z/OS Basic Skills Information Center

Online Workloads for z/OS

z/OS Basic Skills Information Center

Online Workloads for z/OS

Note

Before using this information and the product it supports, read the information in "Notices" on page 77.

This edition applies to z/OS (product number 5694-A01).

Contents

Online workloads for z/OS: an Introduction

In this part, we examine the major categories of online or **interactive** workloads performed on z/OS®, such as transaction processing, database management, and Web-serving. This sections contains discussions of several popular middleware products, including CICS®, IMS™, DB2®, and WebSphere®.

z/OS is a base for many middleware products and functions. It is commonplace to run a variety of diverse middleware functions, with multiple instances of some. The routine use of wide-ranging workloads (mixtures of batch, transactions, Web serving, database queries and updates, and so on) is characteristic of z/OS.

Typical z/OS middleware includes:
- Transaction managers
- Database systems
- Web servers
- Message queueing and routing functions.

Customer Information Control System (CICS) is a transaction manager. It runs applications on your behalf online, by request, at the same time as many other users may be submitting requests to run the same applications, using the same files and programs. CICS manages the sharing of resources, integrity of data, and prioritization of execution, with fast response. CICS applications are traditionally run by submitting a transaction request. Execution of the transaction consists of running one or more application programs that implement the required function.

Information Management System (IMS) consists of three components: the Transaction Manager (TM), the Database Manager (DB), and a set of system services that provide common services to the other two components. IMS DB is a hierarchical database manager. In a hierarchical database, data is organized in the form of a hierarchy (pyramid). Data at each level of the hierarchy is related to, and in some way dependent on, data at the higher level of the hierarchy.

DB2 is a relational database manager. In a relational database, the most fundamental structure is the table with columns and rows. DB2 implements relational principles, such as primary keys, referential integrity, a language to access the database (SQL), nulls, and normalized design.

WebSphere is IBM's integration software platform. It includes the entire middleware infrastructure— servers, services, and tools—needed to write, run, and monitor 24x7 industrial-strength, on demand Web applications and cross-platform, cross-product solutions. WebSphere provides reliable, flexible, and robust integration software.

A middleware product often includes an application programming interface (API). In some cases, applications are written to run completely under the control of this middleware API, while in other cases it is used only for unique purposes. Some examples of mainframe middleware APIs include:
- The WebSphere suite of products, which provides a complete API that is portable across multiple operating systems. Among these products, WebSphere MQ provides cross-platform APIs and inter-platform messaging.
- The DB database management product, which provides an API (expressed in the SQL language) that is used with many different languages and applications.

A Web server is considered to be middleware and Web programming (Web pages, CGIs, and so forth) is largely coded to the interfaces and standards presented by the Web server instead of the interfaces presented by the operating system. Java™ is another example in which applications are written to run under a Java Virtual Machine (JVM) and are largely independent of the operating system being used.

Chapter 1. Transaction management systems on z/OS

To expand your knowledge of mainframe workloads, you must understand the role of mainframes in today's online world.

This section introduces concepts and terminology for transactional processing, and presents an overview of the major types of system software used to process online workloads on the mainframe. In this section, we focus on two of the most widely used transaction management products for z/OS: CICS and IMS.

Online processing on the mainframe

In other sections, we discussed the possibilities of batch processing–but those are not the only applications running on z/OS and the mainframe. Online applications also run on z/OS, as we show in this section. We also describe what online, or interactive, applications are and discuss their common elements in the mainframe environment.

We examine databases, which are a common way of storing application data. Databases make development easier–especially in the case of a relational database management system (RDBMS)–by removing the burden from the programmer organizing and managing the data. Later in this section, we discuss several widely used transaction management systems for mainframe-based enterprises.

We begin with the example of a travel agency with a requirement common to many mainframe customers: Provide customers with more immediate access to services and exploit the benefits of Internet-based commerce.

Example of global online processing–the new big picture

A big travel agency has relied on a mainframe-based batch system for many years. Over the years, the agency's customers have enjoyed excellent service, and the agency has continuously improved its systems.

When the business was begun, their IT staff designed some applications to support the agency's internal and external processes: Employee information, customer information, contacts with car rental companies, hotels all over the world, scheduled flights of airlines, and so on. At first, these applications were updated periodically by batch processing.

This kind of data is not static, however, and has become increasingly prone to frequent change. Because prices, for example, change frequently, it became more difficult over time to maintain current information. The agency's customers wanted their information **now** and that was not always possible through fixed intervals of batch updates (consider the time difference between Asia, Europe, and America).

If these workloads were to be done through traditional mainframe batch jobs, it would mean a certain time lapse between the reception of the change and the actual update. The agency needed a way to update small amounts of data provided in bits and pieces--by phone, fax, or e-mail–the instant that changes occur (Figure 1 on page 2).

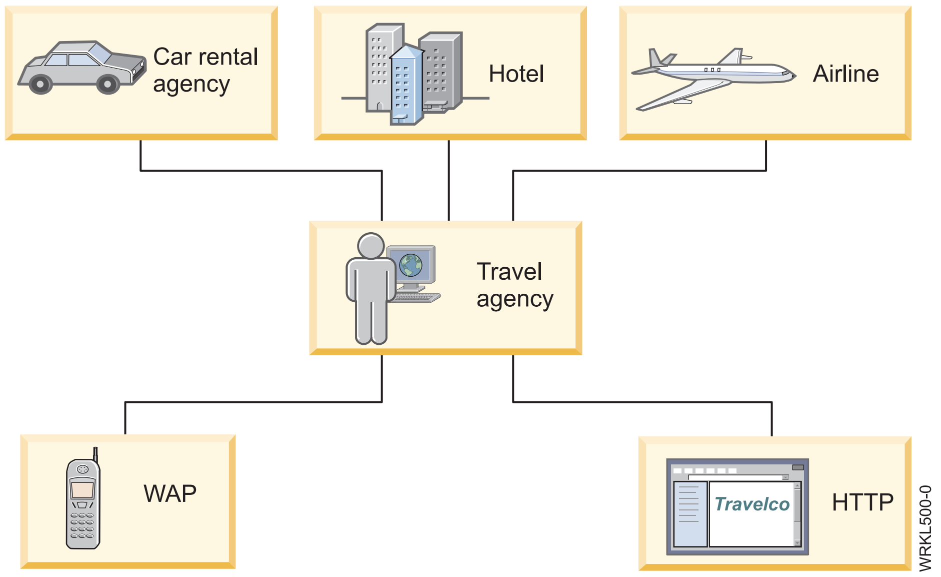


Figure 1. A practical example

Therefore, the agency IT staff created some new applications. Since changes need to be immediately reflected to the applications's end-users, the new applications are transactional in nature. The applications are called transaction or interactive applications because changes in the system data are effective immediately.

