ID, which is from the CURRENT SQLID special register.

One of the following permissions is required:

- Ownership of the table or index
- DBADM authority for the database class of the table referenced by the index
- SYSADM authority

PROCEDURE *procedure-name*

Updates the REMARKS column in SYSIBM.SYSROUTINES table for a stored procedure definition.

Following the PROCEDURE keyword is the procedure name. If qualified, the procedure name is a two-part name, and the authorization ID that qualifies the name is the owner of the procedure. If an unqualified procedure name is supplied, the owner name is the authorization ID from the CURRENT SQLID special register.

One of the following permissions is required:

- Ownership of the procedure
- SYSADM authority

TABLE

Updates the REMARKS column in SYSIBM.SYSTABLES for a table or view.

Following the TABLE keyword is a name that refers to either a table or view. If qualified, the table or view name is a two-part name, and the authorization ID that qualifies the name of the owner of the table or view. If an unqualified table name is supplied, the owner name is the authorization ID that is from the CURRENT SQLID special register.

One of the following permissions is required:

- Ownership of the table or view
- SYSADM authority
- DBADM authority for the database class (only when a table is referenced)

table-name

Name of the table that the comment applies to.

view-name

Name of the view that the comment applies to.

IS *string-constant*

Introduces the comment that you want to make.

string-constant can be any SQL character string constant of up to 254 bytes.

IMS

You can use the CREATE TABLE, ALTER TABLE and CREATE INDEX statements to define tables and indexes that reference IMS databases.

CREATE TABLE statement for IMS

You can use the CREATE TABLE statement to define a logical table that references an IMS database.

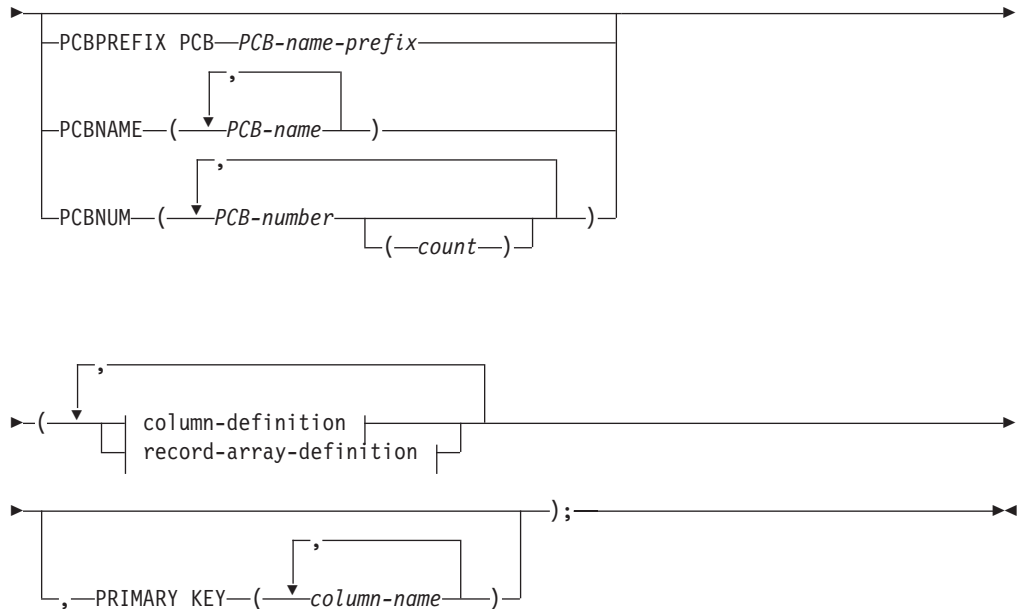
Authorization

The connected user ID must have one of the following privileges to run the CREATE TABLE statement:

- SYSADM
- DBADM for the database type that is referenced in the DBTYPE clause

The owner has all table privileges on the table (such as SELECT, UPDATE, and so on) and the authority to drop the table. The owner can grant equivalent use privileges on the table.

```
▶▶ CREATE TABLE table-name DBTYPE IMS DBD-name
▶ [INDEXROOT perceived-root-segment-name]
▶ leaf-segment-name [SUBSYSTEM IMS-subsystem-ID]
▶ [SCHEDULEPSB (standard-PSB-name [JOIN-PSB-name])]
```



authorization-ID.table-name

Identifies the owner of the table and the name of the table that you want to create.

If you do not provide an authorization ID, the ID in the CURRENT SQLID special register is used.

You must create more than one table if you map to a database or a file that meets either of these criteria:

- The database or file contains repeating data.
- The database or file contains information about one or more distinct sub-objects, because the database or file is not designed to follow the third normalization rules, which are part of the standards to eliminate redundancies and inconsistencies in table data. In a table designed according to third normalization rules, each non-key column is independent of other non-key columns, and is dependent only upon the key.

DBTYPE IMS

Specifies that the CREATE TABLE statement defines a logical table that references an IMS database.

DBD-name

Identifies the IMS DBD (database definition) that the table references. *DBD-name* corresponds to the name in the NAME parameter for the DBD statement. That statement is in the DBDGEN source definition for the IMS logical or physical database that the IMS table references. *DBD-name* follows z/OS load-module naming conventions.

INDEX ROOT *perceived-root-segment-name*

Identifies the root segment for the database hierarchy that the IMS logical table definition maps. By default, the root segment is the physical or logical root segment for the database that *DBD-name* specifies. The physical or logical root segment for the database constitutes the root segment for verification of the IMS hierarchy to the segment that *leaf-segment-name* specifies.

The INDEXROOT clause is required when the logical table references an IMS secondary data structure hierarchy because the intent is to use a secondary index to access or update the IMS database. The INDEXROOT clause is required only when a secondary index is used and the target segment of the secondary index is not the root segment of the database. For additional information about secondary indexes, see the *IMS Administration Guide: Database Manager*.

The name is a short native identifier that follows IMS segment naming conventions. The segment must exist in the DBD that *DBD-name* specifies.

The segment must be either the root segment of the database or the root segment because a secondary index accesses the table and database.

The following caveats apply for the INDEXROOT clause:

- When a secondary index exists in the database that your table references and the target segment in the segment hierarchy is the root segment of the database, create separate tables for each access path.

You can use one table to access the IMS database by using the primary key sequence field. You can use the additional tables to access the database by using the secondary index XDFLD definition. Each of these additional tables must use either a different PSB to access the database, or the PCB prefix option if for a single PSB.

- To use a single mapping for both primary key and secondary index access, you must specify the PCB prefix option and explicitly define indexes by using the CREATE INDEX statement for each access technique (primary key or XDFLD). On each index definition, you identify the PCB prefix that selects the PCB that accesses the IMS database.

In this situation, the PCB that accesses the database depends upon the contents of the WHERE clause:

- If columns in the WHERE clause provide references to all of the columns that make up an XDFLD definition, the PCB prefix for the index that contains the XDFLD columns is used to access the database.
- If the WHERE clause contains references to the columns that map to the primary key sequence field, the index that contains the primary key sequence field columns is selected. The PCB prefix that is associated with that index is used to access the database.
- If the WHERE clause does not contain sufficient information to select either index, the PCB prefix at the table level determines which PCB accesses the database.

leaf-segment-name

Identifies the lowest level segment in the database hierarchy that the table maps to.

The database hierarchy is determined by traversing the parent chain (PARENT= keyword in the DBD definition) for *leaf-segment-name* to either the explicitly identified *perceived-root-segment-name* or the root segment of the physical or logical database.

The name is a short native identifier that follows IMS segment naming conventions. The segment must exist in the DBD that is identified by *DBD-name*.

When *perceived-root-segment-name* is the physical or logical root segment of the database, *leaf-segment-name* must be a physical or logical child of *perceived-root-segment-name*.

When *perceived-root-segment-name* is the root segment in a secondary data structure, *leaf-segment-name* must be a child of *perceived-root-segment-name*.

SUBSYSTEM *IMS-subsystem-ID*

Identifies the IMS subsystem that is the location of the database that *DBD-name* specifies. The Open Database Access (ODBA) interface uses *IMS-subsystem-ID* when it accesses or updates the IMS database for two-phase commit.

The ID is a native identifier that follows IMS subsystem naming conventions. *IMS-subsystem-ID* is 1 to 4 characters in length.

SCHEDULEPSB

Identifies the names of one or two PSBs that access or update the IMS database when the DRA or ODBA interface is used.

standard-PSB-name

Identifies the name of the default PSB that accesses the IMS database.

JOIN-PSB-name

Optionally specifies the name of the PSB that is scheduled to access the IMS database. The DRA and ODBA interfaces use *DBD-name* to identify the IMS database. *JOIN-PSB-name* corresponds to a PSB definition that is defined to the IMS system being accessed and as a PDS member under the same name that exists in the active ACB library of the target IMS subsystem.

If your applications issue joins against multiple IMS tables, specify *JOIN-PSB-name*. *JOIN-PSB-name* follows the naming conventions for the z/OS load module.

The PSB is scheduled when a SELECT statement is run that contains a JOIN predicate that references multiple IMS tables. This first table is the one that JOIN references.

PCBPREFIX **PCB** *PCB-name-prefix*

Identifies a partial PCBNAME that specifies the PCB that accesses or updates the IMS database.

The prefix is a native identifier that follows IMS PCB naming conventions. *PCB-name prefix* is from 1 to 7 characters in length.

PCBNAME (*PCB_name,...*)

Specifies up to 5 PCB names, each of which access an IMS database for a table. Multiple names are required if the same table is referenced more than one time in an SQL statement, or when the same PCB name is associated with more than one table, and the additional tables are referenced in a single SQL statement.

Each PCB name in the list is 1 to 8 characters in length.

PCBNUM (*PCB_number (count),...*)

Allows more potential PCBs to access the IMS database for the table than the PCBNAME keyword allows. Multiple numbers are required if the same table is referenced more than once in an SQL statement, or when the same PSB is associated with more than one table, the PCBs in the PSB have sensitivity to the segments that the table accesses and the same PCB ordinal numbers are specified for these tables. These additional tables are referenced in a single SQL statement.

You can specify up to ten sets of PCB number ranges. The order of the numbers represent which order a PSB is checked to determine whether a PCB accesses the IMS database.

For each item in the list, different methods identify the PCB numbers that are checked. The simplest method is separate each PCB number with commas. The second method identifies a range that consists of a starting PCB number that is followed by parenthesis and a number that identifies the number of PCBs to check from the starting number.

With either method, the PCB number represents the relative 1 ordinal number of the PCB that is checked. Because an input or output PCB must be defined in each PSB, the minimum practical PCB number is 2.

column_definition

Provides SQL descriptions of the contents of the segments in the IMS database hierarchy that this table accesses. Optionally, the column can include one or more record arrays that identify repeating data in the sequential file. IMS databases have limited metadata. The DBD definition defines the segments that make up the database and its hierarchical structure.

A table must contain at least one column and can contain up to 5,000 columns. Columns are named, and each name must be unique within the table.

At a minimum, the DBD definition contains an IMS FIELD definition for the keys for each segment in the database and XDFLD definitions that identifies the keys of any secondary indexes. Whether the DBD contains additional FIELD statements that define the other fields in each segment, is site dependent. A common practice is to only define additional FIELD statements for those attributes that are referenced in segment search arguments (SSAs) that the applications issue. Then IMS filters the data that is returned to the application.

record_array_definition

Identifies repeating data. Record array definitions contain column definitions and possibly more record array definitions.

Federated queries can read record array data if you perform one of the following procedures:

- Map the columns in a flattened structure that provides a separate column for each array instance and field.
- Map a separate table for each array.

You can read, update, and capture changes to redefined data if you create a separate table and view for each record type. You can update and capture changes to record array data if you map the columns in a flattened structure.

PRIMARY KEY *column-name*

Identifies the columns that uniquely identify an IMS database record and the segment instances that this table references in an IMS database.

The specification of primary key information is always valid for a table that references an IMS database.

The ability to determine what constitutes a good set of primary key columns versus a bad set of columns depends primarily upon whether *perceived-root-segment-name* is explicitly specified or implicitly identified, for example by the root segment.

The primary key does not enforce the same restrictions that the CREATE INDEX statement does. There is no prohibition against identifying the columns that make up the primary key sequence field or a DEDB, HDAM or PHDAM database as primary key columns. Likewise, you can use primary key columns for child segments. References to columns that map to the sequence fields of all of the child segments in the database hierarchy are also good primary key column references. Under the assumption that the fully concatenated key is unique, references to child segment sequence fields that are not unique are also acceptable. A warning message is generated when a reference is made to a non-unique child sequence field.

Example

The following is an example of a CREATE TABLE statement for IMS.

```
CREATE TABLE CXAIMS.IMSALDB DBTYPE IMS
  FVT52901 INDEXROOT FVTRoot FVTRoot
  SCHEDULEPSB(PF52901U) PCBPREFIX FVT
(
  ALDBIMSKEY SOURCE DEFINITION ENTRY FVTRoot
  DATAMAP OFFSET 0 LENGTH 4 DATATYPE F
  USE AS INTEGER,
  ALDBIMSCHAR SOURCE DEFINITION ENTRY FVTRoot
  DATAMAP OFFSET 4 LENGTH 254 DATATYPE C
  USE AS CHAR(254)
  NULL IS X'4040',

  ALDBIMSLDECIMAL SOURCE DEFINITION ENTRY FVTRoot
  DATAMAP OFFSET 266 LENGTH 8 DATATYPE P
  USE AS DECIMAL(15,0)
  NULL IS X'0000000000000000C',
  ALDBIMSDECIMALMAX SOURCE DEFINITION ENTRY FVTRoot
  DATAMAP OFFSET 282 LENGTH 8 DATATYPE P
  USE AS DECIMAL(15,15)
  NULL IS X'0000000000000000C',
/*
  ALDBIMSLFLOAT SOURCE DEFINITION ENTRY FVTRoot
  DATAMAP OFFSET 294 LENGTH 8 DATATYPE D
  USE AS FLOAT(53)
  NULL IS X'0000',
  ALDBIMSDBLPREC SOURCE DEFINITION ENTRY FVTRoot
  DATAMAP OFFSET 302 LENGTH 8 DATATYPE D
  USE AS FLOAT(53)
  NULL IS X'0000',
  ALDBIMSINTEGER SOURCE DEFINITION ENTRY FVTRoot
  DATAMAP OFFSET 310 LENGTH 4 DATATYPE F
  USE AS INTEGER
  NULL IS X'00000000',
  ALDBIMSREAL SOURCE DEFINITION ENTRY FVTRoot
  DATAMAP OFFSET 314 LENGTH 4 DATATYPE D
  USE AS FLOAT(21)
  NULL IS X'0000',
  ALDBIMSSMALLINT SOURCE DEFINITION ENTRY FVTRoot
  DATAMAP OFFSET 318 LENGTH 2 DATATYPE H
  USE AS SMALLINT
  NULL IS X'0000',
  ALDBIMSVCHAR SOURCE DEFINITION ENTRY FVTRoot
  DATAMAP OFFSET 320 LENGTH 256 DATATYPE V
  USE AS VARCHAR(254)
  NULL IS X'4040',
/*
  ALDBIMSLVCHAR SOURCE DEFINITION ENTRY FVTRoot
  DATAMAP OFFSET 576 LENGTH 1026 DATATYPE V
  USE AS VARCHAR(1026)
  NULL IS X'4040',
```

```

/*                                     */
ALDBIMSLVGRAPHIC SOURCE DEFINITION ENTRY FVTRoot
DATAMAP OFFSET 2112 LENGTH 1025 DATATYPE V
USE AS VARGRAPHIC(1025)
/* USE AS LONG VARGRAPHIC             */
NULL IS X'4040',
PRIMARY KEY (ALDBIMSKEY)
);

```

Columns for IMS

You can define a column that references an IMS database. For IMS column definitions, you must also identify the name of the segment where the column resides.

Syntax

► *column-name*—SOURCE DEFINITION ENTRY—*segment-name*—*IMS-field-name*

► DATAMAP OFFSET—*relative-offset*—LENGTH—*length*—DATATYPE—C
P
D
F
H
V
UC
UP
UH
UF
B
VB

► USE AS—CHAR—(*length*)
VARIABLE—(*length*)
LONG VARIABLE
GRAPHIC—(*length*)
VARGRAPHIC—(*length*)
LONG VARGRAPHIC
INTEGER
SMALLINT
DECIMAL—(*precision, scale*)
DECIMAL—(*precision*)
FLOAT—(*precision*)
REAL
DOUBLE PRECISION
BINARY—(*length*)
VARBINARY—(*length*)

► WITHOUT CONVERSION
WITH CONVERSION—*field_procedure_name*—NULL IS—*null_value*

PRIMARY KEY

Parameters

column-name

Identifies the name of the column.

Specifies the column name that is a long identifier. Column names cannot be qualified with a CREATE TABLE statement.

segment-name

Identifies the segment where the column is located. The name is a short native identifier.

The name must also exist in the DBD specified in the CREATE TABLE statement.

DATAMAP

Specifies the relative offset for a column. If a LENGTH keyword is specified, information about the length of the column is also defined.

OFFSET *relative-offset*

Follows the DATAMAP keyword to set the offset for the column. For some data sources like Adabas and CA-IDMS, explicit offset and length information does not need to be supplied because the system data dictionaries can provide this information. In these cases, the column definition does not provide offset information. The column definition only identifies the element or field name that the column corresponds to in the data source data dictionary. If a column definition references a portion of a dictionary element or field, the grammar supports specifying column offset and length within the element or field. The column definition can identify the column offset of the start of the column and the length of the element or field that the column is being mapped to.

The relative offset identifies the relative zero offset of the starting position of the column within the object that the column is associated with. For simple objects like a VSAM or sequential file, the offset is generally measured from the start of the record. For more complex databases, like IMS or CA-IDMS, the relative offset is measured from the start of a segment or record. If the column is defined within a record array, then the relative offset is measured from the start of the record array that is measured from the start of the fragment that the column is associated with.

Note: Columns do not need to be defined in ascending relative offset starting sequence. When a mapping contains fixed length record arrays with additional columns following the record array, the starting offsets typically increase for the columns before the record array definition. For the columns in the record array, the relative offset information is reset to one and to larger numbers for those columns that exist after the record array.

LENGTH *length*

The LENGTH clause is required for IMS column definitions.

Specifies the length of the column. Generally, inconsistencies between the length that is specified using the LENGTH keyword and the length that is obtained from the SQL data type definition on the USE AS clause are ignored. For native DECIMAL data types the LENGTH must match the computed physical length of the column, based on the precision and scale specified in the

USE AS clause when the USE AS precision is non-zero. For USE AS DECIMAL definitions the LENGTH must match the computed physical length of the column when the scale is non-zero and greater than the precision.

Additionally, differences between the *length* and the SQL length specification for VARCHAR and VARGRAPHIC data types identify how to interpret the length attribute for the varying length column. The following rules are applied for VARCHAR data types:

- If the *length* and the VARCHAR lengths are identical, then the control length is assumed to include the length (2 bytes) that is taken up by the control length component.
- If the *length* is two bytes greater than the VARCHAR length, then the control length component is assumed to identify the actual length of the data.

For a varying length graphic string, the same kind of conventions are used. However, the lengths are expressed in DBCS characters. Therefore the rules are as follows:

- If the *length* and the VARGRAPHIC lengths are identical, then the control length is assumed to include the length (1 DBCS character, which is 2 bytes) that is taken up by the control length of a component.
- If the *length* is one character greater (two bytes) than the VARGRAPHIC length, then the control length component is assumed to identify the actual length of the data in characters.

Generally, the USE AS length must match the *length* value specified. However, for VARCHAR, the *length* can be two bytes off. For VARGRAPHIC, the length can be different by one. For these data types, the differences do not represent a conflict.

DATATYPE

Identifies the native format of the column.

The following table identifies the basic native data types

Table 78. Native data types

DATATYPE value	Contents	Standard SQL data type	Other SQL data types
C	Mixed mode character data. When the SQL data type is DECIMAL, the data is assumed to consist wholly of numbers with the right most number identifying the sign.	CHAR DECIMAL	DECIMAL, VARCHAR, GRAPHIC, or VARGRAPHIC, BINARY, VARBINARY
P	Packed decimal data where the sign is stored in the far right aggregation of four bits.	DECIMAL	BINARY

Table 78. Native data types (continued)

DATATYPE value	Contents	Standard SQL data type	Other SQL data types
D ¹	Floating point data. The columns length or precision determines whether the SQL data type is DOUBLE PRECISION or DECIMAL. 64-bit data is mapped as DECIMAL.	DOUBLE PRECISION	FLOAT(<i>precision</i>) DECIMAL, BINARY
F ²	32-bit signed binary value where the sign is in the high order bit.	INTEGER REAL	FLOAT(<i>precision</i>) BINARY
H	16-bit signed binary value where the sign is in the high order bit.	SMALLINT	BINARY
V	Variable mixed-mode character data, where the actual data is preceded by a 16-bit signed binary number that identifies the actual length of the data.	VARCHAR	LONG VARCHAR, VARGRAPHIC, or LONG VARGRAPHIC, VARBINARY
UC	Unsigned zoned-decimal data where the last character does not identify the sign. The value is always a positive value.	DECIMAL	CHAR, BINARY, VARBINARY
UP	Packed decimal data where the sign bit is ignored. The value is always positive.	DECIMAL	BINARY
UF	Unsigned 32-bit binary value.	INTEGER	BINARY
UH	Unsigned 16-bit binary value.	SMALLINT	BINARY
B ³	Fixed length binary data.	BINARY	DECIMAL, INTEGER, SMALLINT, VARBINARY
VB ³	Variable length binary data, where the actual data is preceded by a 16-bit signed binary number that identifies the actual length of the data.	VARBINARY	N/A

Table 78. Native data types (continued)

DATATYPE value	Contents	Standard SQL data type	Other SQL data types
			¹ The SQL data type is DOUBLE PRECISION or FLOAT. DOUBLE PRECISION is shorthand for an 8-byte floating point number. In the USE AS clause, you can identify this data type as DOUBLE PRECISION or FLOAT(precision). If the precision value is in the range 22–53, the column represents an 8-byte floating point number. If you assign FLOAT, specify the maximum precision. The column length is based on the precision.
			² The SQL data type is REAL or FLOAT. REAL is shorthand for a 4-byte floating point number. In the USE AS clause, you can identify this data type as REAL or FLOAT(precision). If the precision value is in the range 1–21, the column represents a 4-byte floating point number. If you assign FLOAT, specify the maximum precision. The column length is based on the precision.

If DATATYPE information is not specified, the native data type is synthesized based on the column of the SQL data type or from the database system.

The SQL data type information in the USE AS clause identifies the value in the SIGNED column in the following instances:

- One character code is supplied.
- No DATATYPE information is specified.
- Native data type information is not obtained from the database system.

USE AS

Identifies the SQL data type for the column.

The following table describes the data types for columns. The non-null SQLTYPE identifies the data type in internal control blocks and diagnostic trace information.