The travel agency contacted its suppliers to see what could be done. They needed a way to let the computers talk to each other. Some of the airlines were also working on mainframes, others were not, and everybody wanted to keep their own applications.

Eventually, they found a solution! It made communicating easy: you could just ask a question and some seconds later get the result–great stuff.

More innovations were required because the customers also evolved. The personal computer got into their homes, so they wanted to see travel possibilities through the Internet. Some customers used their mobile computers as a wireless access point (WAP).

Transaction systems for the mainframe

Transactions occur in everyday life, for example, when you exchange money for goods and services or do a search on the Internet. A transaction is an exchange, usually a request and response, that occurs as a routine event in running the day-to-day operations of an organization.

Transactions have the following characteristics:

- Small amount of data is processed and transferred per transaction
- Large numbers of users
- Are executed in large numbers.

What are transaction programs?

A business transaction is a self-contained business deal. Some transactions involve a short conversation (for example, an address change). Others involve multiple actions that take place over an extended period (for example, the booking of a trip, including car, hotel, and airline tickets).

A single transaction might consist of many application programs that carry out the processing needed. Large-scale transaction systems (such as the IBM® CICS product) rely on the **multitasking** and **multithreading** capabilities of z/OS to allow more than one task to be processed at the same time, with each task saving its specific variable data and keeping track of the instructions each user is executing.

Multitasking is essential in any environment in which thousands of users can be logged on at the same time. When a multitasking transaction system receives a request to run a transaction, it can start a new task that is associated with one instance of the execution of the transaction; that is, one execution of a transaction, with a particular set of data, usually on behalf of a particular user at a particular terminal. You might also consider a task to be analogous to a UNIX® thread. When the transaction completes, the task is ended.

Multithreading allows a single copy of an application program to be processed by several transactions concurrently. Multithreading requires that all transactional application programs be reentrant; that is, they must be serially reusable between entry and exit points. Among programming languages, reentrance is ensured by a **fresh** copy of working storage section being obtained each time the program is invoked.

What is a transaction system?

Transaction systems must be able to support a high number of concurrent users and transaction types.

Figure 2 on page 4 shows the main characteristics of a transaction system. Before the advent of the Internet, a transaction system served hundreds or thousands of **terminals** with dozens or hundreds of transactions per second. This workload was rather predictable both in transaction rate and mix of transactions.

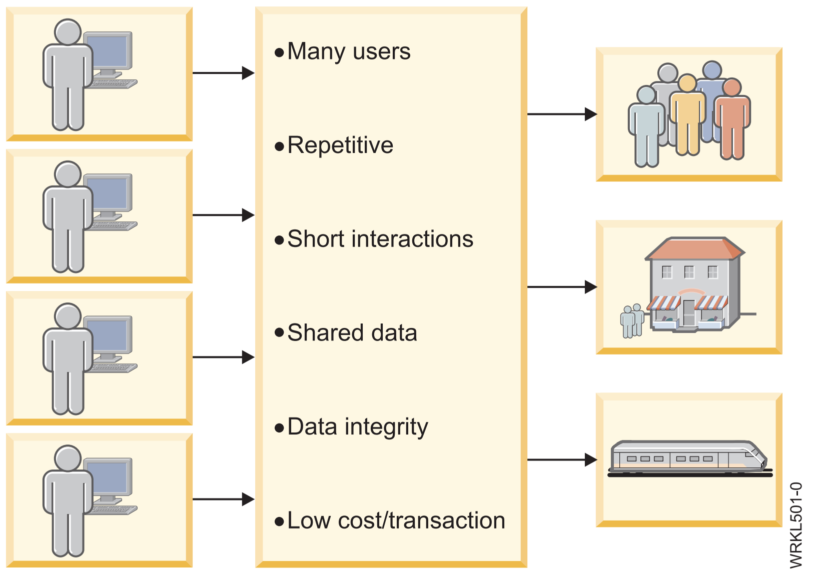


Figure 2. Characteristics of a transaction system

Transaction systems must be able to support a high number of concurrent users and transaction types.

One of the main characteristics of a transaction or online system is that the interactions between the user and the system are very brief. Most transactions are executed in short time periods--one second, in some cases. The user will perform a complete business transaction through short interactions, with immediate response time required for each interaction. These are mission-critical applications; therefore, continuous availability, high performance, and data protection and integrity are required.

Online transaction processing (OLTP) is transaction processing that occurs interactively; it requires:

- Immediate response time
- Continuous availability of the transaction interface to the end user
- Security
- Data integrity.

Online transactions are familiar to many people. Some examples include:

- ATM transactions such as deposits, withdrawals, inquiries, and transfers
- Supermarket payments with debit or credit cards
- Buying merchandise over the Internet.

In fact, an online system has many of the characteristics of an operating system:

- Managing and dispatching tasks
- Controlling user access authority to system resources
- Managing the use of memory
- Managing and controlling simultaneous access to data files
- Providing device independence.

What are the typical requirements of a transaction system?

A transaction system must comply with atomicity, consistency, isolation, and durability.

In a transaction system, transactions must comply with four primary requirements known jointly by the mnemonic A-C-I-D or ACID:

- Atomicity. The processes performed by the transaction are done as a whole or not at all.
- Consistency. The transaction must work only with consistent information.
- Isolation. The processes coming from two or more transactions must be isolated from one another.
- Durability. The changes made by the transaction must be permanent.

Usually, transactions are initiated by an end user who interacts with the transaction system through a terminal. In the past, transaction systems supported only terminals and devices connected through a teleprocessing network. Today, transaction systems can serve requests submitted in any of the following ways:

- Web page
- Remote workstation program
- Application in another transaction system
- Triggered automatically at a predefined time.

What is commit and roll back?

In transaction systems, **commit and roll back** refers to the set of actions used to ensure that an application program either makes **all** changes to the resources represented by a single unit of recovery (UR), or makes **no** changes at all. The two-phase commit protocol provides commit and rollback. It verifies that either all changes or no changes are applied even if one of the elements (like the application, the system, or the resource manager) fails. The protocol allows for restart and recovery processing to take place after system or subsystem failure.

The two-phase commit protocol is initiated when the application is ready to commit or back out its changes. At this point, the coordinating recovery manager, also called the **syncpoint manager**, gives each resource manager participating in the unit of recovery an opportunity to vote on whether its part of the UR is in a consistent state and can be committed. If all participants vote YES, the recovery manager instructs all the resource managers to commit the changes. If any of the participants vote NO, the recovery manager instructs them to back out the changes. This process is usually represented as two phases.

In phase 1, the application program issues the syncpoint or rollback request to the syncpoint coordinator. The coordinator issues a PREPARE command to send the initial syncpoint flow to all the UR agent resource managers. In response to the PREPARE command, each resource manager involved in the transaction replies to the syncpoint coordinator stating whether it is ready to commit or not.