Table 79. SQL data type descriptions

Keyword identifier	Description	Maximum length	SQLTYPE
CHAR(length)	Fixed-length character string that contains mixed mode data.	254	452
VARCHAR(length)	Variable length character string that contains mixed mode data. A half-word length component precedes the character string and identifies the actual length of the data. The VARCHAR length does not include the length of the LENGTH field.	32704	448
LONG VARCHAR	Long character string that contains mixed mode data. A half-word length component precedes the character string and identifies the actual length of the data. The LONG VARCHAR length does not include the length of the LENGTH field.	32704	456
GRAPHIC(length)	Fixed-length graphic string that is assumed to contain pure DBCS data without shift codes. The length is expressed in DBCS characters, and not bytes.	127	468

Table 79. SQL data type descriptions (continued)

Keyword identifier	Description	Maximum length	SQLTYPE
VARGRAPHIC(length)	Varying-length graphic string that is assumed to contain pure DBCS data without shift codes. A half-word length component precedes the graphic string and identifies the actual length of the data. The length is expressed in DBCS characters, and not bytes.	16352	464
LONG VARGRAPHIC	Long graphic string that is assumed to contain pure DBCS data without shift codes. A half-word length component precedes the graphic string and identifies the actual length of the data. The length is expressed in DBCS characters, and not bytes.	16352	472
INTEGER	Large integer.	Exactly 4	496
SMALLINT	Small integer.	Exactly 2	500
DECIMAL (precision,scale) or DECIMAL(precision)	Packed decimal data. A valid precision value is between 1 and 31. A valid scale value is between zero and the precision value.	31	484
FLOAT(precision)	Floating point number. Depending upon the precision, a column with a FLOAT data type takes on the attributes of a REAL or DOUBLE PRECISION data type. When precision is in the range of 1 to 21, the column is treated as a REAL. For precisions between 22 and 53, the column takes on DOUBLE PRECISION attributes. A precision of zero or greater than 53 is nonvalid.	4 or 8	480
REAL	A single-precision, floating-point number that is a 32-bit approximation of a real number. The number can be zero or can range from -3.40282347E+38 to -1.17549435E-38, or from 1.17549435E-38 to 3.40282347E+38.	4	480
DOUBLE PRECISION	A floating-point number that is a 64-bit approximation of a real number. The number can be zero or can range from -1.797693134862315E+308 to -2.225073858507201E-308, or from 2.225073858507201E-308 to 1.797693134862315E+308.	8	480
BINARY(length)	Fixed-length binary data.	32704	912
VARBINARY(length)	Variable length binary data, where the actual data is preceded by a 16-bit signed binary number that identifies the actual length of the data. The VARBINARY length does not include the length of the LENGTH field.	32704	908

When you supply a USE AS LONG VARCHAR or USE AS LONG VARGRAPHIC clause, you do not specify a length for the column. The length is based on the physical attributes that are obtained about the object record or segment where the column is.

When you supply a USE AS VARCHAR clause with a length greater than 254, that column is changed into a LONG VARCHAR column with the specified length. The same behavior occurs for a USE AS VARGRAPHIC clause when the length is greater than 127.

WITHOUT CONVERSION

Specifies that a field procedure does not exist for the column.

WITH CONVERSION

Identifies the name of the load module for a field procedure, if a field procedure is required. The field procedure name is a short identifier.

NULL IS *null_value*

Specifies a value that identifies when the contents of the column is null. By default, the data in a column never contains a null value. This situation is true even for variable length character or graphic columns where the length component indicates that there is no data in the variable length column.

A null value is a character string that can specify up to 16 characters of data. Specifies the value in hexadecimal format. The total length of the string is 35 characters.

A column contains null values based on *null-value*. If the start of the column data matches the null value, then that instance of the column is null. For example, if a column is defined as CHAR(10) and a null value of x'4040' is specified, the null value length is 2. If the first 2 bytes of the column contain spaces, the column is also null.

PRIMARY KEY

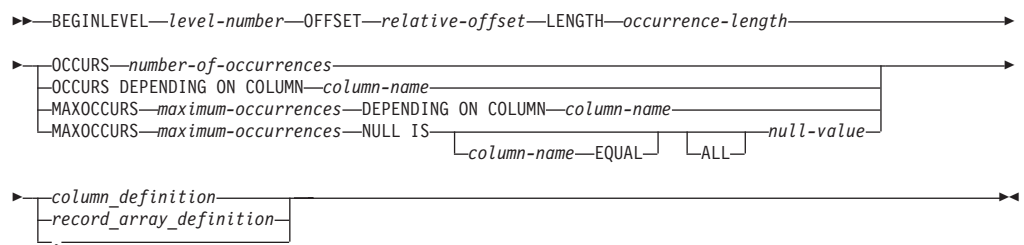
Identifies the column to be one of the columns that constitute the primary key for the table. This form of primary key identification is mutually exclusive with specifying a primary key at the end of the CREATE TABLE statement.

You can identify a primary key at the column level when the columns that make up the primary key for the table are defined in ordinal sequence within the table definition.

Record arrays for IMS

Record array definitions are used to identify and handle data that can repeat multiple times within a non-DB2 database or file system.

Syntax



Parameters

BEGINLEVEL *level-number*

Identifies the start of a record array definition.

The *level-number* on a BEGINLEVEL or ENDDLEVEL definition identifies the nesting level of the record array within the parent fragment. Multiple record arrays can exist with the same *level-number*. Typically, you do not map multiple record arrays within a single table. Create separate tables for each record array on the same level to avoid result sets that are too large.

The following example shows a multiple record array. In this example, the EMPL-ADDRESS data item occurs three times and the EMPL-DEPENDENTS, EMPL-DEP-NAME and EMPL-DEP-DOB data items occur ten times. If you map both arrays to the same table, the query processor generates 30 rows for each EMPL-RECORD read. Each instance of EMPL-ADDRESS is combined with each instance of EMPL-DEPENDENTS to create 30 rows (3 addresses times 10 dependents).

```
01 EMPL-RECORD.
  05 EMPL-SSN          PIC 9(9).
  05 EMPL_NAME
    10 EMPL-LAST      PIC X(30).
    10 EMPL-FIRST     PIC X(30).
    10 EMPL_MI        PIC X.
  05 EMPL-ADDRESS      PIC X(30) OCCURS 3 TIMES.
  05 EMPL-DEPENDENTS  OCCURS 10 TIMES.
    10 EMPL-DEP-NAME  PIC X(30).
    10 EMPL-DEP-DOB   PIC 9(8).
```

If you must access the information in the above example in a single table definition, then two record array definitions are required. These record arrays are both level 1 arrays, because they are not nested within other arrays. To avoid performance problems related to result sets that are too large, flatten the structure or define a separate table for one or both arrays.

OFFSET *relative-offset*

The relative offset identifies the relative starting position of the record array within the segment or within a parent record array definition.

The relative offset is a required numeric parameter.

LENGTH *occurrence-length*

Identifies the length in bytes of one occurrence of the repeating data.

MAXOCCURS *maximum-occurrences* **DEPENDING ON COLUMN** *column-name*

Identifies a record array that occurs a variable number of times. A column definition that appears before the record array definition determines the number of instances.

You cannot map columns that occur after a variable length array, because the starting offsets of data items that are defined after an OCCURS DEPENDING ON definition are not at a predictable offset.

The *maximum-occurrences* value must be numeric, and *column-name* must specify the name of a column in the table. The control column must contain a numeric value. The query processor supports SQL types of SMALLINT, INTEGER, DECIMAL, or CHAR (if the column contains zoned decimal data).

MAXOCCURS *maximum-occurrences* **NULL IS**

Defines a record array that repeats a fixed number of times. If the initial bytes of the first column in the record array match *null-value*, that instance of the array is null.

The *maximum-occurrences* value must be numeric. The null value can be a character string of up to 32 characters. The null-value can also be a 67-character hexadecimal string that consists of 64 hexadecimal nibbles, with a leading x and surrounding quotation marks, for example, x'4040'.

ENDLEVEL *level-number*

Identifies the end of a record array definition.

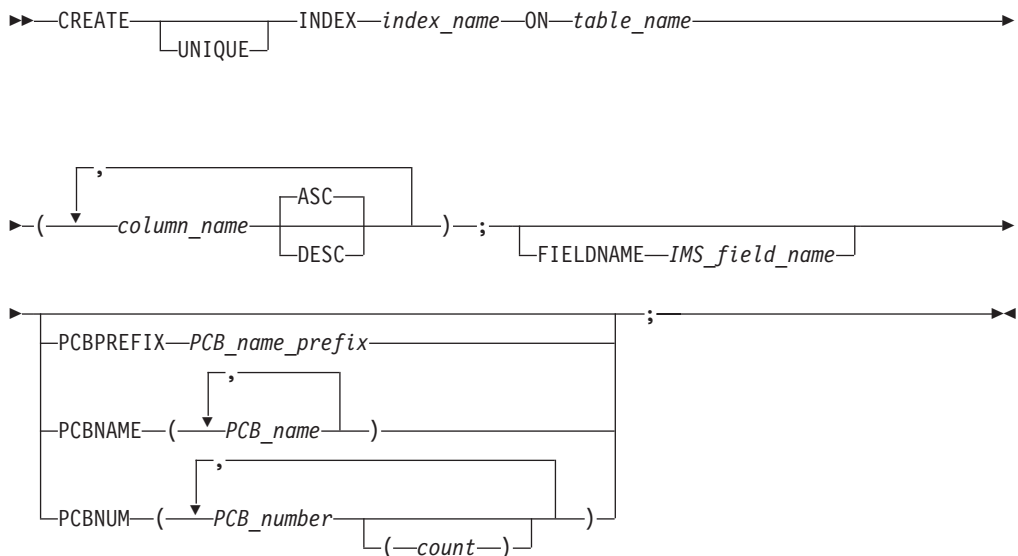
CREATE INDEX statement for IMS

You can use the CREATE INDEX statement to define an index that references the columns that make up the IMS primary key (FIELD). You can also define an index that references the columns that map to the SRCH fields in the XDFLD statement of a secondary index definition to access the IMS database.

Indexes identify columns in a table that correspond to a physical index that is in the source database. The query processor uses the contents of the columns that are referenced in a WHERE clause to create the index. The query processor attempts to build either a full key value to use in database access or a partial key value. The partial key can be used to perform a range scan against the target database to reduce the number of records that are accessed.

Although you might not be able to define an index against columns that correspond to the primary key, you can identify these columns as primary key columns. This primary key information does not affect existing connector index selection and access optimization. The primary key information is made available for use by front-end tools, where key information is either required or beneficial.

Syntax



Parameters

CREATE INDEX *index_name*

Identifies the SQL statement as an index definition statement. A unique index does not have any restrictions about the columns that make up the index. For example, a unique index column can contain null values.

If qualified, the index name is a two-part name, and the authorization ID that qualifies the name is the owner of the index. If an unqualified table name is supplied, the owner name is the authorization ID from the CURRENT SQLID special register.

ON *table_name*

Identifies the table for which the index is being defined. The table name can be a qualified or unqualified table name. For unqualified names, the table owner is from the CURRENT SQLID special register.

The table name is validated with the standard syntax checks that are associated with identifiers, and then the existence of the table is verified. Do not define an index on a view.

column_name **ASC/DESC**

Specifies column names that make up the index key. Optionally, you can identify whether the column is stored or accessed in ascending (ASC) or descending (DESC) key sequence. By default, the key is in ascending key sequence.

There are no fixed limits on the number of columns that you can identify as key columns for an index. The only requirements are as follows:

- The column must exist in the table.
- The column must map to what constitutes a key in the target database. In most cases, this determination is based on the starting offset and length of the column as compared to the starting offset and length of the key in the target database.
- The column cannot overlap another column within the key definition. This determination is based on the starting offset and length of a column as compared to the starting offsets and length of the other columns that are identified as key columns for the index definition.
- For the key column, generally varying length and graphic data types are not supported.

Any kind of varying-length character or graphic string is not allowed.

If the CREATE TABLE statement specifies a segment for the INDEXROOT keyword and that segment is not a root segment, the key columns can only reference an XDFLD that is defined by INDEXROOT. The key columns reference offsets and lengths in the source segment that the XDFLD references. If INDEXROOT is not specified in the CREATE TABLE statement or the segment is the root segment for the DBD, key columns can be specified for any XDFLD and can reference columns that map to the primary key sequence field of the database.

If *DBD_name* in the CREATE TABLE statement references a logical DBD, the key columns can reference an XDFLD in either of the physical segments that are referenced by the INDEXROOT segment. However, if INDEXROOT is not specified in the CREATE TABLE statement or the segment is the root segment for the DBD, primary key references are supported only for the first physical database that is referenced by the logical DBD.

If the index is defined against the root segment of the database and the *DBD_NAME* in the CREATE TABLE statement is a DEDB, HDAM or PHDAM database, the index cannot consist of key columns that reference the primary key sequence field of the IMS database. This also holds true if the *DBD_NAME*

in the CREATE TABLE statement references a logical database and the root segment of the first physical database is a DEDB, HDAM or PHDAM database

PCBPREFIX *PCB_name_prefix*

Identifies the PCBs that access the IMS database when the index is selected based on the WHERE clause. *PCB_name_prefix* is 1 to 7 characters in length and follows IMS PCB naming conventions.

If a PCBPREFIX is in the IMS CREATE TABLE statement, you must specify a PCBPREFIX in the CREATE INDEX statement. When the index accesses the table, the index-level PCBPREFIX is used in preference to any PCB prefix information at the table level.

If the index corresponds to the primary key sequence field of the IMS database, the IMS PCB definitions that correspond to the PCBPREFIX information must not contain a PROCSEQ definition.

Also, if the index definition corresponds to an XDFLD definition, the PCBs must contain a PROCSEQ. This process identifies the secondary index DBD that corresponds to the XDFLD that the index key columns match.

PCBNAME (*PCB_name,...*)

Specifies up to 5 PCBs that access an IMS database through a table. Multiple PCBs are required if the same table is referenced more than once in an SQL statement, or when the same PCB is associated with more than one table and these additional tables are referenced in a single SQL statement.

Each PCB name in the list can be up to 8 characters long. For multiple PCB names, separate each name by a comma.

PCBNUM (*PCB_number (count),...*)

Specifies a number of PCBs to access the IMS database for the table. Multiple PCBs are required under either of these conditions:

- The same table is referenced more than once in an SQL statement.
- The same PSB is associated with more than one table, the PCBs in the PSB have sensitivity to the segments that the table is accessing, the same PCB ordinal numbers are specified for these tables, and these additional tables are referenced in a single SQL statement.

You can specify up to ten sets of PCB number ranges. These PCB numbers can be listed in any order and represent the order in which a PSB is checked to determine whether a PCB is used to access the IMS database.

For each item in the list, two different formats are used to identify the PCB numbers to be checked. The simplest method is to identify the PCB numbers to be checked separated by commas. The second technique identifies a range specification that consists of a starting PCB number that is followed by parenthesis and a number that identifies the number of PCBs to be checked from the starting number.

For either technique, the PCB number represents the relative 1 ordinal number of the PCB that is to be checked. Because an I/O PCB needs to be defined in each PSB that Classic Federation uses, the minimum practical PCB number that can be specified starts at two.

Example

The following is an example of a CREATE INDEX statement for IMS.

```
CREATE UNIQUE INDEX CXAIMS.IMSALDB_IDX1 ON CXAIMS.IMSALDB (ALDBIMSKY ASC);
```

ALTER TABLE statement for IMS

You can use the ALTER TABLE statement to create tables that reference an IMS database.

Authorization

You must have the one of the following authorities to run the ALTER TABLE statement:

- SYSADM
- DBADM for the database type that is referenced in the DBNAME column for the table being altered
- Ownership of the table being altered

Syntax

```
▶▶ALTER TABLE— table-name— DATA CAPTURE— 

|         |
|---------|
| CHANGES |
| NONE    |

— ; —▶▶
```

Parameters

table-name

Identifies the table for which the DATA CAPTURE is turned on or off. The table name can be a qualified or unqualified table name. If an unqualified name is supplied, the table owner is from the CURRENT SQLID special register.

DATA CAPTURE

Indicates that the ALTER TABLE statement is setting the value of the DATA CAPTURE flag.

CHANGES

Turns change capture on, which enables change-capture agents to capture changes to the data that the table is mapped to.

NONE

Turns change capture off. Change-capture agents do not capture changes to the data that the table is mapped to.

Example

The following is an example of an ALTER TABLE statement for IMS.

```
ALTER TABLE CXAIMS.IMSALDB DATA CAPTURE CHANGES;
```

VSAM

You can use the CREATE TABLE and ALTER TABLE statements to define tables that reference VSAM files. You can use the CREATE INDEX statement to define indexes that reference VSAM files.

CREATE TABLE statement for VSAM

You can use the CREATE TABLE statement to define a logical table that references a VSAM file.

Authorization

The connected user ID must have one of the following privileges to run the CREATE TABLE statement:

- SYSADM
- DBADM for the database type that is referenced in the DBTYPE clause

The owner has all table privileges on the table (such as SELECT, UPDATE, and so on) and the authority to drop the table. The owner can grant equivalent use privileges on the table.

Syntax

```
► CREATE TABLE table-name DBTYPE VSAM { DD DD-name | DS dataset-name }  
  
| CICS_connection_information |  
  
| RECORD EXIT exit-name { MAXLENGTH length }  
  
► ( { column_definition | record_array_definition }  
| , PRIMARY KEY ( column-name ) ) ;
```

CICS_connection_information:

```
| CICS APPLID local-LU-name CICS-LU-name LOGMODE logmode-name |  
► TRANID CICS-transaction-ID { NETNAME network-name }  
| DSN data-set-name |
```

Parameters

authorization-ID.table-name

Identifies the owner of the table and the name of the table that you want to create.

If you do not provide an authorization ID, the ID in the CURRENT SQLID special register is used.

You must create more than one table if you map to a database or a file that meets either of these criteria:

- The database or file contains repeating data.
- The database or file contains information about one or more distinct sub-objects, because the database or file is not designed to follow the third normalization rules, which are part of the standards to eliminate redundancies and inconsistencies in table data. In a table designed according to third normalization rules, each non-key column is independent of other non-key columns, and is dependent only upon the key.

DBTYPE VSAM

Specifies that the CREATE TABLE statement defines a logical table that references a VSAM file.

DD *DD-name*

Specifies the VSAM file that the table maps to. Specifies either a local file reference or a reference to a CICS file definition table (FDT) entry name.

If *DD-name* represents a local file reference, a DD clause that references the VSAM file must exist in the Classic federation data server JCL.

If *DD-name* references an FDT entry name, you must specify CICS connection information with the CICS_connection_information clause.

DD-name must correspond to a cluster or alternate index path definition for a VSAM ESDS, KSDS, or RRDS data set.

DD-name is a short native identifier that conforms to the naming conventions for a DD statement in job control language (JCL) on IBM z/OS.

DS *data-set-name*

Specifies the VSAM file that the table definition accesses. Designates a VSAM cluster definition or a PATH component when the VSAM file is accessed from a VSAM alternate index.

The data set name must correspond to a cluster or alternate index path definition for a VSAM ESDS, KSDS, or RRDS data set.

The name can be 1 to 44 characters in length and must follow z/OS naming conventions for VSAM data sets.

CICS_connection_information

Specifies CICS connection information that is required only if your VSAM files are managed by CICS.

For Classic federation, you must specify CICS connection information that identifies the CICS subsystem where the VSAM file is located.

CICS APPLID

Specifies the names of two logical units (LUs) that establish communication with the target CICS subsystem.

local-LU-name

This value identifies the name of a local LU that is used to communicate with CICS. This identifier is 1 to 8 characters in length.

local-LU-name corresponds to either the ACBNAME or the name (label) on the IBM VTAM APPL definition. *local-LU-name* must be active on the image where the server runs.

The name follows VTAM naming conventions. The SASCSAMP member CACCAPPL provides the sample local LU definitions. The sample names for *local-LU-name* are CACCICS1 and CACCICS2, which you can modify. Define the name to CICS as a CONNECTION definition. Sample connection definitions for CACCICS1 and CACCICS2 are in SASCSAMP member CACCDEF.

CICS-LU-name

Designates the VTAM LU 6.2 definition that a CICS region listens on for connection requests. This LU name is 1 to 8 characters in length. The name corresponds to the value of the **APPLID** parameter in the system initialization definition (DFHSIT macro) of the target CICS subsystem where the VSAM file is located. The name follows VTAM naming conventions.

LOGMODE *logmode-name*

Identifies the name of the VTAM log mode table that controls session establishment and the service-level properties of the session that CICS determines. The name is a short native identifier that designates the name of the VTAM logon-mode table that controls the session parameters for the conversation that is established between the local LU and the CICS LU. This name is 1 to 8 characters in length. The logon-mode table name corresponds to a z/OS load module that is accessible to VTAM. The definition for a Classic logon-mode table is in SASCSAMP member CACCMODE.

TRANID *CICS-transaction-ID*

Performs the following functions:

- Identifies the name of the Classic-supplied transaction that verifies the existence of the VSAM file

The name of the CICS transaction is used for data access and validation purposes. This name is 1 to 4 characters in length. In prior releases, *CICS-transaction-ID* was for data access only, and a separate transaction performed validation checking. In version 9.1, these two CICS transactions are combined. *CICS-transaction-ID* corresponds to the CICS TRANSACTION definition.

The SCACSAMP member CACCDEFS provides sample CICS transaction, connection, program, and session definitions. The sample *CICS-transaction-ID* identifier is EXV1, which you can modify.

NETNAME *network-name*

Enables the federation data server to communicate with a remote CICS system by identifying the name of the network where the CICS subsystem is running.

The name of the network where *CICS-LU-name* resides corresponds to the CICS subsystem that accesses a VSAM file. This identifier is 1 to 8 characters in length.

The name is identified on the NETWORK VTAM macro definition on the local image to identify the remote SNA network where the CICS subsystem resides.

The name follows VTAM naming conventions.

DSN *data-set-name*

Identifies the name of a data set that is associated with a file in the CICS system. If the table is not going to be used for replication, then this DSN *data-set-name* clause is not required. It is only required for Classic CDC for VSAM when the CICS file definition is a remote file in a File Owning Region (FOR). When using Classic CDC for VSAM, CDC requires the data set name from which to capture changes. Using this data set name, Classic CDC can then find the correct log stream that contains the changes to capture.

If the CICS file being mapped is local to the CICS region specified, then the *data-set-name* is available and returned to the QP. In this case, the DSN clause is not necessary. If it is specified, then the *data-set-name* that is specified is compared to the data set name on the File Control Table (FCT) entry in CICS when the **CREATE TABLE** command is run.

If the CICS file being mapped is a remote definition in the CICS region that is specified, then the data set name is not available, and the DSN *data-set-name* clause is required so that Classic CDC has the required data set name that is needed for data capture. This is enforced when the **CREATE TABLE** statement is run.

The following validation rules apply when the DSN *data-set-name* is specified. In each case, an error is returned if there is a problem.

- If the CICS file is local to the CICS region, then the DSN name that is specified must match the data set name that is returned from CICS.
- If the CICS file is remote to the CICS region (no data set name returned from CICS), the data set name that is specified by DSN must exist and is validated through the catalog search interface.
- If the CICS file is remote to the CICS region, then the data set name must be specified on the **CREATE TABLE** statement.
- The data set name that is specified must not exceed 44 characters.

RECORD EXIT *exit-name*

Identifies that a record processing exit is invoked for postprocessing, after a record is retrieved from the VSAM file. The record processing exit is also called for preprocessing before a VSAM record is inserted or updated. The record exit is used for altering the record contents or filtering specified records.

You can use the record processing exit with Classic federation from VSAM files.

The name is a short identifier that follows the naming conventions for a z/OS load module.

MAXLENGTH

Specifies the maximum length, in bytes, of the buffer that calling programs can pass to the record exit. The buffer must be large enough to accommodate the largest possible input or output record.

column_definition

Provides SQL descriptions of the contents of the VSAM file. Column mapping must be based on COBOL copybook definitions.

A table must contain at least one column and can contain up to 5,000 columns. Columns are named, and each name must be unique within the table.

record_array_definition

Identifies repeating data. Record array definitions contain column definitions and possibly more record array definitions.

Federated queries can read record array data if you perform one of the following procedures:

- Map the columns in a flattened structure that provides a separate column for each array instance and field.
- Map a separate table for each array.

PRIMARY KEY *column-name*

Specifies the column names that uniquely identify each record in the VSAM file. Depending upon the type of VSAM file that the CREATE TABLE references, statement specification of primary key columns might not be appropriate.

Corresponds to what constitutes the target database or file system equivalent of a primary key. The column or columns logically identify a unique row. Each table must have a primary key consisting of the set of columns that uniquely identifies each record.

You can use either of two approaches to define a primary key. Use the following criteria to determine whether to specify the PRIMARY KEY clause on the column definition or the CREATE TABLE statement.

- If the columns that make up a composite key are defined in the same sequence as represented by the actual primary key, PRIMARY KEY can be specified on the column definitions. When this form of primary key identification is used, the sequence in which the columns are identified represents the ordinal position that you must use to construct a composite key that uniquely identifies a row in the table.
- When PRIMARY KEY information is specified on the main CREATE TABLE statement, the columns do not need to be defined in ordinal sequence.

The following is an example of a CREATE TABLE statement for VSAM. This table includes an array to capture data about employee dependents, but the number of array occurrences depends on the number of dependents, that is to say, the value of EMPL_DEP_COUNT.

```
CREATE TABLE "DBA"."EMPLOYEE" DBTYPE VSAM
DS "SAMPLE.VSAM.EMPLOYEE"
(
    "EMPL_LAST_NAME" SOURCE DEFINITION
    DATAMAP OFFSET 0 LENGTH 20
    DATATYPE C
    USE AS CHAR(20),
    "EMPL_FIRST_NAME" SOURCE DEFINITION
    DATAMAP OFFSET 20 LENGTH 20
    DATATYPE C
    USE AS CHAR(20),
    "EMPL_GENDER" SOURCE DEFINITION
    DATAMAP OFFSET 40 LENGTH 1
    DATATYPE C
    USE AS CHAR(1),
    "EMPL_SSN" SOURCE DEFINITION
    DATAMAP OFFSET 41 LENGTH 9
    DATATYPE UC
    USE AS CHAR(9),
    "EMPL_DOB" SOURCE DEFINITION
    DATAMAP OFFSET 50 LENGTH 4
    DATATYPE UP
    USE AS DECIMAL(6 , 0),
    "EMPL_ADDRESS_1" SOURCE DEFINITION
```

```

DATAMAP OFFSET 54 LENGTH 20
DATATYPE C
USE AS CHAR(20),
"EMPL_ADDRESS_2" SOURCE DEFINITION
DATAMAP OFFSET 74 LENGTH 20
DATATYPE C
USE AS CHAR(20),
"EMPL_STATE" SOURCE DEFINITION
DATAMAP OFFSET 96 LENGTH 2
DATATYPE C
USE AS CHAR(2),
"EMPL_ZIP" SOURCE DEFINITION
DATAMAP OFFSET 98 LENGTH 5
DATATYPE UC
USE AS CHAR(5),
"EMPL_DEP_COUNT" SOURCE DEFINITION
DATAMAP OFFSET 103 LENGTH 2
DATATYPE UH
USE AS SMALLINT,
"EMPL_COUNT" SOURCE DEFINITION
DATAMAP OFFSET 105 LENGTH 2
DATATYPE UH
USE AS SMALLINT,
BEGINLEVEL 1 OFFSET 107 LENGTH 54
MAXOCCURS 20
DEPENDING ON COLUMN "EMPL_DEP_COUNT",
"DEP_LAST_NAME" SOURCE DEFINITION
DATAMAP OFFSET 0 LENGTH 20
DATATYPE C
USE AS CHAR(20),
"DEP_FIRST_NAME" SOURCE DEFINITION
DATAMAP OFFSET 20 LENGTH 20
DATATYPE C
USE AS CHAR(20),
"DEP_GENDER" SOURCE DEFINITION
DATAMAP OFFSET 40 LENGTH 1
DATATYPE C
USE AS CHAR(1),
"DEP_SSN" SOURCE DEFINITION
DATAMAP OFFSET 41 LENGTH 9
DATATYPE UC
USE AS CHAR(9),
"DEP_DOB" SOURCE DEFINITION
DATAMAP OFFSET 50 LENGTH 4
DATATYPE UP
USE AS DECIMAL(6 , 0),
ENDLEVEL 1);

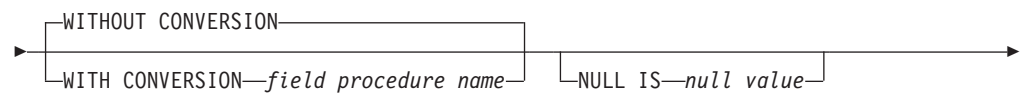
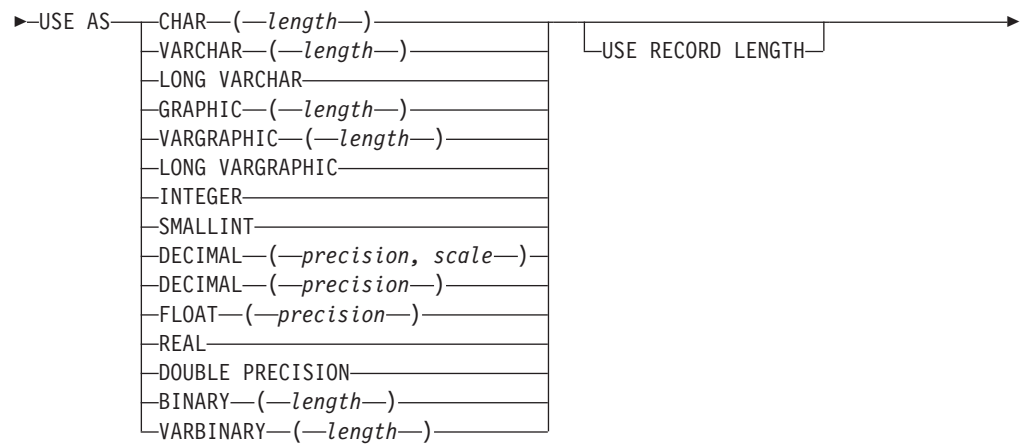
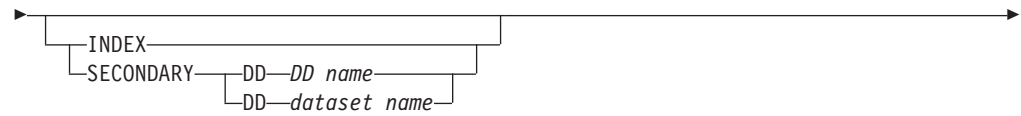
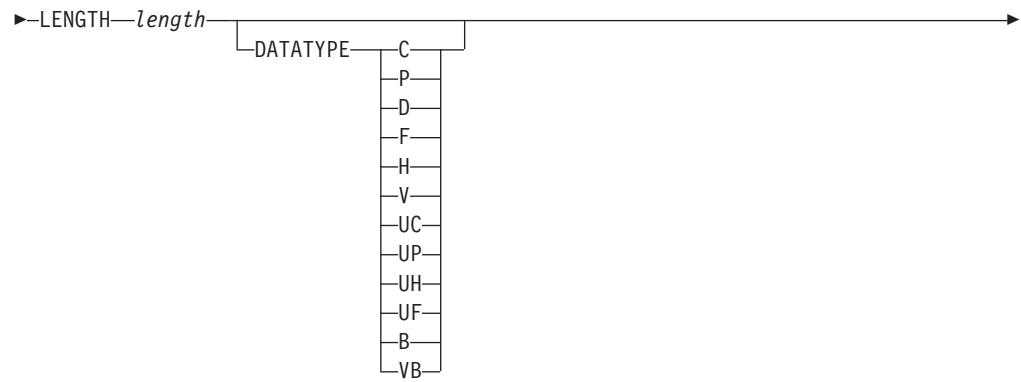
```

Columns for VSAM

Column definitions are a part of CREATE TABLE statements. You use column definitions to define the columns in a table that references a VSAM file. There are no differences between a sequential column definition and the generic column definitions.

Syntax

► *column-name*—SOURCE DEFINITION—DATAMAP OFFSET—*relative-offset*—►



Parameters

column_name

Identifies the name of the column.

Specifies the column name that is a long identifier. Column names cannot be qualified with a CREATE TABLE statement.

DATAMAP

Specifies the relative offset for a column. If a LENGTH keyword is specified, information about the length of the column is also defined.

OFFSET *relative_offset*

Follows the DATAMAP keyword to set the offset for the column. For some data sources like Adabas and CA-IDMS, explicit offset and length information does not need to be supplied because the system data dictionaries can provide this information. In these cases, the column definition does not provide offset information. The column definition only identifies the element or field name that the column corresponds to in the data source data dictionary. If a column definition references a portion of a dictionary element or field, the grammar supports specifying column offset and length within the element or field. The column definition can identify the column offset of the start of the column and the length of the element or field that the column is being mapped to.

The relative offset identifies the relative zero offset of the starting position of the column within the object that the column is associated with. For simple objects like a VSAM or sequential file, the offset is generally measured from the start of the record. For more complex databases, like IMS or CA-IDMS, the relative offset is measured from the start of a segment or record. If the column is defined within a record array, then the relative offset is measured from the start of the record array that is measured from the start of the fragment that the column is associated with.

Note: Columns do not need to be defined in ascending relative offset starting sequence. When a mapping contains fixed length record arrays with additional columns following the record array, the starting offsets typically increase for the columns before the record array definition. For the columns in the record array, the relative offset information is reset to one and to larger numbers for those columns that exist after the record array.

LENGTH *length*

Specifies the length of the column. Generally, inconsistencies between the length that is specified using the LENGTH keyword and the length that is obtained from the SQL data type definition on the USE AS clause are ignored. For native DECIMAL data types the LENGTH must match the computed physical length of the column, based on the precision and scale specified in the USE AS clause when the USE AS precision is non-zero. For USE AS DECIMAL definitions the LENGTH must match the computed physical length of the column when the scale is non-zero and greater than the precision.

Additionally, differences between the *length* and the SQL length specification for VARCHAR and VARGRAPHIC data types identify how to interpret the length attribute for the varying length column. The following rules are applied for VARCHAR data types:

- If the *length* and the VARCHAR lengths are identical, then the control length is assumed to include the length (2 bytes) that is taken up by the control length component.
- If the *length* is two bytes greater than the VARCHAR length, then the control length component is assumed to identify the actual length of the data.

For a varying length graphic string, the same kind of conventions are used. However, the lengths are expressed in DBCS characters. Therefore the rules are as follows:

- If the *length* and the VARGRAPHIC lengths are identical, then the control length is assumed to include the length (1 DBCS character, which is 2 bytes) that is taken up by the control length of a component.
- If the *length* is one character greater (two bytes) than the VARGRAPHIC length, then the control length component is assumed to identify the actual length of the data in characters.

Generally, the USE AS length must match the *length* value specified. However, for VARCHAR, the *length* can be two bytes off. For VARGRAPHIC, the length can be different by one. For these data types, the differences do not represent a conflict.

DATATYPE

Identifies the native format of the column.

The following table identifies the basic native data types

Table 80. Native data types

DATATYPE value	Contents	Standard SQL data type	Other SQL data types
C	Mixed mode character data. When the SQL data type is DECIMAL, the data is assumed to consist wholly of numbers with the right most number identifying the sign.	CHAR DECIMAL	DECIMAL, VARCHAR, GRAPHIC, or VARGRAPHIC, BINARY, VARBINARY
P	Packed decimal data where the sign is stored in the far right aggregation of four bits.	DECIMAL	BINARY
D ¹	Floating point data. The columns length or precision determines whether the SQL data type is DOUBLE PRECISION or DECIMAL. 64-bit data is mapped as DECIMAL.	DOUBLE PRECISION	FLOAT(<i>precision</i>) DECIMAL, BINARY
F ²	32-bit signed binary value where the sign is in the high order bit.	INTEGER REAL	FLOAT(<i>precision</i>) BINARY
H	16-bit signed binary value where the sign is in the high order bit.	SMALLINT	BINARY
V	Variable mixed-mode character data, where the actual data is preceded by a 16-bit signed binary number that identifies the actual length of the data.	VARCHAR	LONG VARCHAR, VARGRAPHIC, or LONG VARGRAPHIC, VARBINARY

Table 80. Native data types (continued)

DATATYPE value	Contents	Standard SQL data type	Other SQL data types
UC	Unsigned zoned-decimal data where the last character does not identify the sign. The value is always a positive value.	DECIMAL	CHAR, BINARY, VARBINARY
UP	Packed decimal data where the sign bit is ignored. The value is always positive.	DECIMAL	BINARY
UF	Unsigned 32-bit binary value.	INTEGER	BINARY
UH	Unsigned 16-bit binary value.	SMALLINT	BINARY
B ³	Fixed length binary data.	BINARY	DECIMAL, INTEGER, SMALLINT, VARBINARY
VB ³	Variable length binary data, where the actual data is preceded by a 16-bit signed binary number that identifies the actual length of the data.	VARBINARY	N/A

¹The SQL data type is DOUBLE PRECISION or FLOAT. DOUBLE PRECISION is shorthand for an 8-byte floating point number. In the USE AS clause, you can identify this data type as DOUBLE PRECISION or FLOAT(precision). If the precision value is in the range 22–53, the column represents an 8-byte floating point number. If you assign FLOAT, specify the maximum precision. The column length is based on the precision.

²The SQL data type is REAL or FLOAT. REAL is shorthand for a 4-byte floating point number. In the USE AS clause, you can identify this data type as REAL or FLOAT(precision). If the precision value is in the range 1–21, the column represents a 4-byte floating point number. If you assign FLOAT, specify the maximum precision. The column length is based on the precision.

If DATATYPE information is not specified, the native data type is synthesized based on the column of the SQL data type or from the database system.

The SQL data type information in the USE AS clause identifies the value in the SIGNED column in the following instances:

- One character code is supplied.
- No DATATYPE information is specified.
- Native data type information is not obtained from the database system.

USE AS

Identifies the SQL data type for the column.

The following table describes the data types for columns. The non-null SQLTYPE identifies the data type in internal control blocks and diagnostic trace information.

Table 81. SQL data type descriptions

Keyword identifier	Description	Maximum length	SQLTYPE
CHAR(length)	Fixed-length character string that contains mixed mode data.	254	452
VARCHAR(length)	Variable length character string that contains mixed mode data. A half-word length component precedes the character string and identifies the actual length of the data. The VARCHAR length does not include the length of the LENGTH field.	32704	448
LONG VARCHAR	Long character string that contains mixed mode data. A half-word length component precedes the character string and identifies the actual length of the data. The LONG VARCHAR length does not include the length of the LENGTH field.	32704	456
GRAPHIC(length)	Fixed-length graphic string that is assumed to contain pure DBCS data without shift codes. The length is expressed in DBCS characters, and not bytes.	127	468
VARGRAPHIC(length)	Varying-length graphic string that is assumed to contain pure DBCS data without shift codes. A half-word length component precedes the graphic string and identifies the actual length of the data. The length is expressed in DBCS characters, and not bytes.	16352	464
LONG VARGRAPHIC	Long graphic string that is assumed to contain pure DBCS data without shift codes. A half-word length component precedes the graphic string and identifies the actual length of the data. The length is expressed in DBCS characters, and not bytes.	16352	472
INTEGER	Large integer.	Exactly 4	496
SMALLINT	Small integer.	Exactly 2	500
DECIMAL (precision,scale) or DECIMAL(precision)	Packed decimal data. A valid precision value is between 1 and 31. A valid scale value is between zero and the precision value.	31	484
FLOAT(precision)	Floating point number. Depending upon the precision, a column with a FLOAT data type takes on the attributes of a REAL or DOUBLE PRECISION data type. When precision is in the range of 1 to 21, the column is treated as a REAL. For precisions between 22 and 53, the column takes on DOUBLE PRECISION attributes. A precision of zero or greater than 53 is nonvalid.	4 or 8	480

Table 81. SQL data type descriptions (continued)

Keyword identifier	Description	Maximum length	SQLTYPE
REAL	A single-precision, floating-point number that is a 32-bit approximation of a real number. The number can be zero or can range from -3.40282347E+38 to -1.17549435E-38, or from 1.17549435E-38 to 3.40282347E+38.	4	480
DOUBLE PRECISION	A floating-point number that is a 64-bit approximation of a real number. The number can be zero or can range from -1.797693134862315E+308 to -2.225073858507201E-308, or from 2.225073858507201E-308 to 1.797693134862315E+308.	8	480
BINARY(length)	Fixed-length binary data.	32704	912
VARBINARY(length)	Variable length binary data, where the actual data is preceded by a 16-bit signed binary number that identifies the actual length of the data. The VARBINARY length does not include the length of the LENGTH filed.	32704	908

When you supply a USE AS LONG VARCHAR or USE AS LONG VARGRAPHIC clause, you do not specify a length for the column. The length is based on the physical attributes that are obtained about the object record or segment where the column is.

When you supply a USE AS VARCHAR clause with a length greater than 254, that column is changed into a LONG VARCHAR column with the specified length. The same behavior occurs for a USE AS VARGRAPHIC clause when the length is greater than 127.

USE RECORD LENGTH

The USE RECORD LENGTH clause is specified after the USE AS clause for a VARCHAR, LONG VARCHAR, VARGRAPHIC or LONG VARGRAPHIC data type specification.

Indicates the varying string or graphic column contains the entire contents of the record that the column is associated with. The entire record content contains character data. The actual data contents were not important to any client application that accessed one of these columns that ran on an MVS system where no code page conversion was performed.

WITHOUT CONVERSION

Specifies that a field procedure does not exist for the column.

WITH CONVERSION

Identifies the name of the load module for a field procedure, if a field procedure is required. The field procedure name is a short identifier.

NULL IS *null_value*

Specifies a value that identifies when the contents of the column is null. By default, the data in a column never contains a null value. This situation is true even for variable length character or graphic columns where the length component indicates that there is no data in the variable length column.

A null value is a character string that can specify up to 16 characters of data. Specifies the value in hexadecimal format. The total length of the string is 35 characters.

A column contains null values based on *null-value*. If the start of the column data matches the null value, then that instance of the column is null. For example, if a column is defined as CHAR(10) and a null value of x'4040' is specified, the null value length is 2. If the first 2 bytes of the column contain spaces, the column is also null.

PRIMARY KEY

Identifies the column to be one of the columns that constitute the primary key for the table. This form of primary key identification is mutually exclusive with specifying a primary key at the end of the CREATE TABLE statement.

You can identify a primary key at the column level when the columns that make up the primary key for the table are defined in ordinal sequence within the table definition.

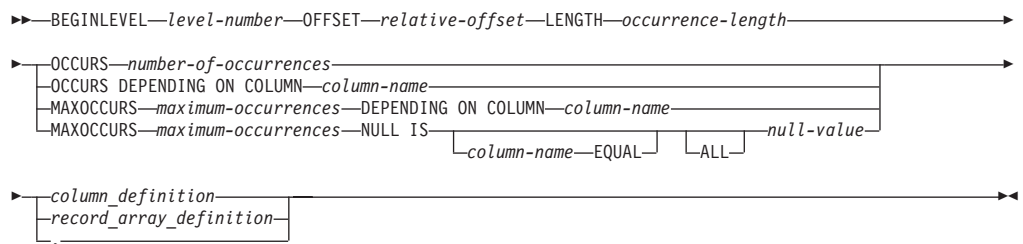
KSDS or files that are accessed through an alternate index of columns that map to the key offset and length information, which are determined during validation, are good candidate columns to identify as primary keys. VSAM ESDS or RRDS files that are not accessed through an alternate index are always poor candidates for primary keys.

You can use the PRIMARY KEY clause on the column definition to identify the primary key columns when the keys are defined in the same order.

Record arrays for VSAM

Record arrays identify and handle data that can repeat multiple times within a non-DB2 database or file system.

Syntax



Parameters

BEGINLEVEL *level-number*

Identifies the start of a record array definition.

The *level-number* on a BEGINLEVEL or ENDDLEVEL definition identifies the nesting level of the record array within the parent fragment. Multiple record arrays can exist with the same *level-number*. Typically, you do not map multiple record arrays within a single table. Create separate tables for each record array on the same level to avoid result sets that are too large.

The following example shows a multiple record array. In this example, the EMPL-ADDRESS data item occurs three times and the EMPL-DEPENDENTS, EMPL-DEP-NAME and EMPL-DEP-DOB data items occur ten times. If you map both arrays to the same table, the query processor generates 30 rows for each EMPL-RECORD read. Each instance of EMPL-ADDRESS is combined

with each instance of EMPL-DEPENDENTS to create 30 rows (3 addresses times 10 dependents).

```
01 EMPL-RECORD.  
  05 EMPL-SSN          PIC 9(9).  
  05 EMPL_NAME  
    10 EMPL-LAST      PIC X(30).  
    10 EMPL-FIRST    PIC X(30).  
    10 EMPL_MI       PIC X.  
  05 EMPL-ADDRESS     PIC X(30) OCCURS 3 TIMES.  
  05 EMPL-DEPENDENTS OCCURS 10 TIMES.  
    10 EMPL-DEP-NAME PIC X(30).  
    10 EMPL-DEP-DOB  PIC 9(8).
```

If you must access the information in the above example in a single table definition, then two record array definitions are required. These record arrays are both level 1 arrays, because they are not nested within other arrays. To avoid performance problems related to result sets that are too large, flatten the structure or define a separate table for one or both arrays.

OFFSET *relative-offset*

The relative offset identifies the relative starting position of the record array within its owning fragment definition. The owning fragment definition is either associated with the table or with a parent record array definition.

LENGTH *occurrence-length*

Identifies the length in bytes of one occurrence of the repeating data.

MAXOCCURS *maximum-occurrences* **DEPENDING ON COLUMN** *column-name*

Identifies a record array that occurs a variable number of times. A column definition that appears before the record array definition determines the number of instances.

You cannot map columns that occur after a variable length array, because the starting offsets of data items that are defined after an OCCURS DEPENDING ON definition are not at a predictable offset.

The *maximum-occurrences* value must be numeric, and *column-name* must specify the name of a column in the table. The control column must contain a numeric value. The query processor supports SQL types of SMALLINT, INTEGER, DECIMAL, or CHAR (if the column contains zoned decimal data).

MAXOCCURS *maximum-occurrences* **NULL IS**

Defines a record array that repeats a fixed number of times. If the initial bytes of the first column in the record array match *null-value*, that instance of the array is null.

The *maximum-occurrences* value must be numeric. The null value can be a character string of up to 32 characters. The null-value can also be a 67-character hexadecimal string that consists of 64 hexadecimal nibbles, with a leading x and surrounding quotation marks, for example, x'4040'.

ENDLEVEL *level-number*

Identifies the end of a record array definition.