When the syncpoint coordinator receives all the responses back from all its agents, phase 2 is initiated. In this phase the syncpoint coordinator issues the commit or rollback command based on the previous responses. If any of the agents responded with a negative response, the syncpoint initiator causes **all** of the syncpoint agents to roll back their changes.

The instant when the coordinator records the fact that it is going to tell all the resource managers to either commit or roll back is known as the **atomic instant**. Regardless of any failures after that time, the coordinator assumes that all changes will either be committed or rolled back. A syncpoint coordinator usually logs the decision at this point. If any of the participants abnormally end (or abend) after the atomic instant, the abending resource manager must work with the syncpoint coordinator, when it restarts, to complete any commits or rollbacks that were in process at the time of the abend.

On z/OS, the primary syncpoint coordinator is called Resource Recovery Services (RRS). Also, the IBM transaction manager product, CICS, includes its own built-in syncpoint coordinator.

During the first phase of the protocol, the agents do not know whether the syncpoint coordinator will commit or roll back the changes. This time is known as the **indoubt** period. The UR is described as having a particular state depending on what stage it is at in the two-phase commit process:

- Before a UR makes any changes to a resource, it is described as being **In-reset**.
- While the UR is requesting changes to resources, it is described as being **In-flight**.
- Once a commit request has been made (Phase 1), it is described as being **In-prepare**.
- Once the syncpoint manager has made a decision to commit (phase 2 of the two-phase commit process), it is **In-commit**.
- If the syncpoint manager decides to back out, it is **In-backout**.

Figure 3 illustrates the two-phase commit.

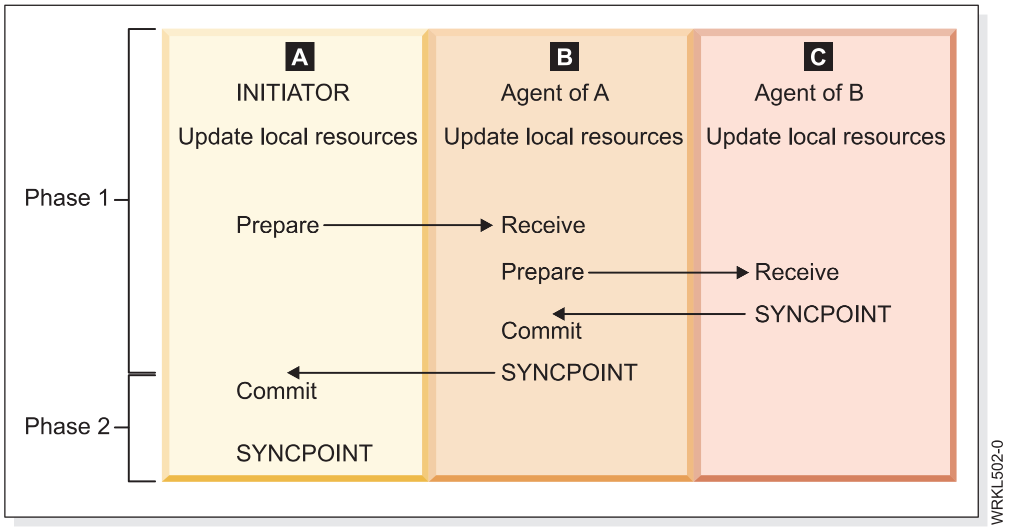


Figure 3. Two-phase commit

Most widely used transaction management systems on z/OS, such as CICS or IMS, support two-phase commit protocols. CICS, for example, supports full two-phase commit in transactions with IMS and the DB2 database management system, and supports two-phase commit across distributed CICS systems.

There are many restrictions imposed on application developers attempting to develop new applications that require updates in many different resource managers, perhaps across a number of systems. Many of these new applications use technologies like DB2 stored procedures and Enterprise Java Beans, and use client attachment facilities of CICS or IMS that do not support two-phase commit. If any of these resource managers are used by an application to update resources, it is not possible to have a global coordinator for the syncpoint.

The lack of a global syncpoint coordinator might influence an application design for the following reasons:

- The application is not capable of having complex and distributed transactions if not all of the resource managers are participating in the two-phase commit protocol.
- The application cannot be designed as a single application (or unit of recovery) across multiple systems (except for CICS).

The application programmer would have to program around these limitations. For example, the programmer could limit the choice of where to put the business data to ensure that all the data could be committed in a single unit of recovery.

Also, these limitations could affect the recoverability of the protected resources or their integrity in case of a failure of one of the components, because resource managers have no way to either commit or roll back the updates.

What is CICS?

CICS stands for “Customer Information Control System.” It is a general-purpose transaction processing subsystem for the z/OS operating system. CICS provides services for running an application online, by request, at the same time as many other users are submitting requests to run the same applications, using the same files and programs.

CICS manages the sharing of resources, the integrity of data and prioritization of execution, with fast response. CICS authorizes users, allocates resources (real storage and cycles), and passes on database requests by the application to the appropriate database manager (such as DB2). We could say that CICS acts like, and performs many of the same functions as the z/OS operating system.

A **CICS application** is a collection of related programs that together perform a business operation, such as processing a travel request or preparing a company payroll. CICS applications execute under CICS control, using CICS services and interfaces to access programs and files.

CICS applications are traditionally run by submitting a transaction request. Execution of the transaction consists of running one or more application programs that implement the required function. In CICS documentation you may find CICS application programs sometimes simply called “programs,” and sometimes the term “transaction” is used to imply the processing done by the application programs.

CICS applications can also take the form of Enterprise Java Beans. You can find out more about this form of programming in the section on Java applications in CICS in the CICS Information Center.

CICS in a z/OS system

In a z/OS system, CICS provides a layer of function for managing transactions, while the operating system remains the final interface with the computer hardware. CICS essentially separates a particular kind of application program (namely, online applications) from others in the system, and handles these programs itself.

When an application program accesses a terminal or any device, for example, it doesn't communicate directly with it. The program issues commands to communicate with CICS, which communicates with the needed access methods of the operating system. Finally, the access method communicates with the terminal or device.