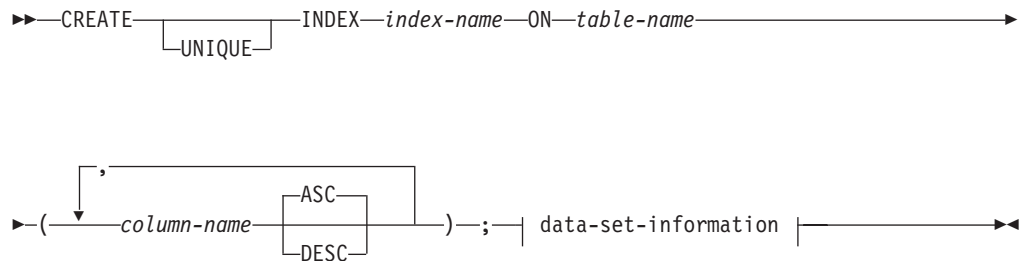
CREATE INDEX statement for VSAM

You can use the CREATE INDEX statement to define an index that references the columns that make up a VSAM KSDS primary key or the columns that map to an alternate index PATH definition.

Indexes identify columns in a table that correspond to a physical index that is in the source database. The query processor uses the contents of the columns that are referenced in a WHERE clause to create the index. The query processor attempts to build either a full key value to use in database access or a partial key value. The partial key can be used to perform a range scan against the target database to reduce the number of records that are accessed.

Although you might not be able to define an index against columns that correspond to the primary key, you can identify these columns as primary key columns. This primary key information does not affect existing connector index selection and access optimization. The primary key information is made available for use by front-end tools, where key information is either required or beneficial.

Syntax



data-set-information::



Parameters

`CREATE INDEX index-name`

Identifies the SQL statement as an index definition statement. A unique index does not have any restrictions about the columns that make up the index. For example, a unique index column can contain null values.

If qualified, the index name is a two-part name, and the authorization ID that qualifies the name is the owner of the index. If an unqualified table name is supplied, the owner name is the authorization ID from the CURRENT SQLID special register.

`ON table-name`

Identifies the table for which the index is being defined. The table name can be a qualified or unqualified table name. For unqualified names, the table owner is from the CURRENT SQLID special register.

The table name is validated with the standard syntax checks that are associated with identifiers, and then the existence of the table is verified. Do not define an index on a view.

`column-name ASC/DESC`

Specifies column names that make up the index key. Optionally, you can identify whether the column is stored or accessed in ascending (ASC) or descending (DESC) key sequence. By default, the key is in ascending key sequence.

There are no fixed limits on the number of columns that you can identify as key columns for an index. The only requirements are as follows:

- The column must exist in the table.
- The column must map to what constitutes a key in the target database. In most cases, this determination is based on the starting offset and length of the column as compared to the starting offset and length of the key in the target database.
- The column cannot overlap another column within the key definition. This determination is based on the starting offset and length of a column as compared to the starting offsets and length of the other columns that are identified as key columns for the index definition.
- For the key column, generally varying length and graphic data types are not supported.

Key column verification is strict and enforces all rules for column names. Rule 4 enforcement (nonvalid data types) is enforced. Identification of any kind of varying-length character or graphic string is not allowed.

Data-set-information

Identifies the data set that accesses the columns that make up the table that *table_name* references. One common reason to create an index is when a VSAM KSDS file has an alternate index that is associated with it. The most likely scenario is that the table references the base cluster name, and some queries do not contain references in the WHERE clause to the VSAM primary key. But, the queries do contain columns in the WHERE clause that map to the key of the alternate index. For these queries, use the alternate index.

To define an alternate index, use either the DD clause or DS clause to identify the name of the alternate index path data set. Also, identify as the key columns those columns that map to the alternate index key. When a query that contains a WHERE clause references these key columns, automatic index selection determines that the index is the best candidate for processing the query. Thus, the alternate index path accesses the actual VSAM data.

You can also define indexes to publish information to tools. For example, if you have a single column that maps the entire social security number and the three-column mapping for the components, you can define two index definitions. In the first index definition, identify the single column as the key column. In the second index definition, identify the three-component columns as the keys. In this example, identification of DD or DS information is not allowed. The DD or DS clauses only identify an alternate index path data set.

DD *DD-name*

Specifies the VSAM file that the table maps to. Specifies either a local file reference or a reference to a CICS file definition table (FDT) entry name.

If *DD-name* represents a local file reference, a DD clause that references the VSAM file must exist in the Classic federation data server JCL.

If *DD-name* references an FDT entry name, you must specify CICS connection information with the `CICS_connection_information` clause.

DD-name must correspond to a cluster or alternate index path definition for a VSAM ESDS, KSDS or RRDS data set.

DD-name is a short native identifier that conforms to the naming conventions for a z/OS JCL DD statement.

DS *data-set-name*

Specifies the VSAM file that the table definition accesses. Designates a VSAM cluster definition or a PATH component when the VSAM file is accessed from a VSAM alternate index.

The data set name must correspond to a cluster or alternate index path definition for a VSAM ESDS, KSDS or RRDS data set.

The name can be 1 to 44 characters in length and follows z/OS naming conventions for VSAM data sets.

Example

The following is an example of a CREATE INDEX statement for VSAM.

```
CREATE UNIQUE INDEX "DBA"."EMPLOYEE_EP_IDX1"  
ON "DBA"."EMPLOYEE_EP" ("EMPL_SSN" ASC);
```

SQL security

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

Overview of SQL security

SQL security in Classic federation is similar to security in DB2 databases.

Implementing security and enforcing security restrictions are the responsibility of the DBMS software. SQL defines an overall framework for database security, and SQL statements specify security access and restrictions.

SQL security involves the following key concepts:

- Users are the actors in the database. When the DBMS retrieves, inserts, deletes, or updates data, it does so on behalf of a user or group of users. The DBMS permits or denies user actions depending on which user makes the request. You can define users and user groups based on categories of administrative authorities.
- Database objects, such as tables, views, and stored procedures are the objects to which SQL security can be applied.
- Privileges are the actions that a user is permitted to perform against a particular database object. For example, a user might have permission to select and insert rows in one table, but lack permission to delete or update rows in that table. These privileges are allowed or prohibited by using the GRANT and REVOKE SQL statements.
- SQL security and the SAF exit work together to ensure that the user ID and its password are checked before allowing access to particular database objects.
- SQL security is required.

Authorization

In Classic federation, each user ID is associated with a particular authority level.

You can assign users to the following authority levels:

SYSADM

The system administrator has privileges for all objects and has the ability to grant authority to other users. The first user to run the metadata utility is granted SYSADM authority.

SYSOPR

The system operator has remote-operator privileges to display and manage an active data server.

DISPLAY

This user or user group has remote-operator privileges for display commands on an active data server.

DBADM

The database administrator has mapping and view-creation privileges for specific database types.

PUBLIC

This user or user group is limited to privileges that are explicitly granted to its user name or PUBLIC.

You assign privileges to individual users or groups of users based on an authorization ID by using the SQL GRANT statement. The authorization ID determines whether the statement is permitted or prohibited by the DBMS. In production databases, the database administrator assigns authorization IDs.

Authorization requirements for SQL statements

To issue GRANT, REVOKE, SELECT, EXECUTE, INSERT, UPDATE, and DELETE statements, you must have specific authorization.

The following table describes authority levels that are required to issue each of these statements.

Table 82. Authorization requirements for SQL statements

Statement	Authority required
{GRANT REVOKE}	To grant a privilege, you must have SYSADM authority, or you must be granted the privilege itself with the WITH GRANT option. To revoke a privilege, you must have SYSADM authority or be the user who originally granted the privilege being revoked. The BY ALL clause requires SYSADM authority because you are revoking revoke privileges that are granted by users other than yourself.
SELECT	SYSADM authority or the specific privilege is required. SELECT authority on all tables and views is referenced in the SELECT statement.
EXECUTE	SYSADM authority or the specific privilege is required. EXECUTE authority is on the procedure.
INSERT	SYSADM authority or the specific privilege is required. INSERT authority is on the table.
UPDATE	SYSADM authority or the specific privilege is required. UPDATE authority is on the table and SELECT authority on all tables is referenced in the WHERE clause.

Table 82. Authorization requirements for SQL statements (continued)

Statement	Authority required
DELETE	SYSADM authority or the specific privilege is required. DELETE authority is on the table. SELECT authority is on all tables that are referenced in the WHERE clause.

Database objects in SQL security

Catalog database types are database objects to which security can be applied.

To manage or secure the metadata utility for mapping purposes, the following implicit database names are associated with metadata catalogs:

\$ADABAS

For Adabas database mappings

\$CFI For system catalog

\$DATACOM

For CA-Datcom database mappings

\$IDMS

For CA-IDMS database mappings

\$IMS For IMS database mappings

\$SEQUENT

For sequential database mappings

\$SP For stored procedure definitions

\$VSAM

For VSAM database mappings

These metadata catalogs map one-to-one with the connectors, with the exception of \$CFI.

You can specify database types in the GRANT DBADM statement. For example:

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

To map tables, you must have the SYSADM authority to run the metadata utility, because the metadata utility does not have security-access checking at the database level. Grant DBADM authority only for DROP commands.

Defining user privileges

Privileges are the set of actions that a user can perform. The SQL GRANT and REVOKE statements assign privileges.

You can use the Classic Data Architect to grant and revoke one or more of the following privileges:

- System
- Database
- Stored procedures
- Tables and views

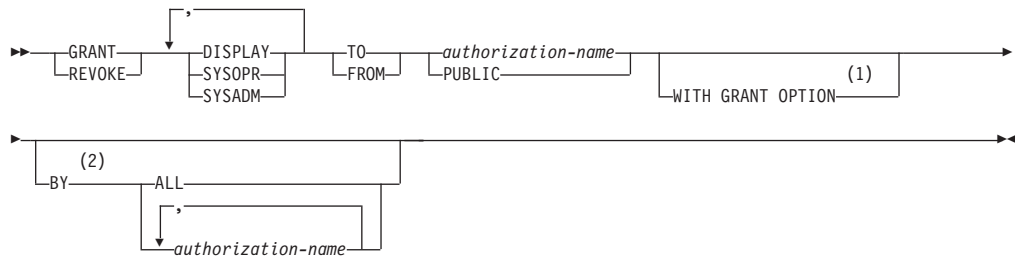
The GRANT and REVOKE are executable statements that can be dynamically prepared.

A specific grantor can grant or revoke specific privileges to specific users, with the restriction that a privilege can only be revoked if it has first been granted.

System privileges:

System privileges allow or deny access to a specific set of catalogs within a data server.

Syntax:



Notes:

- 1 GRANT only
- 2 REVOKE only. Only revokes privileges granted by that user

The following table describes the statement parameters.

Table 83. Parameter descriptions for the GRANT and REVOKE statement.

Parameter	Description
{GRANT REVOKE}	GRANT or REVOKE privileges to user IDs or groups of user IDs.
DISPLAY	GRANT or REVOKE the privileges to remotely issue all forms of the DISPLAY command to a data server.
SYSOPR	GRANT or REVOKE the privilege to remotely issue all commands to a data server including the commands to start and stop services and shut down the data server. Commands issued from the system console are not secured.
SYSADM	GRANT or REVOKE system administrator authority.
{TO FROM} {authorization-name PUBLIC}	GRANT or REVOKE authority to a particular user or group of users, or to all users or groups of users on a system.
WITH GRANT OPTION	GRANT authority to a particular user or group of users to GRANT authority to other users or other groups of users in the system. Although this option can be specified when granting the SYSADM privilege, it has no effect on SYSADM privileges because SYSADM has ALL access privileges available in the data server.

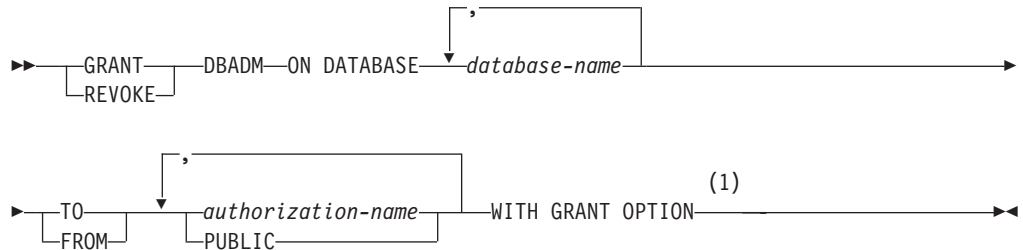
Table 83. Parameter descriptions for the GRANT and REVOKE statement. (continued)

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

Database privileges:

Authorization IDs with DBADM privileges can grant and revoke specific privileges within a particular database.

Syntax:



Notes:

1 GRANT only

The following table describes the statement parameters.

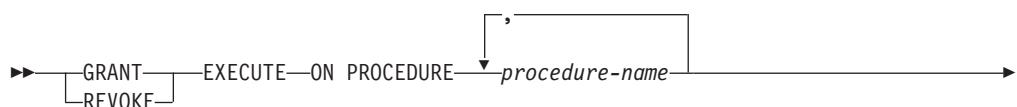
Table 84. Parameter descriptions for the GRANT and REVOKE statement.

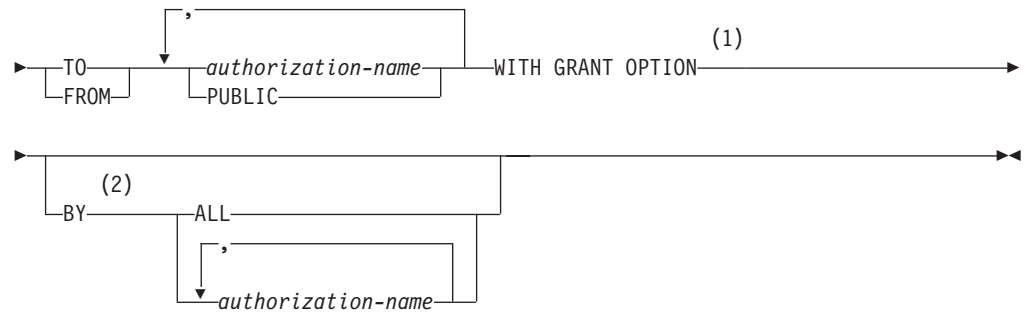
Parameter	Description
GRANT ON DATABASE <i>database-name</i>	<p>Identifies database types on which privileges are to be granted. For each named database type the grantor must have all the specified privileges with the GRANT option. This privilege secures the mappings of tables and dropping of mapped tables. The types are as follows:</p> <p>\$ADABAS For Adabas mappings</p> <p>\$CFI For metadata catalog mappings</p> <p>\$DATACOM For CA-Datcom mappings</p> <p>\$IDMS For CA-IDMS mappings</p> <p>\$IMS For IMS mappings</p> <p>\$SEQUENT For sequential</p> <p>\$SP for stored procedures</p> <p>\$VSAM For VSAM</p>
REVOKE ON DATABASE <i>database-name</i>	<p>Identifies the database type on which you revoke the privileges. For each database type, you or the indicated grantors must have granted at least one of the specified privileges on that database to all identified users (including PUBLIC, if specified). The same database type must not be identified more than once. The database-names are the same as those listed for the GRANT ON DATABASE <i>database-name</i> option.</p>
{TO FROM} { <i>authorization-name</i> PUBLIC}	<p>Specifies to what authorization IDs the privileges are granted or revoked. The <i>authorization-name</i> variable lists one or more authorization IDs.</p>
WITH GRANT OPTION	<p>Allows the named users to grant the database privileges to others. The user can specifically grant any privilege belonging to that authority. If you omit WITH GRANT OPTION, the named users cannot grant the privileges to others unless they have that authority from some other source.</p>

Stored procedure privileges:

Stored procedure privileges allow or deny access to a particular stored procedure.

Syntax:





Notes:

- 1 GRANT only
- 2 REVOKE only. Only revokes privileges granted by that user

The following table describes the statement parameters.

Table 85. Parameter descriptions for the GRANT and REVOKE statement.

Parameter	Description
{GRANT REVOKE}	Grants or revokes authority to run a stored procedure.
ON PROCEDURE <i>procedure-name</i>	Identifies the procedure for which you grant or revoke privileges.
{TO FROM} { <i>authorization-name</i> PUBLIC}	Specifies to or from which authorization IDs the privileges are granted or revoked. The <i>authorization-name</i> variable lists one or more authorization IDs.
WITH GRANT OPTION	Allows the named users to grant the stored procedure privileges to others. The user can specifically grant any privilege belonging to that authority. If you omit WITH GRANT OPTION, the named users cannot grant the privileges to others unless they have that authority from some other source.
BY ALL <i>authorization-name</i>	<p>BY revokes each named privilege that was explicitly granted to some named user or group of users by one of the named grantors. Only an authorization ID with SYSADM authority can use BY, even if the authorization ID names only itself in the BY clause.</p> <p>ALL then revokes each named privilege from all named users or group of users. <i>authorization-name</i> lists one or more authorization IDs of users or group of users who were the grantors of the privileges named.</p> <p>Do not use the same authorization ID more than once. Each grantor listed must have explicitly granted some named privilege to all named users or group of users.</p>

Table and view privileges:

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

GRANT syntax for tables and views

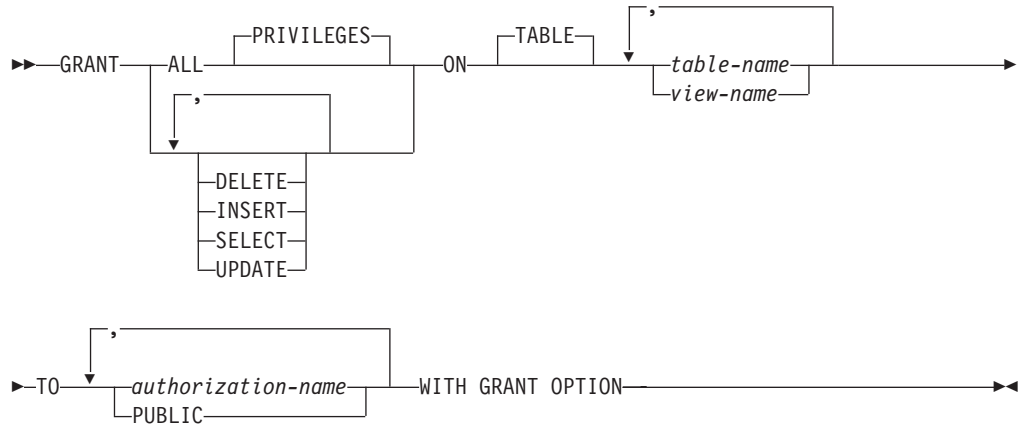


Table 86. Parameter descriptions for the GRANT statement.

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

REVOKE syntax for tables and views

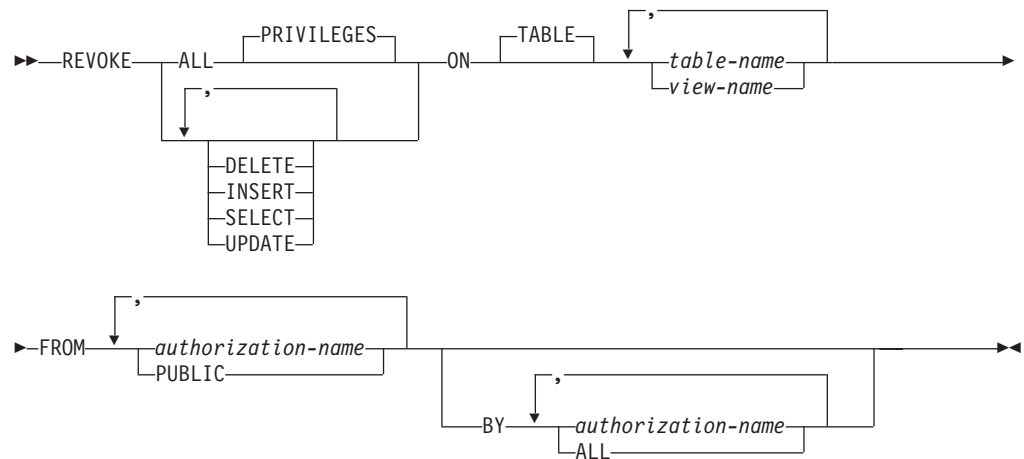


Table 87. Parameter descriptions for the REVOKE statement.

Statement	Description
REVOKE {ALL ...} PRIVILEGES	Revokes all table or view privileges for which you have GRANT authority, for the tables and views named in the ON clause.
DELETE	Revokes privileges to use the DELETE statement.
INSERT	Revokes privileges to use the INSERT statement.
SELECT	Revokes privileges to use the SELECT statement.
UPDATE	Revokes privileges to use the UPDATE statement.
ON TABLE {table-name view-name}	Names the tables or views on which you are granting the privileges. The list can be a list of table names or view names, or a combination of the two.
FROM {authorization-name PUBLIC}	Specifies to what authorization IDs the privileges are revoked. <i>authorization-name</i> lists one or more authorization IDs.

Table 87. Parameter descriptions for the REVOKE statement. (continued)

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

SAF and SMF system exits for SQL security

In addition to authorizations and privileges for SQL security, you need to use the System Authorization Facility (SAF) security exit. The SAF exit is required to validate passwords for data server connections.

With the SAF exit, you can administer SQL security at a group level. The group name, in addition to the user ID, determines whether users are authorized to perform the operation they attempt. A group name makes security administration easier because you only need to grant and revoke privileges to and from the group name for all users that are associated with that group name. When you need to authorize access to new users, you can use your external security package and assign the new user to the default group name.

In addition, the query processor calls the System Management Facility (SMF) exit when an authorization violation is detected. The SMF exit generates an SMF record that logs the user ID, type of authorization failure, and the name of the object for which authorization failed.

Views

In Classic federation, views can provide alternative ways to work with the data.

Database tables define the structure and organization of the data it contains. Using SQL, you can look at the stored data in other ways by defining alternative views of the data. A *view* is an SQL query that is stored in the database and assigned a name, similar to a table name. The results of the stored query are then visible through the view. With SQL, you can access these query results as if the results were a real table in the database.

Views allow you to manage your data in these ways:

- Tailor the appearance of a database so that different users see it from different perspectives.
- Restrict access to data, allowing different users to see only certain rows or certain columns of a table.

- Simplify database access by presenting the structure of the stored data in the way that is most natural for each user.

Record types in data definition examples

Many data definitions contain *redefined data* that contains alternate layout information to map different record types within a single physical file.

To enable applications to read redefined data, create a separate base table and associated view for each record type. You can use the record type field in the view definition to filter the data.

To insert, update, or delete redefined data, client applications must use the base table name and filter the records by specifying the record type field in a WHERE clause.

For example, a single VSAM file stores both employee and address records. The correct record interpretation is managed by associating the value of a record type field with a record layout. The following COBOL definition shows how a REDEFINES clause specifies an alternate record type for ADDRESS-INFORMATION. If RECORD-TYPE = "A" Classic Data Architect uses the layout for address data.

```

01 EMPLOYEE-ADDRESS-RECORD.
   05 EMP-ID          PIC X(6).
   05 RECORD-TYPE    PIC X.
      88 RECORD-IS-EMPLOYEE VALUE 'E'.
      88 RECORD-IS-ADDRESS VALUE 'A'.
   05 EMPLOYEE-INFORMATION.
      10 LAST-NAME    PIC X(20).
      10 FIRST-NAME   PIC X(20).
      10 DATE-OF-BIRTH PIC 9(8).
      10 MONTHLY-SALARY PIC S9(5)V99 COMP-3.
      10 FILLER       PIC X(48).
   05 ADDRESS-INFORMATION REDEFINES EMPLOYEE-INFORMATION.
      10 ADDRESS-LINE-1 PIC X(30).
      10 ADDRESS-LINE-2 PIC X(30).
      10 ADDRESS-CITY   PIC X(20).
      10 ADDRESS-STATE  PIC XX.
      10 ADDRESS-ZIP    PIC 9(5).

```

A COBOL REDEFINES clause redefines data in the identical record locations of a record layout. In this case, EMPLOYEE-INFORMATION and ADDRESS-INFORMATION both start at location 6 in the record. To accurately map the COBOL record above, two distinct SQL mappings are necessary for the file. The first mapping consists of columns for EMP-ID, RECORD-TYPE, LAST-NAME, FIRST-NAME, DATE-OF-BIRTH, and MONTHLY-SALARY. The second mapping consists of EMP-ID, RECORD-TYPE, ADDRESS-LINE-1, ADDRESS-LINE-2, ADDRESS-CITY, ADDRESS-STATE, and ADDRESS-ZIP. Each of these mappings is used to create a separate table definition in the metadata catalog.

By default, in response to an SQL query or data change, every record in the file matches the defined table. If the underlying data does not match the definition, errors in processing occur or nonvalid information is returned. In the case described above, an address record retrieved by using the employee table definition produces a data error because the MONTHLY-SALARY field contains character address data instead of numeric salary data.

To process record instances accurately with different layouts, you must apply record selection criteria to the underlying data. If the selection criteria do not match, records are skipped because they do not apply to the defined mapping.

Example: The following CREATE TABLE statements define the EMPLOYEE and ADDRESS layouts above:

```
CREATE TABLE VSAM.EMPLOYEE_NAME DBTYPE VSAM
DS "VSAMEMP.KSDS"
(
    "EMP_ID" SOURCE DEFINITION
    DATAMAP OFFSET 0 LENGTH 6
    DATATYPE C
    USE AS CHAR(6),
    "RECORD_TYPE" SOURCE DEFINITION
    DATAMAP OFFSET 6 LENGTH 1
    DATATYPE C
    USE AS CHAR(1),
    "LAST_NAME" SOURCE DEFINITION
    DATAMAP OFFSET 7 LENGTH 20
    DATATYPE C
    USE AS CHAR(20),
    "FIRST_NAME" SOURCE DEFINITION
    DATAMAP OFFSET 27 LENGTH 20
    DATATYPE C
    USE AS CHAR(20),
    "DATE_OF_BIRTH" SOURCE DEFINITION
    DATAMAP OFFSET 47 LENGTH 8
    DATATYPE UC
    USE AS CHAR(8),
    "MONTHLY_SALARY" SOURCE DEFINITION
    DATAMAP OFFSET 55 LENGTH 4
    DATATYPE P
    USE AS DECIMAL(7 , 2) );

CREATE TABLE VSAM.EMPLOYEE_ADDRESS DBTYPE VSAM
DS "VSAMEMP.KSDS"
(
    "EMP_ID" SOURCE DEFINITION
    DATAMAP OFFSET 0 LENGTH 6
    DATATYPE C
    USE AS CHAR(6),
    "RECORD_TYPE" SOURCE DEFINITION
    DATAMAP OFFSET 6 LENGTH 1
    DATATYPE C
    USE AS CHAR(1),
    "ADDRESS_LINE_1" SOURCE DEFINITION
    DATAMAP OFFSET 7 LENGTH 30
    DATATYPE C
    USE AS CHAR(30),
    "ADDRESS_LINE_2" SOURCE DEFINITION
    DATAMAP OFFSET 37 LENGTH 30
    DATATYPE C
    USE AS CHAR(30),
    "ADDRESS_CITY" SOURCE DEFINITION
    DATAMAP OFFSET 67 LENGTH 20
    DATATYPE C
    USE AS CHAR(20),
    "ADDRESS_STATE" SOURCE DEFINITION
    DATAMAP OFFSET 87 LENGTH 2
    DATATYPE C
    USE AS CHAR(2),
    "ADDRESS_ZIP" SOURCE DEFINITION
    DATAMAP OFFSET 89 LENGTH 5
    DATATYPE UC
    USE AS CHAR(5) );
```

Given the example, there are two choices:

Create two tables, each table mapping a different record layout

Inserts, updates, and deletes that are issued against a base table by client applications must contain a WHERE clause that selects the correct record type.

For example, you can use an UPDATE statement similar to the following:

```
UPDATE EMPLOYEE_ADDRESS SET ADDRESS_LINE_1='145 MAIN ST APT B'  
WHERE EMP_ID= '130001' and RECORD_TYPE= 'A';
```

Create two tables and create a view on each table

Query the view associated with a base table to read the data. This approach simplifies queries by eliminating the need for WHERE clause filtering. The following view creation statements apply selection logic for the table mappings:

```
CREATE VIEW VEMPL_NAME AS  
SELECT * FROM EMPLOYEE_NAME WHERE RECORD_TYPE = 'E';  
CREATE VIEW VEMPL_ADDRESS AS  
SELECT * FROM EMPLOYEE_ADDRESS WHERE RECORD_TYPE = 'A';
```

You can then use a SELECT statement like this:

```
SELECT * FROM VEMPL_NAME;
```

Views and the query processor in Classic federation

When the query processor encounters a reference to a view in an SQL statement, it finds the definition of the view in the database.

Then the query processor translates the request that references the view into an equivalent request against the source tables of the view and carries out that request. The query processor maintains the illusion of the view while the query processor maintains the integrity of the source tables.

Advantages and disadvantages of views in Classic federation

Views provide a variety of benefits and can be useful for many types of databases.

In a personal computer database, views are usually a convenience, defined to simplify requests to databases. In a production database installation, views can play an important role in defining the structure of the database for users or groups of users and can enforce the database security.

Views provide the following benefits:

- **Built-in security:** Gives each user permission to access the database only through a small set of views that contain the specific data the user or group of users is authorized to see, restricting user access to other data.
- **Simplicity for queries:** A view can draw data from several tables and present a single table, simplifying the information and turning multi-table queries into single-table queries for a view.
- **Simplicity in structure:** Views give users a specific view of the database structure, presenting the database as a set of virtual tables specific to particular users or groups of users.
- **Stabilization of information:** Views present a consistent, unchanged image of the database structure, even if underlying source tables are changed.

Although there are many advantages to views, the main disadvantage to using views rather than real tables is performance degradation. Because views only create the appearance of a table, not a real table, the query processor must translate queries against the view into queries against the underlying source tables. If the

view is defined by a complex, multi-table query, even simple queries against the view become complicated joins that can take a long time to complete.

Joined views in Classic federation

One of the most frequent reasons for using views is to simplify multi-table queries.

By specifying a two-table or a three-table query in the view definition, you can create a joined view. Joined views draw their data from two or three different tables and present the query results as a single virtual table. After you define the view, you can use a simple, single-table query against the view for requests that otherwise require a two-or-more-table join.

Example

A user often runs queries against a particular table, such as the ORDERS table. The user does not want to work with employee numbers, but wants a view of the ORDERS table that has names instead of numbers. You can create the following view:

```
CREATE VIEW ORDER_INFO (ORDER_NUM, COMPANY, REP_NAME, AMOUNT) AS
  SELECT ORDER_NUM, COMPANY, NAME, AMOUNT
  FROM ORDERS, CUSTOMERS, SALESREPS
  WHERE CUST = CUST_NUM
  AND REP = EMPL_NUM
```

This view is defined by a three-table join. As with a grouped view, processing required to create this virtual table is substantial. Each row of the view is created from a combination of one row from the ORDERS table, one row from the CUSTOMERS table, and one row from the SALESREPS table.

Although this view has a complex definition, it can be very valuable. For example, you can create the following query against this view:

```
SELECT REP_NAME, COMPANY SUM(AMOUNT)
  FROM ORDER_INFO
  GROUP BY REP_NAME, COMPANY
```

That query generates a report of orders that are grouped by salesperson:

REP_NAME	COMPANY	SUM(AMOUNT)
Bill Adams	ACME Mfg.	\$35,582.00
Bob Burns	JCP Inc.	\$24,343.00
Dan Jones	First Corp.	\$75,000.00

This query is now a single-table SELECT statement, which is far simpler than the original three-table query. Also, the view makes it easier to see the operations in the query. The query processor, however, still must work harder to generate the query results for this single-table query against the view as it would to generate query results for the same three-table query. However, for the actual user, it is much easier to write and understand a single-table query that references the view.

CREATE VIEW statement

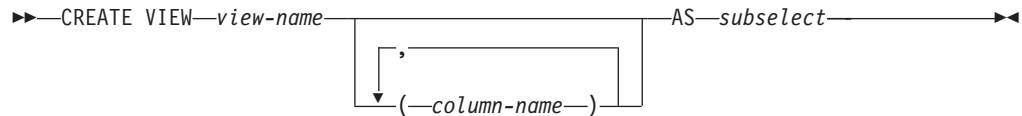
You can create and manage DB2 views by using the Classic Data Architect, the metadata utility, or any client that is connected to the data server.

Although most clients can create a view with standard SQL processing, the Classic Data Architect is a more controlled mechanism for creating and managing views. For this reason, use the Classic Data Architect to create views.

The data type, length, and other characteristics of the columns are derived from the definition of the columns in the source tables.

The CREATE VIEW statement can be embedded in an application program or issued interactively. The statement is an executable statement that can be dynamically prepared.

Syntax diagram



Parameters

view-name

Assigns a name to the view. The name cannot identify a table, view, alias, or synonym that exists on the current server. The name can be a two-part name. The authorization name that qualifies the name is the owner of the view.

column-name

Names the columns in the view. If you specify a list of column names, the list must consist of as many names as there are columns in the result table of the subselect. Each name must be unique and unqualified. If you do not specify a list of column names, the columns of the view inherit the names of the columns of the result table of the subselect.

You must specify a list of column names if the result table of the subselect has duplicate column names or an unnamed column (a column that is derived from a constant, function, or expression that is not given a name by the AS clause).

AS *subselect*

Defines the view. At any time, the view consists of the rows that result if the subselect is run. The subselect cannot refer to host variables or include parameter markers (question marks).

A query that contains either a UNION or an ORDER BY clause is not a valid subselect.

DROP VIEW statement

To drop a view, you must use the DROP VIEW statement.

This statement provides detailed control over what happens when a user attempts to drop a view when the definition of another view depends on it.

Example: Two views on the SALESREPS table are created by these CREATE VIEW statements:

```
CREATE VIEW EASTREPS AS
  SELECT *
  FROM SALESREPS
  WHERE REP_OFFICE IN (11, 12, 13)
CREATE VIEW NYREPS AS
  SELECT *
  FROM EASTREPS
  WHERE REP_OFFICE = 11
```

The following DROP VIEW statement removes both views as well as any views that depend on their definition from the database:

```
DROP VIEW EASTREPS
```

The CASCADE and RESTRICT parameters are not directly supported in the DROP VIEW syntax. However, the DROP VIEW deletes dependent views along with those specified in the DROP VIEW.

Restrictions on binary data in SQL operations

When you reference binary data in SQL operations, some restrictions apply to binary data.

The following restrictions apply to using binary data in SQL operations:

- The maximum length allowed for large BINARY and VARBINARY values is 254 bytes in certain SQL statements that require the query processor to perform additional evaluation or manipulation of the value:
 - Concatenation expression or function
 - GROUP BY clause
 - HAVING clause
 - ORDER BY clause
 - WHERE clause
 - SELECT DISTINCT statement

More specifically, this means that processing will fail if the following limitations are exceeded:

- For concatenations of host variables, constants, or column list items, the resulting concatenated value cannot be larger than 254 bytes.
- For the other constructs listed above, the column size definition for a referenced BINARY or VARBINARY column cannot be larger than 254 bytes.

Values for statements that do not require additional evaluation or manipulation can always be as large as the defined column size without restriction. For example, this applies to binding input host variables for an insert statement.

- The maximum length allowed for binary constants is 254 hexadecimal characters, thus limiting the size of the actual binary value of a constant to 127 bytes.
- You cannot create an index over BINARY or VARBINARY columns.
- The LIKE predicates is not supported for binary strings.
- BINARY and VARBINARY data types are not supported for stored procedures or field procedures.
- Referencing BINARY and VARBINARY columns in scalar functions is not supported.

Programming reference for the JDBC driver

With the JDBC driver, you can use interfaces to manage connection pooling and an interface for distributed transactions. You define configuration parameters in the java.sql.properties file.

ConnectionPool interface

Use the ConnectionPoolDataSource class when you want Classic federation to manage connection pooling for you.

If you want to manage connection pooling by some other means or do not want to use connection pooling, and client applications do not perform distributed transactions, use a `DataSource` object instead. If you want Classic federation to manage connection pooling for you and client applications perform distributed transactions, use a `XADataSource` object instead.

Methods

getDatabaseName

Input parameters: None

Return type: `java.lang.String`

Description: Returns the name of the database. Returns null if the database name is not set.

The database name corresponds to the `DATASOURCE` name which is the query processor service name. The service name is defined in the CACQP task entry in the data server configuration. You can define one or more query processors in the configuration file. The `DATASOURCE` name should correspond to the query processor that the client will connect to.

getDescription

Input parameters: None

Return type: `java.lang.String`

Description: Returns the description that was set for the object. Returns null if the description is not set.

getLoginTimeout

Input parameters: None

Return type: `integer`

Description: Returns the timeout value for logging into the database.

getLogWriter

Input parameters: None

Return type: `java.io.PrintWriter`

Description: Returns the `PrintWriter` that writes to the log. Returns null if the log `PrintWriter` is not set.

getPassword

Input parameters: None

Return type: `java.lang.String`

Description: Returns the specified password. Returns null if the password is not specified.

getPooledConnection

Input parameters: None or two `java.lang.String` parameters

Return type: `javax.sql.PooledConnection`

Description: This method uses two signatures. The first one does not take any input parameters, and returns a connection from the connection pool. The second one takes two strings as input parameters. The first string specifies the URL. The second string specifies the connection properties.

getPort

Input parameters: None

Return type: java.lang.String

Description: Returns the port for the object. Returns null if the port is not set.

getPortNumber

See getPort.

getReference

Input parameters: None

Return type: javax.naming.Reference

Description: Returns the object properties of the data source.

getServerName

Input parameters: None

Return type: java.lang.String

Description: Returns the name of the mainframe where the data server runs. Returns null if the server name is not specified.

getURL

Input parameters: None

Return type: java.lang.String

Description: Returns the connection URL that provides the object with enough information to make a connection.

getUser

Input parameters: None

Return type: java.lang.String

Description: Returns the user name. Returns null if the user name is not specified.

setDatabaseName

Input parameters: java.lang.String

Return type: None

Description: Sets the database name equal to the input parameter.

setDescription

Input parameters: java.lang.String

Return type: None

Description: Sets the description equal to the input parameter.

setLoginTimeout

Input parameters: Integer

Return type: None

Description: Sets the login timeout equal to the input parameter.

setLogWriter

Input parameters: java.io.PrintWriter

Return type: None

Description: Sets the log writer equal to the input parameter.

setPassword

Input parameters: java.lang.String

Return type: None

Description: Sets the password equal to the input parameter.

setPort

Input parameters: java.lang.String

Return type: None

Description: Sets the port equal to the input parameter.

setPortNumber

See setPort.

setServerName

Input parameters: java.lang.String

Return type: None

Description: Sets the server name equal to the input parameter. The server name is the name of the mainframe where the data server runs.

setURL

Input parameters: java.lang.String

Return parameters: None

Description: Sets the URL for the object. The URL provides all of the required connection information for the object in a single location. For certain ConnectionPool managers, you should use this method instead of the individual setDatabaseName, setServerName, setPort, setPassword, and setUser methods.

setUser

Input parameters: java.lang.String

Return type: None

Description: Sets the user name equal to the input parameter.

DataSource interface

Use a DataSource object when you manage connection pooling or do not want to use connection pooling. If you want Classic federation to manage connection pooling for you, use a ConnectionPoolDataSource object. If you want Classic federation to manage connection pooling for you and client applications perform distributed transactions, use a XADataSource object instead.

Methods

getConnection

Input parameters: None or two java.lang.String parameters

Return type: java.sql.Connection

Description: Returns a connection to the specified database. This method uses two signatures. The first signature takes no input parameters. The second signature takes two input parameters. The first input parameter specifies the URL. The second input parameter specifies the connection properties.

If the DataSource object does not have enough information to initiate a connection, it returns an SQL connect exception.

getDatabaseName

Input parameters: None

Return type: java.lang.String

Description: Returns the name of the database. Returns null if the database name is not set.

getDataSourceName

Input parameters: None

Return type: java.lang.String

Description: Returns the name of the data source. Returns null if the data source name is not set.

getDescription

Input parameters: None

Return type: java.lang.String

Description: Returns the description for the object. Returns null if the description is not set.

getLoginTimeout

Input parameters: None

Return type: integer

Description: Returns the timeout value for logging into the database.

getLogWriter

Input parameters: None

Return type: java.io.PrintWriter

Description: Returns the PrintWriter that writes to the log. Returns null if the log PrintWriter is not set.

getPassword

Input parameters: None

Return type: java.lang.String

Description: Returns the specified password. Returns null if the password is not specified.

getPort

Input parameters: None

Return type: java.lang.String

Description: Returns the port for the object. Returns null if the port is not set.

getPortNumber

See getPort.

getReference

Input parameters: None

Return type: javax.naming.Reference

Description: Returns the object properties of the data source.

getServerName

Input parameters: None

Return type: java.lang.String

Description: Returns the name of the server. Returns null if the server name is not specified.

getURL

Input parameters: None

Return type: java.lang.String

Description: Returns the connection URL that provides the object with enough information to make a connection.

getUser

Input parameters: None

Return type: java.lang.String

Description: Returns the user name. Returns null if the user name is not specified.

setDatabaseName

Input parameters: java.lang.String

Return type: None

Description: Sets the database name equal to the input parameter.

setDescription

Input parameters: java.lang.String

Return type: None

Description: Sets the description equal to the input parameter.

setLoginTimeout

Input parameters: Integer

Return type: None

Description: Sets the login timeout equal to the input parameter.

setLogWriter

Input parameters: java.io.PrintWriter

Return type: None

Description: Sets the log writer equal to the input parameter.

setPassword

Input parameters: java.lang.String

Return type: None

Description: Sets the password equal to the input parameter.

setPort

Input parameters: java.lang.String

Return type: None

Description: Sets the port equal to the input parameter.

setPortNumber

See setPort.

setServerName

Input parameters: java.lang.String

Return type: None

Description: Sets the server name equal to the input parameter.

setURL

Input parameters: java.lang.String

Return parameters: None

Description: Sets the URL for the object. The URL provides all of the required connection information for the object in a single location. For certain ConnectionPool managers, you should use this method instead of the individual setDatabaseName, setServerName, setPort, setPassword, and setUser methods.

setUser

Input parameters: java.lang.String

Return type: None

Description: Sets the user name equal to the input parameter.

XADataSource interface

Use an XADataSource object for distributed transactions.

The XADataSource class extends the ConnectionPoolDataSource class. Only the methods for the XADataSource class are included in this topic.

Methods

getXAConnection

Input parameters: Two java.lang.String objects

Return type: javax.sql.XAConnection

Description: Returns an XAConnection object that you use for distributed transactions. This method takes two java.lang.String objects as input parameters. The first string provides the URL. The second string provides the connection parameters.

If a connection cannot be established with the information that is provided for the object, the method throws an SQL connect exception.

getXAConnection

Input parameters: None

Return type: javax.sql.XAConnection

Description: Returns an XAConnection object that you use for distributed transactions. If a connection cannot be established with the information that is provided for the object, the method throws an SQL connect exception.

java.sql.properties

You can use several properties objects with the JDBC clients.

Properties objects

CODEPAGE

Description: Optional parameter that specifies the code page that converts characters between systems. Java provides code pages to convert characters between various formats, such as EBCDIC to ASCII.

Do not enter a code page if you use the English version of the Java Runtime Environment. Code page converters are only supported by the international version of the Java Runtime Environment.

For USS support, the setting is as follows: CODEPAGE=USS.

Restrictions:

- Use CODEPAGE=USS only when the environment is pure USS and the local code page is EBCDIC

- If the JVM on USS uses ASCII, do not set CODEPAGE=USS.

FETCHBUFFERSIZE

Description: Optional parameter that specifies the size of the result set buffer that is returned to a client application. You specify this parameter in the configuration file for the client application.

Regardless of the size of the fetch buffer specified, the data server always returns a complete row of data in this buffer. If you set the fetch buffer size to 1, the data server returns single rows of data to the client application.

An appropriate FETCHBUFFERSIZE depends upon the average size of the result set rows that are sent to the client application and the optimum communication packet size. For better performance, fit as many rows as possible into a fetch buffer. The default FETCHBUFFERSIZE is adequate for most queries.

If the FETCHBUFFERSIZE is set smaller than a single result set row, the size of the actual fetch buffer that is transmitted is based on the result set row size. The size of a single result set row in the fetch buffer depends on the number of columns in the result set and the size of the data that is returned for each column.

Use the following formula to determine the size of a result set row:

$(\text{number of data bytes returned}) * (\text{number of columns} * 6)$

There is also a fixed overhead in bytes for each fetch buffer. Use the following formula to calculate the size of this overhead:

$\text{fetchbufferoverhead} = 100 + (\text{number of columns} * 8)$

If your applications routinely retrieve large result sets, contact your network administrator to determine the optimum communication packet size, and set the FETCHBUFFERSIZE to a size that takes this into account.

Maximum value: 64000

Minimum value: 1

Default: 64000

NL CAT

Description: Required parameter that specifies the full path name of the language catalog. The language catalog contains system messages in a specified language. A file contained within the configuration files points to the language catalog. System messages include errors that are generated in the data server and created on the client side.

The default catalog is engcat, or English Catalog, the only supported catalog in this version of JDBC.

RESPONSETIMEOUT

Description: Optional parameter that specifies the response time-out. This value specifies the maximum amount of time in milliseconds that this service waits for an expected response before a connection terminates. The default is 0, wait forever (do not time out). All other values ultimately cause a timeout error and request an end to query processing.

Type of allowed value: Numeric with alpha modifier, which can be as follows:

- MS: Milliseconds
- S: Seconds
- M: Minutes

Representation: Decimal

Maximum value: 1000MS, 60S, and 60M respectively

Minimum value: 0MS

Default: 0M

TRACELEVEL

Description: Optional parameter that regulates the amount of information that is placed in the trace log by data server tasks. Any non-zero number activates the diagnostic tracing. Trace information is recorded by JDBC in the JDBC system log. Tracing can be resource intensive. Do not use it unless you need to debug system problems.

Maximum value: 1

Minimum value: 0

Valid values and results:

1 Generates tracing information

0 Does not generate tracing information

Default: 0

Programming reference for the ODBC/CLI driver

The ODBC/CLI driver includes all of the necessary APIs and functionality to conform to the core specification of Microsoft ODBC 3.51.

In addition to running as an ODBC driver in Win32 environments, the base APIs can be called directly by ISO/IEC and X/Open CAE Call Level Interface (CLI) applications in non-Windows environments such as Solaris, AIX, and HP-UX.

ODBC and CLI applications can use a single set of APIs. A CLI application header file, `caccli.h`, is provided for CLI applications that run in non-Win32 environments. This header file replaces the Microsoft ODBC header files `sql.h` and `sqlext.h`. The prototypes in `caccli.h` include only the CLI subset of the ODBC API function prototypes.

You can find ODBC 3.51 documentation about the APIs and descriptions of error states at the following location:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odbc/htm/dasdkodbcoverview.asp>

Similarities and differences between ODBC and CLI

The differences between ODBC and CLI can affect applications.