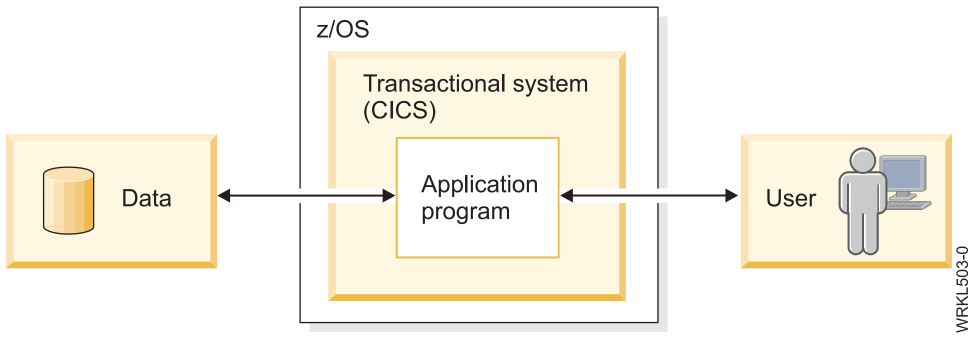


Figure 4. Transactional system and the operating system

A z/OS system might have multiple copies of CICS running at one time. Each CICS starts as a separate z/OS address space. CICS provides an option called muti-region operation (MRO), which enables the separation of different CICS functions into different CICS regions (address spaces); so a specific CICS address space (or more) might do the terminal control and will be named terminal owning region (TOR). Other possibilities include application-owning regions (AORs) for applications and file-owning regions (FORs) for files.

CICS programs, transactions and tasks

CICS allows you to keep your application logic separate from your application resources. To develop and run CICS applications, you need to understand the relationship between CICS programs, transactions, and tasks.

These terms are used throughout CICS publications and appear in many commands:

Transaction

A transaction is a piece of processing initiated by a single request. This is usually from an end user at a terminal, but might also be made from a Web page, from a remote workstation program, from an application in another CICS system, or triggered automatically at a predefined time. The *CICS Internet Guide* and the *CICS External Interfaces Guide* describe different ways of running CICS transactions.

A CICS transaction is given a 4-character name, which is defined in the program control table (PCT).

Application program

A single transaction consists of one or more **application programs** that, when run, carry out the processing needed.

However, the term **transaction** is used in CICS to mean both a single event and all other transactions of the same type. You describe each transaction type to CICS with a **transaction resource definition**. This definition gives the transaction type a name (the transaction identifier or TRANSID) and tells CICS several things about the work to be done, such as what program to invoke first and what kind of authentication is required throughout the execution of the transaction.

You run a transaction by submitting its TRANSID to CICS. CICS uses the information recorded in the TRANSACTION definition to establish the correct execution environment, and starts the first program.

Unit of work
The term transaction is now used extensively in the IT industry to describe a unit of recovery or what CICS calls a **unit of work**. This is typically a complete operation that is recoverable; it can be committed or backed out as an entirety as a result of a programmed command or system failure. In many cases, the scope of a CICS transaction is also a single unit of work, but you should be aware of the difference in meaning when reading non-CICS publications.

Task You will also see the word **task** used extensively in CICS publications. This word also has a specific meaning in CICS. When CICS receives a request to run a transaction, it starts a new task that is associated with this one instance of the execution of the transaction–that is, one execution of a transaction, with a particular set of data, usually on behalf of a particular user at a particular terminal. You can also consider it analogous to a **thread**. When the transaction completes, the task is terminated.

Using programming languages with CICS

You can use COBOL, OO COBOL, C, C++, Java, PL/I, or Assembler language to write CICS application programs to run on z/OS. Most of the processing logic is expressed in standard language statements, but you use CICS commands, or the Java and C++ class libraries, to request CICS services.

Most of the time, you use the CICS command level programming interface, EXEC CICS. This is the case for COBOL, OO COBOL, C, C++, PL/I and assembler programs. These commands are defined in detail in the *CICS Application Programming Reference*.

Programming in Java with the JCICS class library is described in the section on Java applications in the CICS component of the CICS Information Center.

Programming in C++ with the CICS C++ classes is described in the CICS C++ OO Class Libraries documentation.

CICS conversational and pseudo-conversational programming

In CICS, when the programs being executed enter into a conversation with the user, it is called a **conversational transaction**. A non-conversational transaction, by contrast, processes one input, responds, and ends (disappears). It never pauses to read a second input from the terminal, so there is no real conversation.

There is a technique in CICS called pseudo-conversational processing, in which a series of non-conversational transactions gives the appearance (to the user) of a single conversational transaction. No transaction exists while the user waits for input; CICS takes care of reading the input when the user gets around to sending it. Figure 5 on page 10 and Figure 6 on page 11 show different types of conversation in an example of a record update in a banking account.

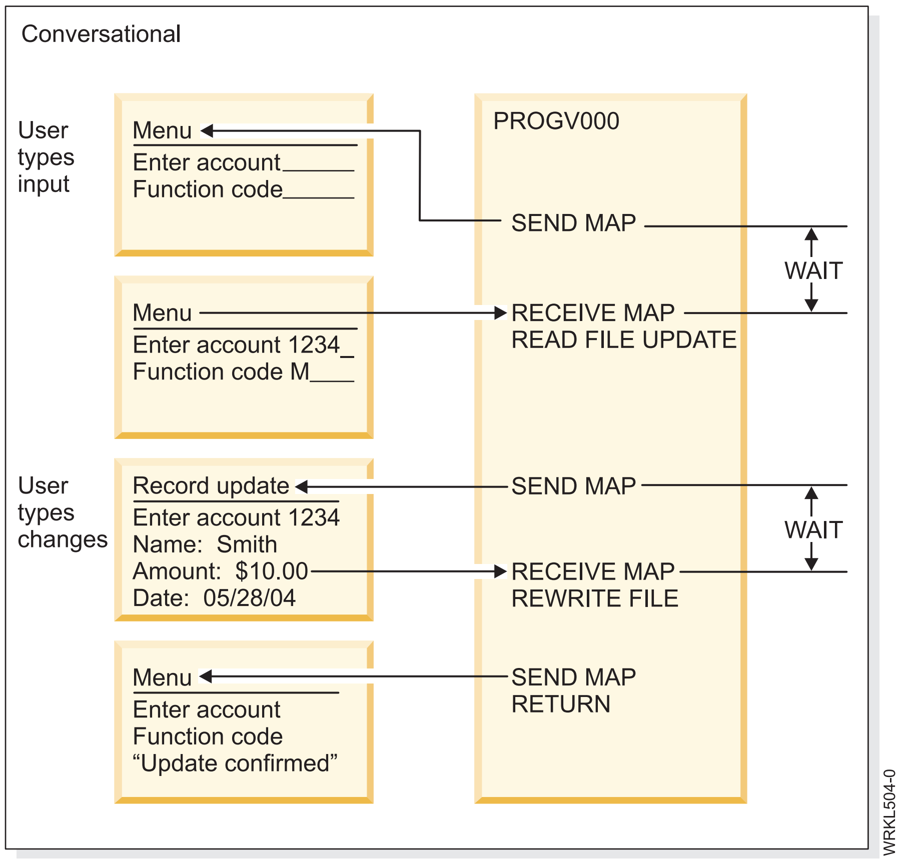


Figure 5. Example of a conversational transaction

In a conversational transaction, programs hold resources while waiting to receive data. In a pseudo-conversational model, no resources are held during these waits (Figure 6 on page 11).

More information about these topics can be found in *CICS Application Programming Guide*.