Similarities between ODBC and CLI

In addition to core functionality, ODBC and CLI support the following ODBC Level 1 and Level 2 features:

ODBC Level 1

- Schema names in object qualification with two-part naming
- Stored procedures, including metadata queries about stored procedures with `SQLProcedures` and `SQLProcedureColumns`

- Transaction support, including `SQLEndTran` for issuing commit and rollback requests

ODBC Level 2

- `OUTPUT` and `INOUT` parameters in stored procedure calls
- Queries for metadata information about table privileges by using `SQLTablePrivileges`
- Timeouts of login requests and SQL queries

Differences between ODBC and CLI

Some differences exist between the specifications:

- You bind parameters by using `SQLBindParameter` in ODBC applications and `SQLBindParam` in CLI applications.
- CLI applications cannot use these ODBC-only APIs:
 - `SQLBindParameter`
 - `SQLDriverConnect`
 - `SQLMoreResults`
 - `SQLNativeSQL`
 - `SQLNumParams`
 - `SQLProcedureColumns`
 - `SQLProcedures`
 - `SQLTablePrivileges`
- ODBC applications are, by default, auto-commit enabled. Commits are automatically issued when an `SQLExecute` is called for a non-`SELECT` statement. CLI applications do not have the ability to set the auto-commit feature.
- ODBC applications can use ODBC escape sequences in SQL statement text. By default, all SQL that is passed by ODBC applications is scanned for escape sequences. CLI applications have no scanning capability, and all SQL is passed on to the server as-is.

Implemented and deprecated APIs for ODBC and CLI

ODBC includes several APIs that are not part of the CLI specification. This topic lists the APIs that are available for developing CLI applications and the deprecated APIs.

Implemented APIs

The following APIs are implemented in the ODBC/CLI 3.51 driver. These APIs are available to both ODBC and CLI applications unless otherwise specified.

Table 88. Implemented APIs

ODBC API name	Comments
<code>SQLAllocConnect</code>	
<code>SQLAllocEnv</code>	
<code>SQLAllocHandle</code>	
<code>SQLAllocStmt</code>	
<code>SQLBindCol</code>	

Table 88. Implemented APIs (continued)

ODBC API name	Comments
SQLBindParam	CLI only. This API is the same as the ODBC SQLBindParameter API, with the omission of the parameter type and buffer length arguments (arguments 3 and 9). The SQLBindParameter description for the parameters other than InputOutputType and BufferLength in the ODBC documentation can be used as reference material for SQLBindParam. All parameters bound with SQLBindParam are assumed to be INPUT.
SQLBindParameter	ODBC only.
SQLCancel	For SQLCancel to succeed, the server INTERLEAVE INTERVAL parameter must be set to a non-0 value to successfully cancel statements. Interleaving permits checking for additional messages (such as cancel) from clients while SQL request processes.
SQLColAttribute	
SQLColumns	
SQLConnect	
SQLCopyDesc	
SQLCloseCursor	
SQLDataSources	
SQLDescribeCol	
SQLDescribeParam	
SQLDisconnect	
SQLDriverConnect	ODBC only.
SQLEndTran	
SQLError	
SQLExecDirect	
SQLExecute	
SQLFetch	
SQLFetchScroll	Support for scrollable result sets is limited to SQLFetchScroll support with a fetch orientation of SQL_FETCH_NEXT.
SQLFreeConnect	
SQLFreeEnv	
SQLFreeHandle	
SQLFreeStmt	
SQLGetData	
SQLGetDescField	
SQLGetDescRec	

Table 88. Implemented APIs (continued)

ODBC API name	Comments
SQLGetDiagField	
SQLGetDiagRec	
SQLGetConnectAttr	
SQLGetCursorName	
SQLGetEnvAttr	
SQLGetFunctions	
SQLGetStmtAttr	
SQLGetInfo	
SQLGetTypeInfo	
SQLMoreResults	ODBC only.
SQLNativeSql	ODBC only.
SQLNumResultCols	
SQLNumParams	ODBC only.
SQLParamData	
SQLPrepare	
SQLProcedureColumns	ODBC only.
SQLProcedures	ODBC only.
SQLPutData	
SQLRowCount	
SQLSetConnectAttr	
SQLSetCursorName	
SQLSetDescField	
SQLSetDescRec	
SQLSetEnvAttr	
SQLSetStmtAttr	
SQLSpecialColumns	
SQLStatistics	
SQLTablePrivileges	ODBC only.
SQLTables	
SQLTransact	

Deprecated APIs

The following APIs that were deprecated in the ODBC 3.x specification are not supported in the ODBC/CLI driver. These APIs can still be used under the Windows ODBC 3.x driver manager, as they are automatically remapped by the driver manager to their newer replacements:

Table 89. List of deprecated APIs

Deprecated API	ODBC 3.x replacement
SQLColAttributes	SQLColAttribute
SQLGetConnectOption	SQLGetConnectAttr

Table 89. List of deprecated APIs (continued)

Deprecated API	ODBC 3.x replacement
SQLGetStmtOption	SQLGetStmtAttr
SQLParamOptions	SQLSetStmtAttr
SQLSetConnectOption	SQLSetConnectAttr
SQLSetParam	SQLBindParameter
SQLSetScrollOption	SQLSetStmtAttr
SQLSetStmtOption	SQLSetStmtAttr

C and SQL data types for ODBC and CLI

This topic lists the C and SQL data types for ODBC and CLI applications.

C data types for ODBC applications

You can pass the following C data types when you bind result set columns and parameters from ODBC applications.

```
SQL_C_DEFAULT  
SQL_C_CHAR  
SQL_C_LONG  
SQL_C_SLONG  
SQL_C_ULONG  
SQL_C_SHORT  
SQL_C_SSHORT  
SQL_C_USHORT  
SQL_C_FLOAT  
SQL_C_DOUBLE  
SQL_C_BINARY  
SQL_C_NUMERIC
```

C data types for CLI applications

You can pass the following data type values when you bind result set columns and parameters from CLI applications.

```
SQL_DEFAULT  
SQL_CHAR  
SQL_INTEGER  
SQL_SMALLINT  
SQL_FLOAT  
SQL_REAL  
SQL_DOUBLE
```

SQL data types for ODBC and CLI applications

You can pass the following SQL data type values when you bind result set columns and parameters from ODBC and CLI applications.

```
SQL_CHAR  
SQL_VARCHAR  
SQL_LONGVARCHAR  
SQL_INTEGER  
SQL_SMALLINT  
SQL_FLOAT  
SQL_REAL  
SQL_DOUBLE  
SQL_DECIMAL  
SQL_TYPE_DATE  
SQL_TYPE_TIME
```

```
SQL_TYPE_TIMESTAMP
SQL_TYPE_DATE
SQL_BINARY
SQL_VARBINARY
SQL_LONGVARBINARY
```

In the Win32 environment, the ODBC 3.x driver manager is required to use the ODBC 3.51 driver. This version of the driver manager automatically supports both 3.x applications and older, pre-3.x applications. Calls to deprecated APIs by older applications are automatically re-mapped to the 3.x APIs.

Binding input and output parameters from CLI applications

CLI applications must bind parameters for stored procedure calls by using the SQLBindParam API call.

Unlike the ODBC SQLBindParameter function with which you can pass the parameter type, SQLBindParam assumes that all parameters are input type parameters for the stored procedure call.

To bind output and input parameters from a CLI application, programs must retrieve the implementation parameter descriptor after calling SQLBindParam and modify the descriptors parameter mode field by using SQLSetDescField. Descriptors make bound parameter or column information available to ODBC and CLI applications.

Example: To bind an input parameter for a stored procedure call, the application program might issue the following API calls:

```
/* Bind an 8 byte character parameter. CLI assumes the parameter */
/* is INPUT, mode must be changed to INOUT after the bind */
sqlrc = SQLBindParam( hStmt, 1, SQL_CHAR, SQL_CHAR,
                    8, 0, DataPtr, IndPtr );
/* Retrieve the implementation parameter descriptor */
if ( sqlrc == SQL_SUCCESS )
    sqlrc = SQLGetStmtAttr( hStmt, SQL_ATTR_IMP_PARAM_DESC,
                          &hIPD, sizeof(hIPD), NULL );
/* Change the parameter's mode from the INPUT default to OUTIN */
if ( sqlrc == SQL_SUCCESS )
    sqlrc = SQLSetDescField( hIPD, 1, SQL_DESC_PARAMETER_MODE,
                          (SQLPOINTER) SQL_PARAM_MODE_INOUT,
                          sizeof( SQLPOINTER ) );
```

Logs for the ODBC/CLI driver

The ODBC/CLI driver automatically logs errors and debugging traces when the configuration trace level is set to a value less than 8.

The amount of tracing varies with the trace level value. The trace value 0 produces the maximum amount of tracing, and 7 log errors only. In general, set tracing to 8 unless IBM Software Support requests diagnostic information.

In Windows 32-bit and UNIX environments, the log file is named CACLOG and is placed in the same directory as the ODBC/CLI driver itself. You can set the CACLOG environment variable to point to a different directory and file name. This file is overwritten each time the ODBC/CLI driver runs.

On UNIX, the driver managers also support logging. Typically, the tracing that the driver manager provides is sufficient to debug problems that you might encounter. If additional tracing is required, you can set the ODBC/CLI trace levels and obtain the ODBC/CLI traces.

The ODBC/CLI software logs the following categories of information:

- Diagnostic messages. If an API call results in the creation of a diagnostic record that is due to an ERROR or INFO situation, the message is logged. If the message is an error message, then logging takes place if the TRACE LEVEL parameter is less than 8. If the message is an INFO message, then logging takes place if the TRACE LEVEL parameter is less than 3.
- API call entry and exit with return code. With few exceptions, API calls start with the validation of a passed handle and end with unlocking the passed handle. The API called and the return code are logged if trace level is set to 1. In cases where any nonvalid handle is passed or an SQL_ERROR is returned, the logging takes place if the TRACE LEVEL parameter is set to any value less than 8.

Recommendation: Do not use logging for ODBC applications. The log file cannot be shared by multiple application processes, and ODBC has its own tracing facility.

The log file is binary. You can use the cacprtlog utility to format and display the log messages. On Windows, you can also view log messages in the Application log of the System Event Viewer.

Code pages for the ODBC/CLI driver

Classic federation supports databases that are enabled for SBCS and DBCS data.

Those databases include DB2, IMS, VSAM, sequential, and CA-IDMS. The Classic federation Windows ODBC driver translates SBCS and DBCS data by using ICU4C to perform code page conversions

The form of character data that Classic federation supports is *mixed-mode* data. Mixed-mode character data can be strictly SBCS data or can include DBCS data. ODBC driver support includes conversion of graphic data types and bidirectional languages.

- In Windows, you can use the ODBC Data Source Administrator interface to define client and server code pages when you configure the ODBC data source.
- In Linux and UNIX, you define the client and server code pages in your configuration file for your CLI driver. If you have data sources for which you want to use different code pages, you can define the data sources in separate configuration files.

Restrictions for bidirectional layout transformation

Bidirectional layout transformation is restricted in the following situations:

- Problems can occur in visual to logical text transformations when converting complex multidirectional text and when numbers are involved in the conversion. For better results, you can insert bidirectional control characters (LRM <left-to-right mark> and RLM <right-to-left mark>). However, these control characters are not defined in all client code pages. For the code pages that do not include bidirectional control characters, the ICU conversion replaces them by substitution characters. As a result, text might not display correctly.
- The column size definitions on the data server must be large enough to allow space for expansion. Otherwise, layout transformation fails with a decoding error. The length of a string might increase when layout transformation occurs from the server to the client code pages in the following situations:
 - With Arabic text and when the SHAPING option is set.

- When converting from visual to logical text presentations for Arabic and Hebrew text due to the insertion of bidirectional control characters.
- Bidirectional options are not available for the Japanese versions of the ODBC Administrator and the Microsoft ODBC Data Source Administrator.

Restrictions on accessing binary data

When you access binary data from the ODBC driver, some restrictions apply to data type conversions.

The following conversions are not supported for binary data:

- Conversion of input host variables from SQL_C_BINARY to any numeric SQL or column types.
- Conversion from any numeric C or SQL data type into BINARY and VARBINARY columns.
- Conversion from any binary C and SQL data type into GRAPHIC, VARGRAPHIC and LONG VARGRAPHIC.

National language support

You can access non-English data, including double byte character set (DBCS) data for some data sources.

DBCS support is provided for DB2, CA-IDMS, IMS, sequential, and VSAM. National language support is implemented in the clients and the query processor.

For clients, you use a set of configuration parameters that define the local code pages for the client and the server code pages. For Chinese and Japanese, *mixed-mode*, a mixture of SBCS and DBCS data, code pages perform all data conversions. When you access pure DBCS data (the GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC data types), federation automatically performs the required conversions to use this data with the mixed-mode code page converters.

For most U.S. and European customers, conversion parameters that define code pages are not required. By default, federation performs the appropriate ASCII and EBCDIC conversions for you.

The query processor also supports mixed-mode and DBCS comparison operations. Comparison operations are performed by using memory comparisons.

SBCS, DBCS, and database objects

Classic federation supports SBCS, DBCS, and mixed mode data.

You can use the following database objects:

- DBCS data in GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC data types
These data types are expected to contain pure DBCS data. For these data types, the DBCS-related code page conversions are performed.
- SBCS and DBCS mixed data in CHAR, VARCHAR, and LONG VARCHAR data types
- Mixed mode object names delimited in double quotes.
- DBCS column names in the AS clause

The CHAR, VARCHAR, and LONG VARCHAR data types can contain SBCS data or mixed-mode data. Classic federation inspects the contents of these data types

and performs the appropriate conversions by using the code page information in the SBCS-related configuration parameters. When the query processor manipulates mixed-mode data, you must delimit the DBCS data with DBCS shift codes.

Code page conversion for drivers

You can specify code page conversion information that the JDBC driver, Windows ODBC driver, or the UNIX and Linux CLI client use to transfer data. For bidirectional server and client code pages, you can set bidirectional language options.

You use the following methods to define code pages for the Classic federation drivers:

- For the JDBC driver, you use the CODEPAGE parameter in the `java.sql.properties` file.
- For the ODBC driver, you use the Code Pages tab of the ODBC Administrator.
- For the CLI driver on UNIX and Linux, you use the CLIENT CODEPAGE and SERVER CODEPAGE parameters in the `cac.ini` configuration file.

Code page converters

Use the tables in this topic to identify the appropriate code page converters for your single byte or double byte locale. You can then specify the converters for the Windows ODBC driver in the Windows ODBC Administrator, or the converters for the UNIX and Linux CLI client interface in the UNIX and Linux `cac.ini` configuration file.

Multilingual data conversion

Depending on whether you are setting up an SBCS or DBCS locale, select a code page converter from the pair of tables related to that locale. Refer to the z/OS table for the server code page, and the Windows and Unix table for the client code page.

1. Find the language you require in the Character set column or the code page number in the Code page column.
2. Scan across the row and confirm the ODBC name (Windows) or the Converter name (UNIX and Linux).
3. Specify the ODBC name in the Windows ODBC Administrator or the Converter name in the `cac.ini` configuration file.

SBCS locales

For SBCS (non-graphic) locales, the following tables identify the available code page converters for single byte character sets:

- z/OS EBCDIC SBCS converter support
- Windows and UNIX SBCS converter support

DBCS locales

For DBCS (graphic) locales such as Chinese or Japanese, the following tables identify the available code page converters for double byte character sets:

- z/OS EBCDIC mixed-mode (DBCS) converter support
- Windows and UNIX mixed-mode (DBCS) converter support

Code page converters

Table 90. z/OS EBCDIC SBCS converter support

Converter name (UNIX and Linux)	Code page	ODBC name (Windows)	Character set
ibm-37	37	ibm-37_P100-1995,swaplfnl	CECP (Country Extended Code Page), USA, Canada (ESA*), Netherlands, Portugal, Brazil, Australia, New Zealand
ibm-273	273	ibm-273_P100-1995	CECP, Austria, Germany
ibm-277	277	ibm-277_P100-1995	CECP, Denmark, Norway
ibm-278	278	ibm-278_P100-1995	CECP, Finland, Sweden
ibm-280	280	ibm-280_P100-1995	CECP, Finland, Sweden
ibm-284	284	ibm-284_P100-1995	CECP, Spain, Latin America, Spanish
ibm-285	285	ibm-285_P100-1995	CECP, United Kingdom
ibm-290	290	ibm-290_P100-1995	Japanese Katakana host extended SBCS
ibm-297	297	ibm-297_P100-1995	CECP, France
ibm-420	420	ibm-420_X120-1999	Arabic, all presentation shapes string type 4
ibm-424	424	ibm-424_P100-1995	Hebrew I, legacy IDs: CS 941/2, CP 424/2, string type 4
ibm-500	500	ibm-500_P100-1995	CECP, Belgium, Canada (AS/400*), Switzerland, International Latin-1
ibm-803	803	ibm-803_P100-1999	Hebrew Set A, legacy code, string type 4
ibm-838	838	ibm-838_P100-1995	Thai host extended SBCS
ibm-870	870	ibm-870_P100-1995	Latin-2 - EBCDIC multilingual
ibm-871	871	ibm-871_P100-1995	CECP, Iceland
ibm-875	875	ibm-875_P100-1995	Greek
ibm-1025	1025	ibm-1025_P100-1995	Cyrillic, Multilingual
ibm-1026	1026	ibm-1026_P100-1995	Turkey Latin-5
ibm-1047	1047	ibm-1047_P100-1995,swaplfnl	Latin-1 and Open Systems
ibm-1097	1097	ibm-1097_P100-1995	Farsi
ibm-1112	1112	ibm-1112_P100-1995	Baltic, multilingual
ibm-1122	1122	ibm-1122_P100-1999	Estonia
ibm-1123	1123	ibm-1123_P100-1995	Cyrillic Ukraine
ibm-1130	1130	ibm-1130_P100-1997	Vietnamese
ibm-1132	1132	ibm-1132_P100-1998	Lao
ibm-1137	1137	ibm-1137_P100-1999	Devanagari EBCDIC, based on Unicode character set

Table 90. z/OS EBCDIC SBCS converter support (continued)

Converter name (UNIX and Linux)	Code page	ODBC name (Windows)	Character set
ibm-1140	1140	ibm-1140_P100-1997,swaplfnl	ECECP, USA, Canada, Netherlands, Portugal, Brazil, Australia, New Zealand
ibm-1141	1141	ibm-1141_P100-1997	ECECP, Austria, Germany
ibm-1142	1142	ibm-1142_P100-1997,swaplfnl	ECECP, Denmark, Norway
ibm-1143	1143	ibm-1143_P100-1997,swaplfnl	ECECP, Finland, Sweden
ibm-1144	1144	ibm-1144_P100-1997,swaplfnl	ECECP, Italy
ibm-1145	1145	ibm-1145_P100-1997,swaplfnl	ECECP, Spain, Latin America, Spanish
ibm-1146	1146	ibm-1146_P100-1997,swaplfnl	ECECP, United Kingdom
ibm-1147	1147	ibm-1147_P100-1997,swaplfnl	ECECP, France
ibm-1148	1148	ibm-1148_P100-1997,swaplfnl	ECECP, International 1
ibm-1149	1149	ibm-1149_P100-1997,swaplfnl	ECECP, Iceland
ibm-1153	1153	ibm-1153_P100-1999,swaplfnl	Latin-2 - EBCDIC, multilingual with euro
ibm-1154	1154	ibm-1154_P100-1999	Cyrillic Multilingual with euro
ibm-1155	1155	ibm-1155_P100-1999	Turkey, Latin-5 with euro
ibm-1156	1156	ibm-1156_P100-1999	Baltic, multilingual with euro
ibm-1157	1157	ibm-1157_P100-1999	Estonia, EBCDIC with euro
ibm-1158	1158	ibm-1158_P100-1999	Cyrillic Ukraine, EBCDIC with euro
ibm-1160	1160	ibm-1160_P100-1999	Thai host with euro
ibm-1164	1164	ibm-1164_P100-1999	Vietnamese with euro
ibm-4899	4899	ibm-4899_P100-1998	Hebrew Set A, legacy code, maximal set with euro and new sheqel, string type 4
ibm-4971	4971	ibm-4971_P100-1999	Greek, with euro
ibm-5123	5123	ibm-5123_P100-1999	Japanese Latin host extended SBCS, with euro
ibm-8482	8482	ibm-8482_P100-1999	Japanese Katakana, with euro, growing CS
ibm-12712	12712	ibm-12712_P100-1998,swaplfnl	Hebrew, max set with euro and new sheqel, string type 10
ibm-16804	16804	ibm-16804_X110-1999,swaplfnl	Arabic, all presentation shapes, string type 4, with euro

Table 91. Windows and UNIX SBCS converter support

Converter name	Code page	ODBC name	Description
ibm-367	367	ibm-367_P100-1995	ANSI X3.4 ASCII standard; USA

Table 91. Windows and UNIX SBCS converter support (continued)

Converter name	Code page	ODBC name	Description
ibm-437	437	ibm-437_P100-1995	USA, many other countries and regions, PC base PC data
ibm-737	737	ibm-737_P100-1997	MS-DOS Greek, PC data
ibm-775	775	ibm-775_P100-1996	MS-DOS Baltic, PC data
ibm-813	813	ibm-813_P100-1995	ISO 8859-7, Greek and Latin
ISO-8559-1	819	ISO-8859-1	ISO 8859-1, Latin-1 countries and regions
ibm-850	850	ibm-850_P100-1995	LP 222, Latin-1 countries and regions, PC data
ibm-851	851	ibm-851_P100-1995	Greek, PC data
ibm-852	852	ibm-852_P100-1995	Latin-2 multilingual, PC data
ibm-855	855	ibm-855_P100-1995	Cyrillic, PC data
ibm-856	856	ibm-856_P100-1995	Hebrew, string type 5, PC data
ibm-857	857	ibm-857_P100-1995	Turkey Latin-5, PC data
ibm-858	858	ibm-858_P100-1997	MLP 222, Latin-1 with euro, Latin-1 countries and regions, PC data
ibm-860	860	ibm-860_P100-1995	Portugal, PC data
ibm-861	861	ibm-861_P100-1995	Iceland, PC data
ibm-862	862	ibm-862_P100-1995	Hebrew, migration, string type 4, PC data
ibm-863	863	ibm-863_P100-1995	Canada, PC data
ibm-864	864	ibm-864_X110-1999	Arabic, string type 5, PC data
ibm-865	865	ibm-865_P100-1995	Denmark, Norway, PC data
ibm-866	866	ibm-866_P100-1995	Cyrillic, Russian, PC data
ibm-867	867	ibm-867_P100-1998	Hebrew, a modification of code page 862, string type 4, PC data
ibm-869	869	ibm-869_P100-1995	Greek, PC data
ibm-874	874	ibm-874_P100-1995	Thai PC data extended SBCS
ibm-878	878	ibm-878_P100-1996	Russian Internet koi8-r
ibm-897	897	ibm-897_P100-1995	Japanese PC data single byte.
ibm-901	901	ibm-901_P100-1999	Baltic, 8-bit with euro
ibm-902	902	ibm-902_P100-1999	Estonia, 8-bit with euro
ibm-912	912	ibm-912_P100-1995	Latin-2 - ISO 8859-2
ibm-913	913	ibm-913_P100-2000	ISO Latin-3 - 8859-3
ibm-914	914	ibm-914_P100-1995	Latin-4 - ISO 8859-4
ibm-915	915	ibm-915_P100-1995	Cyrillic, 8-bit, ISO 8859-5
ibm-916	916	ibm-916_P100-1995	ISO 8859-8, Hebrew, string type 5
ibm-920	920	ibm-920_P100-1995	ISO 8859-9 Latin-5, ECMA-128, Turkey TS-5881