Pseudo-conversational A series of non-conversational transactions appears to the user as a conversation

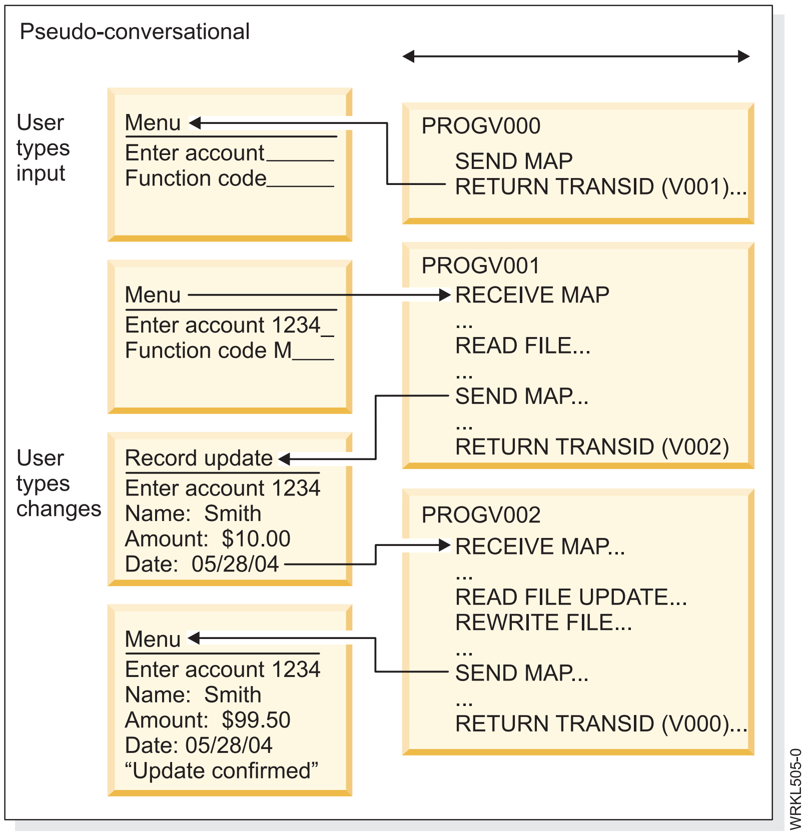


Figure 6. Example of a pseudo-conversational transaction

CICS programming commands

The general format of a CICS command is EXECUTE CICS (or EXEC CICS) followed by the name of the command and possibly one or more options.

You can write many application programs using the CICS command-level interface without any knowledge of, or reference to, the fields in the CICS control blocks and storage areas. However, you might need to get information that is valid outside the local environment of your application program.

When you need a CICS system service, for example when reading a record from a file, you just include a CICS command in your code. In COBOL, for example, CICS commands look like this:

```
EXEC CICS function option option ... END-EXEC.
```

The "function" is the action you want to perform. Reading a file is **READ**, writing to a terminal is **SEND**, and so on.

An "option" is some specification that's associated with the function. Options are expressed as keywords. For example, the options for the **READ** command include **FILE**, **RIDFLD**, **UPDATE**, and others. **FILE** tells CICS which file you want to read, and is always followed by a value indicating or pointing to the file name. **RIDFLD** (record identification field, that is, the key) tells CICS which record and likewise needs a value. The **UPDATE** option, on the other hand, simply means that you intend to change the record, and it doesn't take any value. So, to read with intent to modify, a record from a file known to CICS as ACCTFIL, using a key that we stored in working storage as ACCTC, we issued the command shown in Figure 7.

```
EXEC CICS
READ FILE('ACCTFIL')
RIDFLD(ACCTC) UPDATE ...
END-EXEC.
```

Figure 7. CICS command example

You can use the ADDRESS and ASSIGN commands to access such information. For programming information about these commands, see *CICS Application Programming Reference*. When using the ADDRESS and ASSIGN commands, various fields can be read but should not be set or used in any other way. This means that you should not use any of the CICS fields as arguments in CICS commands, because these fields may be altered by the EXEC interface modules.

How a CICS transaction flows

While it runs, your application program requests various CICS facilities to handle message transmissions between it and the terminal, and to handle any necessary file or database accesses.

To begin an online session with CICS, users usually begin by ″signing on,″ the process which identifies them to CICS. Signing on to CICS gives users the authority to invoke certain transactions. When signed on, users invoke the particular transaction they intend to use. A CICS transaction is usually identified by a 1- to 4-character transaction identifier or TRANSID, which is defined in a table that names the initial program to be used for processing the transaction.

Application programs are stored in a library on a direct access storage device (DASD) attached to the processor. They can be loaded when the system is started, or simply loaded as required. If a program is in storage and isn't being used, CICS can release the space for other purposes. When the program is next needed, CICS loads a fresh copy of it from the library.

In the time it takes to process one transaction, the system may receive messages from several terminals. For each message, CICS loads the application program (if it isn't already loaded), and starts a task to execute it. Thus, multiple CICS tasks can be running concurrently.

Multithreading is a technique that allows a single copy of an application program to be processed by several transactions concurrently. For example, one transaction may begin to execute an application program (a traveller requests information). While this happens, another transaction may then execute the same copy of the application program (another traveller requests information). Compare this with **single-threading**, which is the execution of a program to completion: processing of the program by one transaction is completed before another transaction can use it. Multithreading requires that all CICS application programs be quasi-reentrant; that is, they must be serially reusable between entry and exit points. CICS application programs using the CICS commands obey this rule automatically.

CICS maintains a separate thread of control for each task. When, for example, one task is waiting to read a disk file, or to get a response from a terminal, CICS is able to give control to another task. Tasks are managed by the CICS **task control** program.

CICS manages both multitasking and requests from the tasks themselves for services (of the operating system or of CICS itself). This allows CICS processing to continue while a task is waiting for the operating system to complete a request on its behalf. Each transaction that is being managed by CICS is given control of the processor when that transaction has the highest priority of those that are ready to run.

While it runs, your application program requests various CICS facilities to handle message transmissions between it and the terminal, and to handle any necessary file or database accesses. When the application is complete, CICS returns the terminal to a standby state. Figure 8, Figure 9 on page 14, and Figure 10 on page 14 help you understand what goes on.

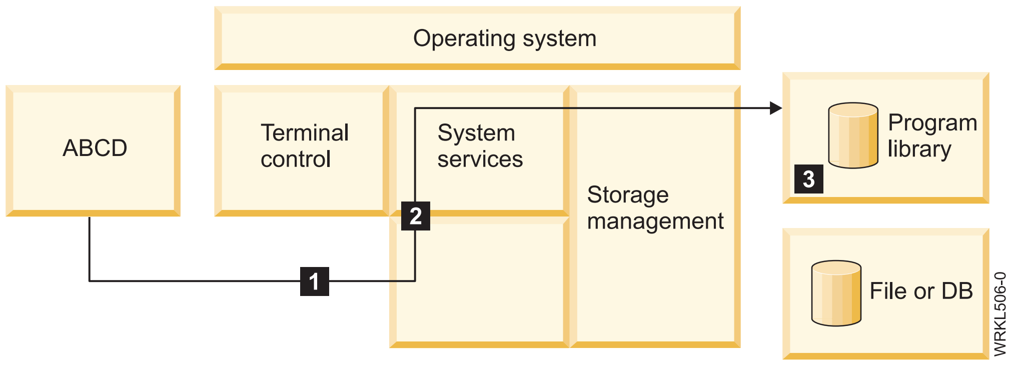


Figure 8. CICS transaction flow (part 1)

The flow of control during a transaction (code ABCD) is shown by the sequence of numbers 1 to 8. (We're only using this transaction to show some of the stages than can be involved.) The meanings of the eight stages are as follows:

1. **Terminal control** accepts characters ABCD, typed at the terminal, and puts them in working storage.
2. **System services** interpret the transaction code ABCD as a call for an application program called ABCD00. If the terminal operator has authority to invoke this program, it is either found already in storage or loaded into storage.
3. Modules are brought from the **program library** into working storage.

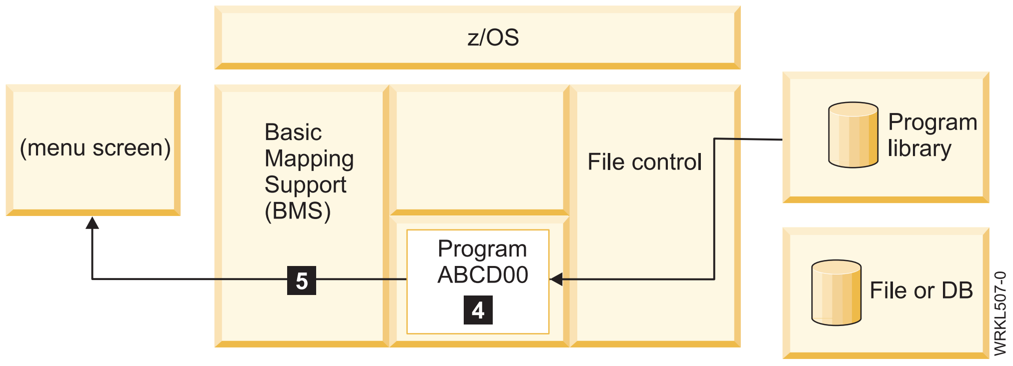


Figure 9. CICS transaction flow (part 2)

4. A **task** is created. Program ABCD00 is given control on its behalf.
5. ABCD00 invokes **Basic mapping support** (BMS) and terminal control to send a menu to the terminal, allowing the user to specify precisely what information is needed.

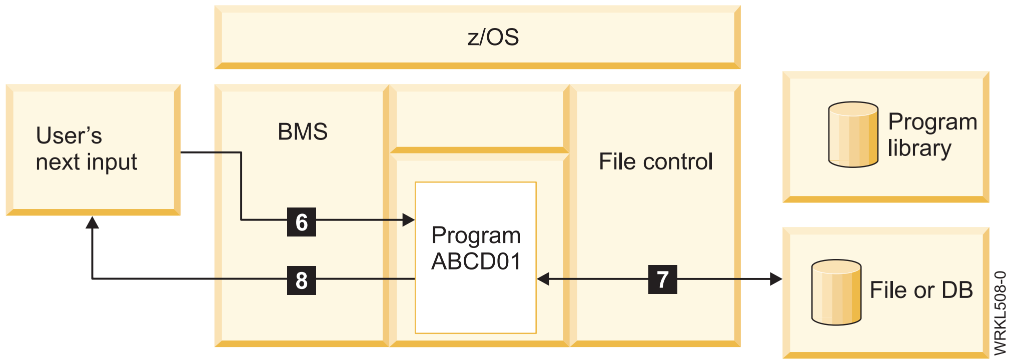


Figure 10. CICS transaction flow (part 3)

6. BMS and terminal control also handle the user's next input, returning it to ABCD01 (the program designated by ABDC00 to handle the next response from the terminal) which then invokes file control.
7. **File control** reads the appropriate file for the invocation the terminal user has requested.
8. Finally, ABCD01 invokes BMS and terminal control to format the retrieved data and present it on the terminal.

CICS services for application programs

CICS applications execute under CICS control, using CICS services and interfaces to access programs and files.

Application programming interface

You use the **application** programming interface or API to access CICS services from the application program. You write a CICS program in much the same way as you write any other program. Most of the processed logic is expressed in standard language elements, but you can use CICS commands to request CICS services.

Terminal control services

These services allow a CICS application program to communicate with terminal devices. Through these services, information may be sent to a terminal screen and the user input may be retrieved from it. It's not easy to deal with **terminal control services** in a direct way. Basic Mapping Support, or BMS, lets you communicate with a terminal with a higher language level. It formats your data, and you do not need to know the details of the data stream.

File and database control services

We may differentiate the following two different CICS data management services:

1. CICS file control offers you access to data sets that are managed by either the Virtual Storage Access Method (VSAM) or the Basic Direct Access Method (BDAM). CICS file control lets you read, update, add, and browse data in VSAM and BDAM data sets and delete data from VSAM data sets.
2. Database control lets you access DL/I and DB2 databases. Although CICS has two programming interfaces to DL/I, we recommend that you use the higher-level EXEC DL/I interface. CICS has one interface to DB2: the EXEC SQL interface, which offers powerful statements for manipulating sets of tables, thus relieving the application program of record-by-record (or segment-by-segment, in the case of DL/I) processing.

Other CICS services

- Task control can be used to control the execution of a task. You may suspend a task or schedule the use of a resource by a task by making it serially reusable. Also, the priority assigned to a task may be changed.
- Program control governs the flow of control between application programs in a CICS system. The name of the application referred to in a program control command must have been defined as a program to CICS. You can use program control commands to link one of your application programs to another, and transfer control from one application program to another, with no return to the requesting program.
- Temporary Storage (TS) and Transient Data (TD) control. The CICS temporary storage control facility provides the application programmer with the ability to store data in temporary storage queues, either in main storage or in auxiliary storage on a direct-access storage device, or, in the case of temporary storage, the coupling facility. The CICS transient data control facility provides a generalized queuing facility to queue (or store) data for subsequent or external processing.
- Interval control services provide functions that are related to time. Using interval control commands, you can start a task at a specified time or after a specified interval, delay the processing of a task, and request notification when a specified time has expired, among other actions.
- Storage control facility controls requests for main storage to provide intermediate work areas and other main storage needed to process a transaction. CICS makes working storage available with each program automatically, without any request from the application program, and provides other facilities for intermediate storage both within and among tasks. In addition to the working storage provided automatically by CICS, however, you can use other CICS commands to get and release main storage.
- Dump and trace control. The dump control provides a transaction dump when an abnormal termination occurs during the execution of an application program.