Table 91. Windows and UNIX SBCS converter support (continued)

Converter name	Code page	ODBC name	Description
ibm-921	921	ibm-921_P100-1995	Baltic, 8-bit, ISO 8859-13
ibm-922	922	ibm-922_P100-1999	Estonia, 8-bit
ibm-923	923	ibm-923_P100-1998	ISO 8859-15, Latin-9 with euro, total of 8 characters replaced from 819
ibm-1051	1051	ibm-1051_P100-1995	HP emulation, for use with Latin-1. GCGID SF150000 is mapped to a control X'7F'
ibm-1089	1089	ibm-1089_P100-1995	ISO 8859-6, Arabic, string type 5
ibm-1098	1098	ibm-1098_P100-1995	Farsi, personal computer
ibm-1124	1124	ibm-1124_P100-1996	Cyrillic Ukraine, 8-Bit
ibm-1125	1125	ibm-1125_P100-1997	Cyrillic Ukraine, PC data Windows, Cyrillic
ibm-1129	1129	ibm-1129_P100-1997	ISO-8 Vietnamese
ibm-1131	1131	ibm-1131_P100-1997	Cyrillic Belarus, PC data
ibm-1133	1133	ibm-1133_P100-1997	ISO-8 Lao
ibm-1162	1162	ibm-1162_P100-1999	Thai Windows, with euro
ibm-1168	1168	ibm-1168_P100-2002	Ukrainian KOI8-U
ibm-1250	1250	ibm-1250_P100-1995	Windows Latin-2
ibm-1251	1251	ibm-1251_P100-1995	Windows, Cyrillic
ibm-1252	1252	ibm-1252_P100-2000	Windows, Latin-1
ibm-1253	1253	ibm-1253_P100-1995	Windows, Greek
ibm-1254	1254	ibm-1254_P100-1995	Windows, Turkey
ibm-1255	1255	ibm-1255_P100-1995	Windows, Hebrew, string type 5
ibm-1256	1256	ibm-1256_P110-1997	Windows, Arabic, string type 5
ibm-1257	1257	ibm-1257_P100-1995	Windows, Baltic Rim
ibm-1258	1258	ibm-1258_P100-1997	Windows, Vietnamese
ibm-1276	1276	ibm-1276_P100-1995	Adobe PostScript standard encoding
ibm-4909	4909	ibm-4909_P100-1999	ISO-8, Greek and Latin with euro
ibm-5346	5346	ibm-5346_P100-1998	Windows Latin-2, version 2 with euro
ibm-5347	5347	ibm-5347_P100-1998	Windows, Cyrillic version 2 with euro
ibm-5348	5348	ibm-5348_P100-1997	Windows, Latin-1, Version 2 with euro)
ibm-5349	5349	ibm-5349_P100-1998	Windows, Greek version 2 with euro
ibm-5350	5350	ibm-5350_P100-1998	Windows, Turkey version 2 with euro

Table 91. Windows and UNIX SBCS converter support (continued)

Converter name	Code page	ODBC name	Description
ibm-5351	5351	ibm-5351_P100-1998	Windows, Hebrew version 2 with euro, string type 5
ibm-5352	5352	ibm-5352_P100-1998	Windows, Arabic version 2 with euro, string type 5
ibm-5353	5353	ibm-5353_P100-1998	Windows, Baltic Rim version 2 with euro
ibm-5354	5354	ibm-5354_P100-1998	Windows, Vietnamese version 2 with euro
ibm-9005	9005	ibm-9005_X100-2005	Greek ISO 8859-7,2003
ibm-9447	9447	ibm-9447_P100-2002	Windows, Hebrew, Windows 1255-12/2001, string type 5
ibm-9449	9449	ibm-9449_P100-2002	Windows, Baltic Rim with euro and 7 additional characters

Table 92. z/OS EBCDIC mixed-mode (DBCS) converter support

Converter Nname	Code page	ODBC name	Description
ibm-930	930	ibm-930_P120-1999	Japanese Katakana-Kanji host mixed including 4370 UDC (User Defined Character), extended SBCS
ibm-933	933	ibm-933_P110-1995	Korean host mixed including 1880 UDC, Extended SBCS
ibm-935	935	ibm-935_P110-1999	Simplified Chinese host mixed including 1880 UDC, extended SBCS
ibm-937	937	ibm-937_P110-1999	Traditional Chinese host mixed including 6204 UDC, extended SBCS
ibm-939	939	ibm-939_P120-1999	Japanese Latin-Kanji host mixed including 4370 UDC, extended SBCS
ibm-1364	1364	ibm-1364_P110-1997	Korean host mixed extended including 11,172 full Hangul
ibm-1371	1371	ibm-1371_P100-1999	Traditional Chinese host mixed including: 6204 UDC,Extended SBCS including SBCS and DBCS euro
ibm-1388	1388	ibm-1388_P103-2001	Simplified Chinese DBCS- GB 18030 host with UDCs and Uygur extension.

Table 92. z/OS EBCDIC mixed-mode (DBCS) converter support (continued)

Converter Nname	Code page	ODBC name	Description
ibm-1390	1390	ibm-1390	Extended Japanese Katakana-Kanji host mixed for JIS X0213 (Japanese Industrial Standards), including: 6205 UDC,extended SBCS, includes SBCS & DBCS euro
ibm-1399	1399	ibm-1399	Extended Japanese Latin-Kanji host mixed for JIS X0213 including: 6205 UDC,extended SBCS, includes SBCS & DBCS euro

Table 93. Windows and UNIX mixed-mode (DBCS) converter support

Converter name	Code page	ODBC name	Description
ibm-942	942	ibm-942_P12A-1999	Japanese PC data mixed including 1880 UDC, extended SBCS
ibm-943_P130	943	ibm-943_P130-1999	Japanese PC data mixed for open environment multi-vendor code, 6878 JIS X 0208-1990 chars, 386 IBM selected DBCS chars, 1880 UDC (X'F040' to X'F9FC')
ibm-943_P15A	943	ibm-943-P15A-2003	Japanese, PC data mixed for open environment multi-vendor code: 6878 JIS X 0208-1990 chars,386 IBM selected DBCS chars, 1880 UDC (X'F040' to X'F9FC')
ibm-949_P110	949	ibm-949_P110-1999	IBM KS code - PC data mixed including 1880 UDC
ibm-949_P11A	949	ibm-949_P11A-1999	IBM KS code - PC data mixed including 1880 UDC
ibm-950	950	ibm-950_P110-1999	Traditional Chinese PC data mixed for IBM BIG-5
ibm-954	954	ibm-954_P101-2000	Japanese EUC (Extended Unix Code), G0 - JIS X201 Roman set 00895, G1 - JIS X208-1990 set 00952, G2 - JIS X201 Katakana set 04992, G3 - JIS X212 set 00953

Table 93. Windows and UNIX mixed-mode (DBCS) converter support (continued)

Converter name	Code page	ODBC name	Description
ibm-964	964	ibm-964_P110-1999	Traditional Chinese EUC; G1 - CNS 11643 plane 1 00960; G2 - CNS 11643 plane 2 00961
ibm-970	970	ibm-970_P110-1995	Korean EUC; G0 - ASCII; G1 - KS C5601-1989, with 188 UDC
ibm-1363_P110	1363	ibm-1363_P110-1997	Windows Korean PC Mixed, including 11,172 full Hangul
ibm-1363_P11B	1363	ibm-1363-P11B-1998	Windows Korean PC Mixed, including 11,172 full Hangul
ibm-1375	1375	ibm-1375_P100-2003	Mixed big-5 extension for HKSCS (Hong Kong Supplementary Character Set)
ibm-1383	1383	ibm-1383_P110-1999	Simplified Chinese EUC; G0 set, ASCII; G1 set, GB 2312-80 set 1382
ibm-1386	1386	ibm-1386_P100-2002	Simplified Chinese PC data GBK (Guojia Biaozhun Kuozhan) mixed, all GBK character set and other growing characters
ibm-33722	33722	ibm-33722_P120-1999	Japanese EUC (IBMeucJP); G1 - JIS X208-1990 set 00952; G2 - JIS X201 Katakana set 04992; G3 - JIS X212 set 09145

Sample VTAM and CICS definitions for federated or stored procedure access

Configure resource definitions for VTAM and CICS to set up federated and stored procedure access to CICS.

For federated access to CICS, configure the following definitions:

- **VTAM:** An APPL definition on the data server to provide a local Logical Unit (LU) name for the communication session
- **CICS:** Program and transaction definitions for communication between the data server and the CICS system

For stored procedure access to CICS, configure the following definitions:

- **VTAM:** Additional APPL definitions to accommodate the expected number of concurrent requests that the data server manages
- **CICS:** File definitions and session definitions in addition to program and transaction definitions

VTAM resource definitions for federated or stored procedure access

VTAM resource definitions require an APPL definition on the server to provide a local Logical Unit (LU) name for the communication session.

Federated access

For federated access, configure a single APPL definition to provide an LU name for the communication session with CICS. See the topic VTAM APPL definition and mode table entry definition.

Stored procedure access

For stored procedure access, you might need multiple APPL definitions (local LU names), depending upon the number of procedure users on the data server that you allow to be active concurrently. Allow one local LU name per active server-stored procedure user to eliminate the possibility of communication failures because the local LU name is busy. However, a one-to-one relationship between local LU names and active server-stored procedure users is not usually required. In general, a relatively small number of local LU names is adequate for most sites. The required number of local LU names is the number of expected concurrent requests that the server handles. Set the value of the **MAXTHREADS** parameter for the query processor to the number of local LU names that are defined.

You assign the Application Control Block (ACB) name in the APPL definition. The ACB name is specified on the OPEN request. This request is issued by the user-written stored procedure that runs in the data server address space. In the sample communication programs CACSPCOM or CACSPVCM, a pool of APPL definitions must be created by assigning sequentially ascending ACB names like CACAPPC0, CACAPPC1, CACAPPC2, and so on. To use this pool of ACB names, the OPEN request must then specify the local LU name as CACAPPC*. The communications processor attempts to open the specified ACB name after replacing the asterisk position with a sequentially ascending value beginning at zero.

If a pool of 10 ACB names is insufficient, the APPL definitions can use names that end in two digits, for example, CACPPC00, CACPPC01, CACPPC02, and so on. To use this pool of ACB names, the OPEN request must then specify the local LU name as CACPPC**. You can specify up to seven suffix asterisk characters (*).

Important: You need to carefully control the size of the APPL definition pool. If an OPEN request fails, the next ACB name is generated, and the stored procedure attempts another OPEN request. If each subsequent OPEN request fails, the stored procedure attempts the entire pool of ACB names before the communications processor reports an OPEN failure. To use only a specific ACB name, the OPEN request specifies the exact name without an asterisk suffix.

VTAM APPL definition and mode table entry definition

The following examples demonstrate how to create an APPL definition and a VTAM mode table entry definition.

VTAM APPL definition

VTAM APPL Definition

```
CACAPPL VBUILD TYPE=APPL
CACICS1 APPL ACBNAME=CACICS1,
```

X

```

          APPC=YES, X
          AUTOSES=1, X
          MODETAB=CACCMODE, X
          DLOGMOD=MTLU62, X
          AUTH=(ACQ), X
          EAS=100, PARSESS=YES, X
          SONSCIP=YES, X
          DMINWNL=0, X
          DMINWNR=1, X
          DSESLIM=100
CACCCICS2 APPL ACBNAME=CACCCICS2, X
               APPC=YES, X
               AUTOSES=1, X
               MODETAB=CACCMODE, X
               DLOGMOD=MTLU62, X
               AUTH=(ACQ), X
               EAS=1, PARSESS=YES, X
               SONSCIP=YES, X
               DMINWNL=0, X
               DMINWNR=1, X
               DSESLIM=1

```

The example shows APPL definitions to be used by the example program CACSPCOM or CACSPVCM. Your actual APPL definitions can vary based upon site standards, and you can use the examples shown here for federated access. If you set up both federated and stored procedure access, define different names for each.

The example provides only two APPL definitions, so in this case the pool actually consists of only two entries. CACCMODE is defined as the Logon Mode Table name, and MTLU62 is defined as the Logon Mode Table entry name. The Logon Mode Table entry must be in either the specified Logon Mode Table or in ISTINCLM, an IBM-supplied Logon Mode Table. The OPEN request issued by CACSPCOM also specifies the Logon Mode Table entry (DLOGMOD) name.

If you plan to run the example program CACSPCOM and changes were made to the ACB name or DLOGMOD, you must correct CACSPCOM to specify the modified values, recompile, and link-edit the program before you run the example. You can make changes to MODETAB to identify your correct Mode Table Name without affecting the example program, CACSPCOM.

VTAM mode table entry definition

VTAM Mode Table Entry Definition

```

CACCMODE  MODEENT LOGMODE=CACCMODE, *
          TYPE=0, *
          FMPROF=X'13', *
          TSPROF=X'07', *
          PRIPROT=X'B0', *
          SECPROT=X'B0', *
          COMPROT=X'D0B1', *
          RUSIZES=X'8585', *
          PSERVIC=X'0602000000000000000000000000300'

```

The example above shows a Mode Table entry definition that is used by the example program CACSPCOM or CACSPVCM. Your actual Mode Table entry definition can vary based upon site standards. As stated earlier, the OPEN request issued by CACSPCOM specifies Logon Mode Table entry MTLU62. If you plan to run the example program CACSPCOM and changes were made to LOGMODE, you must correct CACSPCOM to specify the new name, recompile, and link-edit the program before attempting to run the example. The CICS system must have

access to a Logon Mode Table entry with an identical name.

Configuring CICS resource definitions for federated or stored procedure access

To complete specific resource definitions in CICS, you need to review several definitions within your CICS or VTAM system.

Before you begin

Run the sample JCL found in the CACCDEF member of *user.SCACSAMP* to create the basic CICS definitions required for CICS VSAM support. See the topic about Setting up access to CICS VSAM.

About this task

The CICS Application Identifier (APPLID) becomes the remote LU name for the communication session. If you implement stored procedure access, the CICS APPLID can handle multiple sessions from stored procedure processors on the data server.

For more information about CICS resource definitions and intersystem connectivity, see the IBM documentation on CICS/ESA intercommunications.

To configure CICS resource definitions for federated or stored procedure access:

Procedure

1. Ensure that the CICS system initialization table (DFHSIT) definition or initialization overrides include ISC=YES to enable intercommunication programs.
2. Ensure that the ACF/VTAM application definition for your CICS system includes the following options on the VTAM APPL statement.

AUTH=(ACQ,VPACE,...)

Allows CICS to acquire LUTYPE6 sessions and to allow pacing of intersystem flows.

VPACING=*n*

Specifies the pacing rate.

EAS=*n*

Specifies the maximum number of network addressable units with which CICS can establish sessions.

PARSESS=YES

Enables LUTYPE6 parallel session support.

SONSCIP=YES

Enables session outage notification support.

3. Ensure that the APPL statement does not specify APPC=YES.
4. If you implement stored procedure access, install resource definitions for the required LU6.2 connection handler.
 - a. Configure and install the program definition.

This program is in the load module CACSP62 in *user.SCACLOAD*. This load module must be in a library that is included in the DFHRPL concatenation for your CICS system. The program name appears only in the transaction definition.
 - b. Configure and install the transaction definition.

You assign the transaction name. The default transaction name is XASP. The OPEN request from the sample program CACSPCOM specifies the default name, so if you change the transaction name, you must correct CACSPCOM to specify the new transaction name, recompile, and link-edit the program before attempting to run CACSPCOM.

CEDA definitions for federated or stored procedure access

The following examples show online resource definitions that define the CICS resources that you need for federated or stored procedure access to CICS. The examples use the CICS-provided CEDA transaction.

Installing CICS resource definitions

After you configure and verify the CICS resource definitions shown in the examples, install the group by using the online command CEDA INSTALL GROUP(gggg) ALL. When you install the group, the definitions are immediately available for use.

Add the group to the startup group list, so the stored procedure group is automatically installed during each subsequent startup of the CICS system. To add the group to the startup group list, locate the **GRPLIST** parameter with the format GRPLIST=xxxxxxx on the system initialization table (DFHSIT) or in the SIT overrides used to start the CICS system. Use the name of the group list in the online command CEDA ADD GROUP(gggg) LIST(xxxxxxx) to permanently add the stored procedure processing group to the CICS system startup.

Federated access

Federated access requires that you install a program definition and a transaction definition, as shown in the examples for federated access. The resource definitions that you need include the following:

- The program definition for the sample program CACCIVS.
- The transaction definition for CACCIVS.

The default transaction name is EXV1.

Beginning in Classic federation Version 10.1 PTF rollup 2, the CACCIVS program provides access to extended-format VSAM data sets with CICS version 3.2 or higher.

The required fields are in underlined bold type, for example, xxx. Define fields that contain only lower case characters according to site standards. Enter fields that are in mixed case exactly as shown.

Stored procedure access

If you want to add stored procedure access, use the examples to install the sample CICS resource definitions for stored procedures.

The resource definitions that you need include the following:

- The program definitions for the sample programs CACSP62 and CACSPREM.
- The transaction definition XASP for the CACSP62 program.
- The file definition for the sample VSAM Employee file CACIVP.
- A CONNECTION/SESSION definition is required for each LU name that you might use to communicate with CICS from the data server.

This example provides two LU NAMES for VTAM:

- CACPPC00
- CACPPC01

The sample program CACSPCOM, which runs in the data server address space, uses one of these local LU names when it requests an **OPEN** for the LU6.2 conversation.

The required fields are in underlined bold type, for example, xxx. Define fields that contain only lower case characters according to site standards. Enter fields that are in mixed case exactly as shown.

Language considerations with CACSPREM

The CACSPREM example program is a COBOL II program. The program, as supplied, does not use Language Environment/370 facilities. However, if you compile the program using an SAA AD/Cycle COBOL/370 Version 1 Release 1 or later, the attribute LANGUAGE ==>Le370 must remain as shown in the example. If the program is not compiled by a Language Environment/370 enabled compiler, change the attribute to LANGUAGE==>C0bo1.

If the LANGUAGE attribute specifies Le370, you must have Language Environment/370 support installed in your CICS system. If the LANGUAGE attribute specifies COBOL, you must install the VS COBOL II interface in your CICS system. If CICS is not enabled to support the LANGUAGE attribute specified for this program, see the CICS documentation about system definitions.

Example definitions for federated access

PROGRAM DEFINITION

```

OBJECT CHARACTERISTICS CICS RELEASE = 0640
CEDA View PROGRAM( CACCIVS )
PROGram : CACCIVS
Group : gggg
Description : 'CICS VSAM SERVICE'
Language : C C0bo1 | Assembler | Le370 | C | Pl1
RELoad : No No | Yes
RESident : No No | Yes
USAge : Normal Normal | Transient
USEIpacopy : No No | Yes
Status : Enabled Enabled | Disabled
RS1 : 00 0-24 | Public
CEdf : Yes Yes | No
DATAlocation : Below Below | Any
EXECKey : User User | Cics
COncurrency : Quasirent Quasirent | Threadsafe
Api : Cicsapi Cicsapi | Openapi
REMOTE ATTRIBUTES
DYnamic : No No | Yes
REMOTESystem :
REMOTENAME :
Transid :
EXECutionset : Fullapi Fullapi | Dplsubset
JVM ATTRIBUTES
JVM : No No | Yes
JVMClass :
(Mixed Case) :
:
:
:
JVMProfile : DFHJVMPR (Mixed Case)
JAVA PROGRAM OBJECT ATTRIBUTES
Hotpool : No No | Yes

```

```

:
:
:
JVMPProfile : DFHJVMPR (Mixed Case)
JAVA PROGRAM OBJECT ATTRIBUTES
Hotpool : No No | Yes
TRANSACTION DEFINITION

OBJECT CHARACTERISTICS CICS RELEASE = 0640
CEDA View TRANSAction( EXV1 )
TRANSAction : EXV1
Group : CAC1
Description : 'TRANSACTION FOR CAC VSAM SERVICE'
PROGram : CACCIVS
TWasize : 00000 0-32767
PROFile : DFHCICST
PARTitionset :
STATUS : Enabled Enabled | Disabled
PRIMedsize : 00000 0-65520
TASKDATAloc : Below Below | Any
TASKDATAKey : User User | Cics
STOrageclear : No No | Yes
RUNaway : System System | 0 | 500-2700000
SHUTDOWN : Disabled Disabled | Enabled
ISolate : Yes Yes | No
Brexit :
DYnamic : No No | Yes
ROutable : No No | Yes
REMOTESystem :
REMOTENAME :
TRProf :
Localq : No | Yes
SCHEDULING
PRIORity : 001 0-255
TCClass : No No | 1-10
TRANClass : DFHTCL00
ALIASES
ALias :
TASKReq :
XTRanid :
TPName :
:
XTPname :
:
:
RECOVERY
DTImout : No No | 1-6800 (MMSS)
REStart : No No | Yes
SPurge : No No | Yes
TPUrges : No No | Yes
DUmp : Yes Yes | No
TRACe : Yes Yes | No
CONfdata : No No | Yes
OtstImout : No No | 0-240000 (HHMMSS)
INDOUBT ATTRIBUTES
ACtion : Backout Backout | Commit
WAIT : Yes Yes | No
WAITTime : 00 , 00 , 00 0-99 (Days,Hours,Mins)
INDoubt : Backout Backout | Commit | Wait
SECURITY
RESSec : No No | Yes
CMdsec : No No | Yes
Extsec : No No | Yes
TRANSec : 01 1-64
RS1 : 00 0-24 | Public

```

Example definitions for stored procedure access

PROGRAM DEFINITION

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0410
CEDA DEFINE PROGRAM( ppppppp )
  PROGRAM      : CACSP62
  GROUP       : gggg
  DESCRIPTION  ==>
  LANGUAGE    ==> Assembler CObol | Assembler | Le370 | C | PlI
                    | Rpg
  RELOAD     ==> No          No | Yes
  RESIDENT   ==> No          No | Yes
  USAGE      ==> Normal     Normal | Transient
  USELPACOPY ==> No          No | Yes
  STATUS     ==> Enabled    Enabled | Disabled
  RS1        : 00          0-24 | Public
  CEDF       ==> Yes        Yes | No
  DATALOCATION ==> Below    Below | Any
  EXECKEY    ==> User      User | Cics
REMOTE ATTRIBUTES
  REMOTESYSTEM ==>
  REMOTENAME  ==>
  TRANSID     ==>
  EXECUTIONSET ==> Fullapi  Fullapi | Dplsubset
```

PROGRAM DEFINITION

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0410
CEDA DEFINE PROGRAM( CACSPREM )
  PROGRAM      : CACSPREM
  GROUP       : gggg
  DESCRIPTION  ==>
  LANGUAGE    ==> Le370 CObol | Assembler | Le370 | C | PlI
                    | Rpg
  RELOAD     ==> No          No | Yes
  RESIDENT   ==> No          No | Yes
  USAGE      ==> Normal     Normal | Transient
  USELPACOPY ==> No          No | Yes
  STATUS     ==> Enabled    Enabled | Disabled
  RS1        : 00          0-24 | Public
  CEDF       ==> Yes        Yes | No
  DATALOCATION ==> Below    Below | Any
  EXECKEY    ==> User      User | Cics
REMOTE ATTRIBUTES
  REMOTESYSTEM ==>
  REMOTENAME  ==>
  TRANSID     ==>
  EXECUTIONSET ==> Fullapi  Fullapi | Dplsubset
```