CICS trace is a debugging aid for application programmers that produces trace entries of the sequence of CICS operations.

CICS program control

A transaction (task) may execute several programs in the course of completing its work.

The program definition contains one entry for every program used by any application in the CICS system. Each entry holds, among other things, the language in which the program is written. The transaction definition has an entry for every transaction identifier in the system, and the important information kept about each transaction is the identifier and the name of the first program to be executed on behalf of the transaction.

You can see how these two sets of definitions, transaction and program, work in concert:

- The user types in a transaction identifier at the terminal (or the previous transaction determined it).
- CICS looks up this identifier in the list of installed transaction definitions.
- This tells CICS which program to invoke first.
- CICS looks up this program in the list of installed transaction definitions, finds out where it is, and loads it (if it isn't already in the main storage).
- CICS builds the control blocks necessary for this particular combination of transaction and terminal, using information from both sets of definitions. For programs in command-level COBOL, this includes making a private copy of working storage for this particular execution of the program.
- CICS passes control to the program, which begins running using the control blocks for this terminal. This program may pass control to any other program in the list of installed program definitions, if necessary, in the course of completing the transaction.

There are two CICS commands for passing control from one program to another. One is the LINK command, which is similar to a **CALL** statement in COBOL. The other is the XCTL (transfer control) command, which has no COBOL counterpart. When one program links another, the first program stays in main storage. When the second (linked-to) program finishes and gives up control, the first program resumes at the point after the LINK. The linked-to program is considered to be operating at one logical level lower than the program that does the linking.

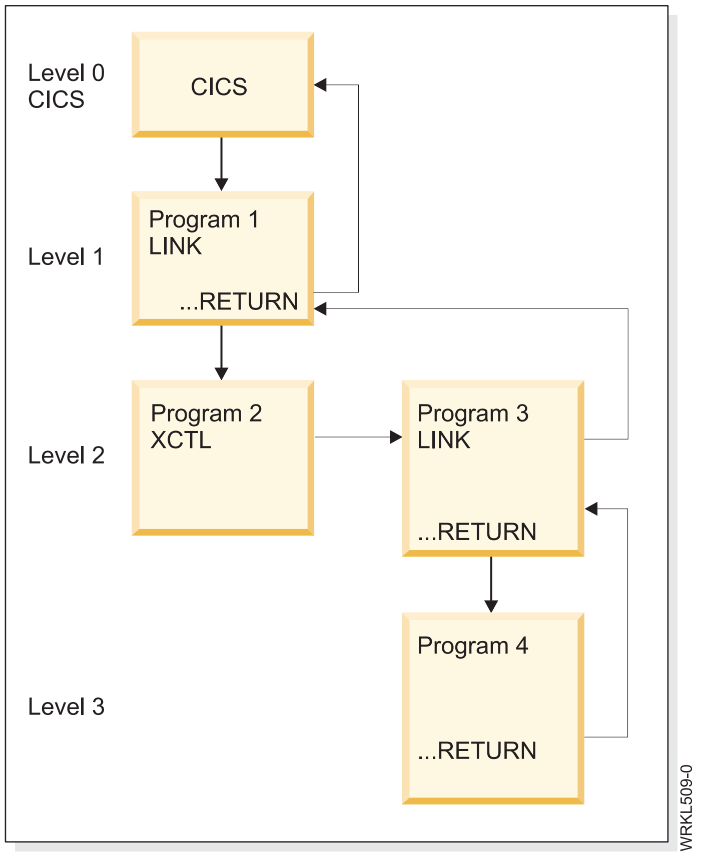


Figure 11. Transferring control between programs (normal returns)

In contrast, when one program transfers control to another, the first program is considered terminated, and the second operates at the same level as the first. When the second program finishes, control is returned not to the first program, but to whatever program last issued a LINK command.

Some people like to think of CICS itself as the highest program level in this process, with the first program in the transaction as the next level down, and so on. Figure 11 illustrates this concept.

The LINK command looks like this:

```
EXEC CICS LINK PROGRAM(pgmname)
     COMMAREA(commarea) LENGTH(length) END-EXEC.
```

where *pgmname* is the name of the program to which you wish to link. *commarea* is the name of the area containing the data to be passed and/or the area to which results are to be returned. The COMMAREA interface is also an option to invoke CICS programs.

A sound principle of CICS application design is to separate the presentation logic from the business logic; communication between the programs is achieved by using the LINK command and data is passed between such programs in the COMMAREA. Such a modular design provides not only a separation of functions, but also much greater flexibility for the Web enablement of existing applications using new presentation methods.

CICS programming roadmap

Here are typical steps for developing a CICS application that uses the EXEC CICS command level programming interface.

1. Design the application, identifying the CICS resources and services you will use. See the section on Application Design in *CICS Application Programming Guide*.
2. Write the program in the language of your choice, including EXEC CICS commands to request CICS services. See *CICS Application Programming Reference* for a list of CICS commands.

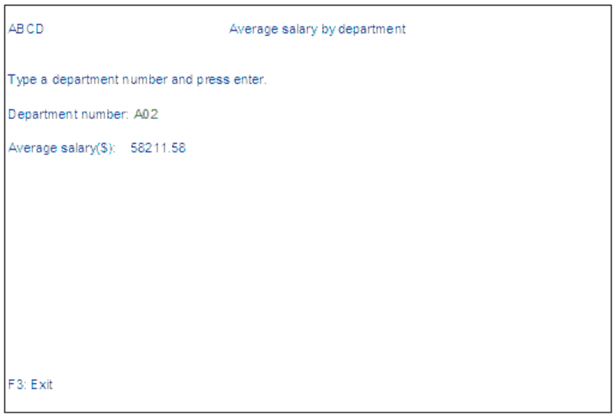
   One of the needed components for online transactions is the screen definition, that is, the layout of what is displayed on the screen (such as a Web page); in CICS we call this a **map**.
3. Depending on the compiler, you might only need to compile the program and install it in CICS, or you might need to define translator options for the program and then translate and compile your program. See *CICS Application Programming Guide* for more details.
4. Define your program and related transactions to CICS with PROGRAM resource definitions and TRANSACTION resource definitions, as described in *CICS Resource Definition Guide*.
5. Define any CICS resources that your program uses, such as files, queues, or terminals.
6. Make the resources known to CICS using the CEDA INSTALL command described in *CICS Resource Definition Guide*.

CICS online example

Here are examples of CICS transactions.

- Adding, updating and/or deleting employee information
- Adding, updating and/or deleting available cars by rental company
- Getting the number of available cars by rental company
- Updating prices of rental cars
- Adding, updating and/or deleting regular flights by airline
- Getting the number of sold tickets by airline or by destination.