TRANSACTION DEFINITION

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0410
CEDA DEFINE TRANSACTION( tttt )
  TRANSACTION : XASP
  GROUP       : gggg
  DESCRIPTION ==>
  PROGRAM     ==> CACSP62
  TWSIZE     ==> 00000          0-32767
  PROFILE    ==> DFHCICST
  PARTITIONSET ==>
  STATUS     ==> Enabled      Enabled | Disabled
  PRIMESIZE  : 00000          0-65520
  TASKDATALOC ==> Below      Below | Any
  TASKDATAKEY ==> User      User | Cics
  STORAGECLEAR ==> No        No | Yes
  RUNAWAY    ==> System     System | 0-2700000
  SHUTDOWN   ==> Disabled    Disabled | Enabled
```

ISolate	==> Yes	Yes No
REMOTE ATTRIBUTES		
DYnamic	==> No	No Yes
REMOTESystem	==>	
REMOTENAME	==>	
TRProf	==>	
Localq	==>	No Yes
SCHEDULING		
PRIOrity	==> 001	0-255
TClass	: No	No 1-10
TRANClass	==>	
ALIASES		
Alias	==>	
TASKReq	==>	
XTRanid	==>	
TPName	==>	
	==>	
XTPname	==>	
	==>	
	==>	
RECOVERY		
DTimeout	==> No	No 1-6800
INDoubt	==> Backout	Backout Commit Wait
REStart	==> No	No Yes
SPurge	==> No	No Yes
TPUrge	==> No	No Yes
DUmp	==> Yes	Yes No
TRACe	==> Yes	Yes No
COnfdata	==> No	No Yes
SECURITY		
RESec	==> No	No Yes
CMdsec	==> No	No Yes
Extsec	: No	
TRANSec	: 01	1-64
RS1	: 00	0-24 Public

FILE DEFINITION

OVERTYPE TO MODIFY CICS RELEASE = 0410

```

CEDA DEFINE File( CACIVP )
  File      : CACIVP
  Group     : gggg
  Description ==>
VSAM PARAMETERS
  DSName    ==> your.vsam.cluster.name.here
  Password  ==> PASSWORD NOT SPECIFIED
  Lsrpoolid ==> 1          1-8 | None
  DSNSharing ==> Allreqs  Allreqs | Modifyreqs
  STRings   ==> 001      1-255
  Nsrgroup  ==>
REMOTE ATTRIBUTES
  REMOTESystem ==>
  REMOTENAME ==>
  RECORDSize ==>          1-32767
  Keylength  ==>          1-255
INITIAL STATUS
  STatus    ==> Enabled   Enabled | Disabled | Unenabled
  Opentime  ==> Firstref  Firstref | Startup
  DISposition ==> Share   Share | Old
BUFFERS
  DATabuffers ==> 00002   2-32767
  Indexbuffers ==> 00001  1-32767
DATATABLE PARAMETERS
  Table     ==> No        No | Cics | User
  Maxnumrecs ==>          16-16777215
DATA FORMAT
  RECORDFormat ==> F    V | F
OPERATIONS

```

Add	==> Yes	No Yes
BRowse	==> <u>No</u>	No Yes
DElete	==> Yes	No Yes
REAd	==> <u>Yes</u>	Yes No
Update	==> <u>Yes</u>	No Yes
AUTO JOURNALLING		
JOurnal	==> No	No 1-99
JNLRead	==> None	None Updateonly Readonly All
JNLSYNRead	==> No	No Yes
JNLUpdate	==> No	No Yes
JNLAdd	==> None	None Before AFter AL1
JNLSYNWrite	==> Yes	Yes No
RECOVERY PARAMETERS		
RECOVery	==> None	None Backoutonly All
Fwdrecovlog	==> No	No 1-99
BAckuptype	==> Static	Static Dynamic
SECURITY		
RESsecnum	: 00	0-24 Public

CONNECTION DEFINITION (1 OF 2)

OVERTYPE TO MODIFY CICS RELEASE = 0410

CEDA DEFine Connection(ccc1)

Connection : **ccc1**

Group : **gggg**

DEscription ==>

CONNECTION IDENTIFIERS

Netname ==> **CACPPC00**

INDsys ==>

REMOTE ATTRIBUTES

REMOTESYSem ==>

REMOTEName ==>

REMOTESYSNet ==>

CONNECTION PROPERTIES

ACcessmethod	==> Vtam	Vtam IRc INdirect Xm
PRotocol	==> Appc	Appc Lu61 Exci
Conntype	==>	Generic Specific
SINGLEsess	==> No	No Yes
DATAstream	==> User	User 3270 SCs STRfield Lms
RECORDformat	==> U	U Vb
Queuelimit	==> No	No 0-9999
Maxqtime	==> No	No 0-9999

OPERATIONAL PROPERTIES

AUTOconnect	==> No	No Yes All
INService	==> Yes	Yes No

SECURITY

SEcurityname ==>

ATTachsec	==> Local	Local Identify Verify Persistent Mixidpe
-----------	-----------	---

BINDPassword : PASSWORD NOT SPECIFIED

BINDSecurity	==> No	No Yes
Usedfltuser	==> No	No Yes

RECOVERY

PSrecovery	==> Sysdefault	Sysdefault None
------------	----------------	-------------------

CONNECTION DEFINITION (2 OF 2)

OVERTYPE TO MODIFY CICS RELEASE = 0410

CEDA DEFine Connection(ccc2)

Connection : **ccc2**

Group : **gggg**

DEscription ==>

CONNECTION IDENTIFIERS

Netname ==> **CACPPC01**

INDsys ==>

REMOTE ATTRIBUTES

REMOTESYSem ==>

```

REMOTENAME ==>
REMOTESYSNET ==>
CONNECTION PROPERTIES
ACcesSMETHOD ==> Vtam          Vtam | IRc | INdirect | Xm
PRotocol      ==> Appc          Appc | Lu61 | Exci
Conntype      ==>              Generic | Specific
SIngleseSS    ==> No           No | Yes
DATastream    ==> User         User | 3270 | SCs | STRfield
                                   | Lms
RECORDformat  ==> U            U | Vb
Queuelimit    ==> No           No | 0-9999
Maxqtime      ==> No           No | 0-9999
OPERATIONAL PROPERTIES
AUtoconnect   ==> No           No | Yes | All
INService     ==> Yes         Yes | No
SECURITY
SEcurityname  ==>
ATTachsec     ==> Local        Local | Identify | Verify
                                   | Persistent | Mixidpe
BINDPassword  :                PASSWORD NOT SPECIFIED
BINDSecurity  ==> No           No | Yes
Usedfltuser   ==> No           No | Yes
RECOVERY
PSrecovery    ==> Sysdefault   Sysdefault | None

```

Tip: If you want CICS to verify user IDs and passwords as valid CICS users, set the ATTACHSEC parameter to Verify.

SESSIONS DEFINITION (1 OF 2)

```

OVERTYPE TO MODIFY                                CICS RELEASE = 0410
CEDA DEFINE Sessions( ssssss1 )
  Sessions      : sssssss1
  Group         : gggg
  DESCRIPTION   ==>
SESSION IDENTIFIERS
Connection     ==> ccc1
SESSName      ==>
NETnameq      ==>
M0dename      ==> MTLU62
SESSION PROPERTIES
Protocol       ==> Appc          Appc | Lu61 | Exci
MAXimum       ==> 001 , 000    0-999
RECEIVEPfx    ==>              1-999
RECEIVECount  ==>              1-999
SENDPfx       ==>              1-999
SENDCount     ==> 00256       1-30720
SENDSize      ==> 00256       1-30720
RECEIVESize   ==> 00256       1-30720
SESSPriority   ==> 000         0-255
Transaction   :
OPERATOR DEFAULTS
OPERId        :
OPERPriority   : 000            0-255
OPERRs1       : 0              0-24,...
OPERSecurity  : 1              1-64,...
PRESET SECURITY
USERId        ==>
OPERATIONAL PROPERTIES
Autoconnect   ==> No           No | Yes | All
INService     :
Buildchain    ==> Yes         Yes | No
USERArealen   ==> 000         0-255
IOarealen     ==> 00000 , 00000 0-32767
RELreq        ==> No           No | Yes
DIScreq       ==> No           No | Yes
NEPclass      ==> 000         0-255

```

```

RECOVERY
  RECOVOption ==> Sysdefault      Sysdefault | Clearconv
                                      | Releasesess| Uncondrel | None
  RECOVNotify   : None           None | Message | Transaction

```

SESSIONS DEFINITION (2 OF 2)

```

OVERTYPE TO MODIFY                                CICS RELEASE = 0410
CEDA DEFine Sessions( ssssss2 )
  Sessions      : sssssss2
  Group         : gggg
  Description   ==>
SESSION IDENTIFIERS
  Connection    ==> ccc2
  ESSName       ==>
  NETnameq     ==>
  M0dename     ==> MTLU62
SESSION PROPERTIES
  Protocol      ==> Appc           Appc | Lu61 | Exci
  MAXimum       ==> 001 , 000    0-999
  RECEIVEPfx   ==>
  RECEIVECount ==>                1-999
  SENDPfx      ==>
  SENDCount    ==>                1-999
  SENDSize     ==> 00256        1-30720
  RECEIVESize  ==> 00256        1-30720
  SESSPriority ==> 000          0-255
  Transaction   :
OPERATOR DEFAULTS
  OPERId       :
  OPERPriority : 000            0-255
  OPERRs1     : 0
  OPERSecurity : 1             0-24,...
                                   1-64,...
PRESET SECURITY
  USERId      ==>
OPERATIONAL PROPERTIES
  Autoconnect ==> No           No | Yes | All
  INservice   :
  Buildchain  ==> Yes         Yes | No
  USERArealen ==> 000         0-255
  IOarealen   ==> 00000 , 00000 0-32767
  RELreq      ==> No         No | Yes
  DIScreq     ==> No         No | Yes
  NEPclass    ==> 000         0-255
RECOVERY
  RECOVOption ==> Sysdefault      Sysdefault | Clearconv
                                      | Releasesess| Uncondrel | None
  RECOVNotify   : None           None | Message | Transaction

```

Field procedures

You use field procedures to transform data values from one format to another.

Field procedures are assigned to columns as conversion exits in the Classic Data Architect and metadata utility. The specified field procedure is invoked each time that the column is referenced.

The transformation that the field procedure performs on a value in a WHERE clause is called *field-encoding*. The same routine undoes the transformation when values are retrieved; that operation is called *field-decoding*. You can also use the Classic Data Architect to specify the SQL data types that describe column values.

You can use the sample field procedures that are provided in SCACSAMP to assist you.

Restriction: The field-decoding function must be the exact inverse of the field-encoding function. For example, if a routine encodes HYDROGEN to 01, it must decode 01 to HYDROGEN.

Field procedure setup

You define field procedures to convert data values from one format to another when SQL requests reference a column.

To create a field procedure for a column, use the Classic Data Architect. In the column properties view, you can define a conversion exit in the column information. The conversion exit is included in the DDL that creates the metadata catalog.

Alternatively, you can specify a field procedure in a column definition by using the WITH CONVERSION parameter.

After you define a field procedure, place the load module in a library that is referenced by the server STEPLIB DD statement. The field procedure routine must be written and linked as AMODE 31, RMODE ANY. Any storage or other resources that the field procedure allocates must be freed each time that the field procedure is called.

Field procedures and data transformations

With basic knowledge about how to use field procedures, you can perform data transformations during query processing.

Field procedures are conversion routines that translate between data items in a database record and their corresponding SQL data types. These procedures are used when the underlying data type in a record does not match the SQL data type to be returned in SQL requests. These examples of situations show how to use field procedures:

- Database fields that are transformed to reverse or change sorting order.
For example, decimal date fields are often stored in *nines compliment* format in order to reverse the sorting order of dates such that the most recent date comes first. The nines complement is the amount necessary to complete a number up to the highest number in the number system. In the decimal system, this is the difference between a given number and all 9s. The nines complement of 254 is 999 minus 254, or 745.
- Abbreviations or coded tables. Abbreviations or codes for application data items can be used to save space in database records. For example, the value 01 can be stored in place of the element name hydrogen.
- Encryption. For security purposes, password fields can be stored in a database record in an encrypted format.

A query invokes a field procedure to either encode or decode the data value of a column. Field encoding takes place when either of these actions occur:

- The field is compared to a value in the WHERE clause by using the equal or not equal comparison operators.
- A column is assigned a value with an SQL INSERT or UPDATE statement.

Field-decoding takes place when:

- A stored value is to be compared to any operator other than equal or not equal.

- A query retrieves the contents of a field in a select list. The value is field-decoded back into its original string value.

A field procedure is never invoked to determine if the field is NULL or NOT NULL.

Exception: The use of a key column with a field procedure in a WHERE clause can cause the data connector to scan the database. This situation can happen if the column is the beginning of the key and the comparison operator is not the equal comparison operator.

Control blocks for field procedures

You use control blocks to communicate parameters to a field procedure and provide information about working storage, operations, errors, addresses, data types, and other value attributes.

Field procedure parameter list (FPPL)

The field procedure parameter list points to the addresses of these areas: the work area, the field procedure information block, the column value descriptor, and the field descriptor.

Register 1 points to the field procedure parameter list on entry to a field procedure. The parameter list, in turn, contains the addresses of other areas, as shown in the following figure. The mapping macro DSNDFPPBFPPL describes the areas that the FPPL points to.

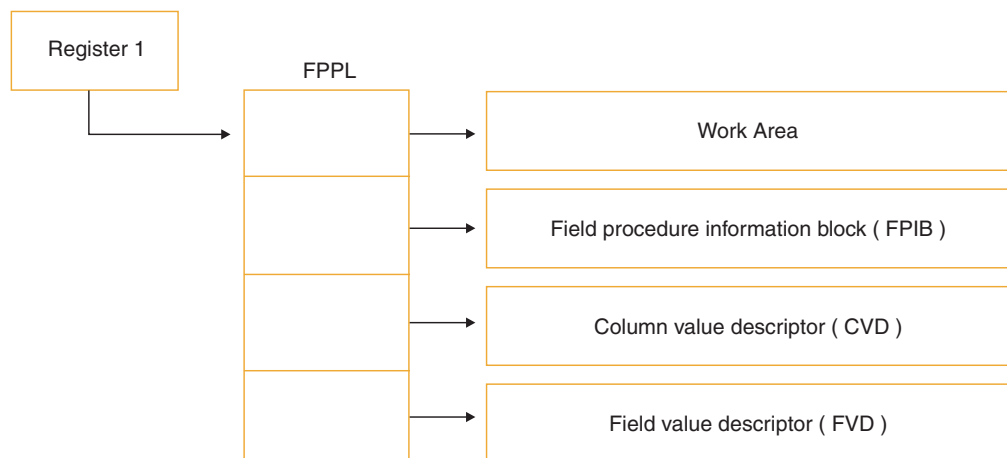


Figure 24. Field procedure parameter list

Work area

The work area is 512 bytes of contiguous, uninitialized storage that a field procedure can use as working storage.

If 512 bytes is sufficient for your operations, your field-definition operation does not need to change the value that is supplied by the query processor. If less than 512 bytes is required, the field-definition can return a smaller value. If your program requires more than 512 bytes, the field procedure must acquire the necessary storage.

Field procedure information block (FPIB)

The field procedure information block communicates general information to a field procedure, including the operation that is performed, the size of the work area, and error message addressing.

The following table displays the format of the field procedure information block:

Table 94. Format of FPIB

Value description	Hex offset	Integer type	Contents
FPBFICODE	0	Signed 2-byte integer	Function code: <ul style="list-style-type: none">• 0 (field-encoding)• 4 (field-decoding)
FPBWKLN	2	Signed 2-byte integer	512, length of work area in bytes
FPBSORC	4	Signed 2-byte integer	Reserved
FPBRTNC	6	Character, 2 bytes	Return code that is set by the field procedure
FPBRSNC	8	Character, 4 bytes	Reason code that is set by the field procedure
FPBTOKP	C	Address	Address of a 40-byte area in which to return an error message

Error messages in the FPBTOKP are not returned to the client application when a field procedure reports an error. Error messages are, however, logged in the server log for debugging purposes.

Value descriptors

Value descriptors describe the data type and other attributes of a column value or field value.

During field-encoding and field-decoding, the decoded column value and the encoded field value are described by the column value descriptor (CVD) and the field value descriptor (FVD):

Column value descriptor (CVD)

Contains a description of a column value and, if appropriate, the value itself. During field-encoding, the CVD describes the value to be encoded. During field-decoding, the CVD describes the decoded value to be supplied by the field procedure.

Field value descriptor (FVD)

Contains a description of a field value and, if appropriate, the value itself. During field-encoding, the FVD describes the encoded value to be supplied by the field procedure. During field-decoding, the FVD describes the value to be decoded.

The value descriptors have the format shown in the following table:

Table 95. Format of value descriptors

Value descriptor	Hex offset	Integer type	Contents
FPVDTYP	0	Signed 2-byte integer	Data type of the value: <ul style="list-style-type: none"> • 0 (INTEGER) • 4 (SMALLINT) • 8 (FLOAT) • 12 (DECIMAL) • 16 (CHAR) • 20 (VARCHAR) • 24 (GRAPHIC) • 28 (VARGRAPHIC)
FPVDVLEN	2	Signed 2-byte integer	<ul style="list-style-type: none"> • For a varying-length string value, its maximum length • For a decimal number value, its precision (byte 1) and scale (byte 2) • For any other value, its length <p>For GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC data types, the length is specified in bytes.</p>
FPVDVALE	4	None	The value of the column or field. If the value is a varying-length string, the first halfword is the actual length of the value in bytes. This field is not present in a CVD, or in an FVD that is used as input to the field-definition operation. An empty varying-length string has a length of zero with no data following.

Field-encoding (function code 0)

Your field procedure performs encoding operations when you specify field-encoding function code 0 in the FPBFCD field of the field procedure information block. The information that follows describes the input and required output of field-encoding operations.

On ENTRY, the registers contain the following information:

Table 96. Field encoding on entry

Register	Contents
1	Address of the field procedure parameter list (FPPL)
0, 2 through 12	Values that must be restored on exit
13	Address of the calling program save area that must be restored on exit
14	Return address
15	Address of entry point of exit routine

The work area is uninitialized.

The FPIB contains the following information:

Table 97. FPIB fields and contents

Field	Contents
FPBFCD	0, the function code
FPBWKLN	512, the length of the work area in bytes

The CVD contains the following information:

Table 98. CVD fields and contents

Field	Contents
FPVDTYPE	The numeric code for the data type of the column value.
FPVDVLEN	The length of the column value.
FPVDVALE	The column value; if the value is a varying-length string, the first halfword contains its length.

The FVD contains the following information:

Table 99. FVD fields and contents

Field	Contents
FPVDTYPE	The numeric code for the data type of the field value
FPVDVLEN	The length of the field value
FPVDVALE	Uninitialized area with a value of FPVDVLEN

On EXIT, the registers contain the following information:

Table 100. EXIT registers and contents

Register	Contents
1	Address of the field procedure parameter list (FPPL).
0, 2 through 14	The values that they contained on entry.
15	The integer zero. An error in processing is indicated by the value in FPBRTNC.

The FVD must contain the encoded field value in field FPVDVALE. If the value is a varying-length string, the first halfword must contain its length.

The FPIB can contain the following information:

Table 101. FPIB fields and contents

Field	Contents
FPBRTNC	Return code. The character "0" that is followed by a space indicates success. Anything other than "0" indicates an error.
FPBRSNC	An optional, 4-byte character reason code, which is defined by the field procedure, blanks if no reason code is provided.
FPBTOKP	Address of a 40-byte area that contains an error message when an error message is detected.

Errors that are signaled by a field procedure result in SQLCODE -681 and are written to the error log, which is set in the SQL Communications Area (SQLCA). FPBRTNC is the return code; FPBRSNCD is the reason code; and a 40-byte error field for the specific error message is FPBTOKP.

Field-decoding (function code 4)

Your field procedure performs decoding operations when you specify field-encoding function code 4 in the FPBFCD field of the field procedure information block. The information that follows describes the input and required output of field-decoding operations.

On ENTRY, the registers contain the following information:

Table 102. ENTRY registers and fields

Register	Contents
1	Address of the field procedure parameter list (FPPL)
0, 2 through 12	Values that must be restored on exit
13	Address of the calling program save area that must be restored on exit
14	Return address
15	Address of entry point of the exit routine

The work area is uninitialized.

The FPIB contains the following information:

Table 103. FPIB fields and contents

Field	Contents
FPBFCODE	4, the function code
FPBWKLN	512, the length of the work area in bytes.

The CVD contains the following information:

Table 104. CVD fields and contents

Field	Contents
FPVDTYPE	The numeric code for the data type of the column value
FPVDVLEN	The length of the column value
FPVDVALE	Uninitialized area with a value of FPVDVLEN

The FVD contains the following information:

Table 105. FVD fields and contents

Field	Contents
FPVDTYPE	The numeric code for the data type of the field value
FPVDVLEN	The length of the field value
FPVDVALE	The field value; if the value is a varying-length string, the first halfword contains its length

On EXIT, the registers contain the following information:

Table 106. EXIT registers and contents

Register	Contents
1	Address of the field procedure parameter list (FPPL).
0, 2 through 14	The values that they contained on entry.
15	The integer zero. An error in processing is indicated by the value in FPBRINC.

The CVD must contain the decoded column value in field FPVDVALE. If the value is a varying-length string, the first halfword must contain its length.

The FPIB can contain the following information:

Table 107. FPIB fields and contents

Field	Contents
FPBRTNC	Return code. The character "0" that is followed by a space indicates success. Anything other than "0" indicates an error.
FPBRSNC	An optional, 4-byte character reason code, defined by the field procedure; blanks if no reason code is provided.
FPBTOKP	Address of a 40-byte area in which to return an error message.

Errors that are signaled by a field procedure result in SQLCODE -681. The errors are written to the error log, which is set in the SQL Communications Area (SQLCA). FPBRTNC is the return code; FPBRSNCD is the reason code; and a 40-byte error field for the specific error message is FPBTOKP.

Sample field procedures

Use the sample field procedures that are provided in SCACSAMP to guide you when you create your own field procedures.

The following table lists each member and its description:

Table 108. SCACSAMP members and descriptions

SCACSAMP member name	Description
CACFP001	A nines complement field procedure that converts decimal or zoned decimal data
CACFP999	Creates site-specific field procedures

CACFP001 - Sample field procedure

Use this nines complement field procedure with decimal or zoned decimal data to force decimal date fields into descending sort order.

The following mapping combinations are supported:

- DATATYPE P|UP USE AS DECIMAL(p,s) in which decimal fields are mapped as SQL DECIMAL
- DATATYPE C|UC USE AS CHAR(n) in which zoned decimal fields are mapped as SQL CHAR
- DATATYPE C|UC USE AS DECIMAL(p,s) in which zoned decimal fields are mapped as SQL DECIMAL

CACFP999 - Sample field procedure

Use this generic field procedure to create your own site-specific field procedures and transform supported data types.

This field procedure copies field data directly to column data as-is and supports the following data types:

- SMALLINT
- INTEGER
- DECIMAL
- FLOAT
- CHARACTER

This field procedure does not perform any data conversion.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS," without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies, and have been used at least once in this information:

- Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc., in the United States, other countries, or both.
- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Privacy policy considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies

and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, See IBM's Privacy Policy at <http://www.ibm.com/privacy> and IBM's Online Privacy Statement at <http://www.ibm.com/privacy/details> the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at <http://www.ibm.com/software/info/product-privacy>.



Printed in USA