Figure 12 on page 19 shows how a user can calculate the average salary by department. The department is entered by the user and the transaction calculates the average salary.


ABCD Average salary by department

Type a department number and press enter.

Department number: A02

Average salary($): 58211.58

F3: Exit

Figure 12. CICS application user screen

Notice that you can add PF key definitions to the user screens in your CICS applications.

What is IMS?

Created in 1969 as Information Management System/360™, IMS is both a transaction manager and a database manager for z/OS. IMS consists of three components: the Transaction Manager (TM), the Database Manager (DB), and a set of system services that provide common services to the other two components.

As IMS developed over the years, new interfaces were added to meet new business requirements. It is now possible to access IMS resources using a number of interfaces to the IMS components.

In this section, we look at the transaction manager functions of IMS.

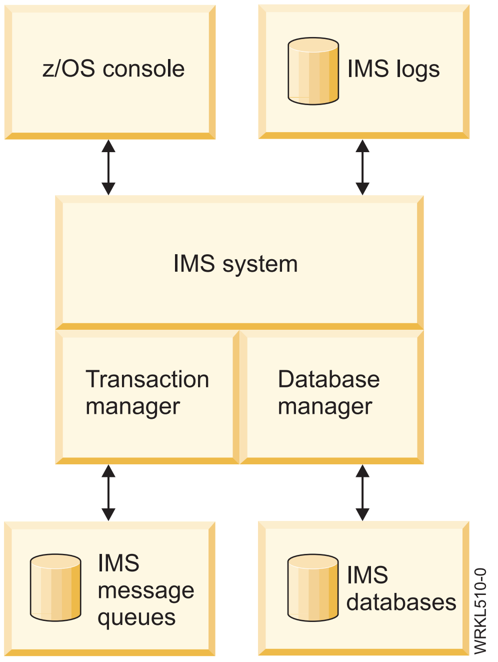


Figure 13. Overview of the IMS product

You write an IMS program in much the same way you write any other program. You can use COBOL, OO COBOL, C, C++, Java, PL/I, or Assembler language to write IMS application programs. More information about programming in Java can be found in the IBM publication *IMS Java Guide and Reference*.

IMS Transaction Manager

The IMS Transaction Manager provides users of a network with access to applications running under IMS. The users can be people at terminals or workstations, or they can be other application programs either on the same z/OS system, on other z/OS systems, or on non-z/OS platforms.

A transaction is a setup of input data that triggers the execution of a specific business application program. The message is destined for an application program, and the return of any results is considered one transaction.

IMS Database Manager

The IMS Database Manager component of IMS provides a central point of control and access for the data that is processed by IMS applications. It supports databases using the IMS hierarchical database model and provides access to these databases from applications running under the IMS Transaction Manager, the CICS transaction monitor (now known as Transaction Server for z/OS), and z/OS batch jobs.

The Database Manager component provides facilities for securing (backup/recovery) and maintaining the databases. It allows multiple tasks (batch and/or online) to access and update the data, while retaining the integrity of that data. It also provides facilities for tuning the databases by reorganizing and restructuring them. IMS databases are organized internally using a number of IMS database organization access methods. The database data is stored on disk storage using the normal operating system access methods.

IMS System Services

There are a number of functions that are common to both the Database Manager and Transaction Manager:
- Restart and recovery of the IMS subsystems following failures
- Security: controlling access to IMS resources
- Managing the application programs: dispatching work, loading application programs, providing locking services
- Providing diagnostic and performance information
- Providing facilities for the operation of the IMS subsystems
- Providing an interface to other z/OS subsystems with which IMS applications interface

IMS in a z/OS system

IMS runs on zSeries® and earlier forms of the S/390® architecture or compatible mainframes, and on z/OS and earlier forms of the operating system. An IMS subsystem runs in several address spaces in a z/OS system. There is one controlling address space and several dependent address spaces providing IMS services and running IMS application programs.

For historical reasons, some documents describing IMS use the term **region** to describe a z/OS address space, for example, IMS Control Region. We use the term region whenever this is in common usage. You can take the term region as being the same as a z/OS address space.

To make the best use of the unique strengths of z/OS, IMS does the following:

- Runs in multiple address spaces–IMS subsystems (except for IMS/DB batch applications and utilities) normally consist of a control region address space, dependent address spaces providing system services, and dependent address spaces for application programs.
- Runs multiple tasks in each address space–IMS, particularly in the control regions, creates multiple z/OS subtasks for the various functions to be performed. This allows other IMS subtasks to be dispatched by z/OS while one IMS subtask is waiting for system services.
- Uses z/OS cross-memory services to communicate between the various address spaces making up an IMS subsystem. It also uses the z/OS Common System Area (CSA) to store IMS control blocks that are frequently accessed by the IMS address spaces, thus minimizing the overhead of using multiple address spaces.
- Uses the z/OS subsystem feature–IMS dynamically registers itself as a z/OS subsystem. It uses this facility to detect when dependent address spaces fail, and prevent cancellation of dependent address spaces (and to interact with other subsystems like DB2 and WebSphere MQ).
- Can make use of a z/OS sysplex–Multiple IMS subsystems can run on the z/OS systems making up the sysplex and access the same IMS databases.

IMS Transaction Manager messages

The network inputs and outputs to IMS Transaction Manager take the form of messages that are input and output to and from IMS and the physical terminals or application programs on the network. These messages are processed asynchronously (that is, IMS will not always send a reply immediately, or indeed ever, when it receives a message, and unsolicited messages may also be sent from IMS).

The messages can be of four types:

- Transactions–Data in these messages is passed to IMS application programs for processing.
- Messages to go to other logical destinations, such as network terminals.
- Commands for IMS to process.
- Messages for the IMS APPC feature to process. Since IMS uses an asynchronous protocol for messages, but APPC uses synchronous protocols (that is, it always expects a reply when a message is sent), the IMS TM interface for APPC has to perform special processing to accommodate this.

If IMS is not able to process an input message immediately, or cannot send an output message immediately, the message is stored on a message queue external to the IMS system. IMS will not normally delete the message from the message queue until it has received confirmation that an application has processed the message, or it has reached its destination.

Chapter 2. Database management systems on z/OS

Two of the most widely used database management system (DBMS) products for z/OS are DB2 and IMS DB.

This section gives an overview of basic database (DB) concepts, what they are used for, and what the advantages are. There are many databases, but here we limit the scope to the two types that are used most on mainframes: hierarchical and relational databases.

This section focuses on DB2 and IMS DB.

What is a database?

A database provides for the storing and control of business data. It is independent from (but not separate from the processing requirements of) one or more applications. If properly designed and implemented, the database should provide a single consistent view of the business data, so that it can be centrally controlled and managed.

One way of describing a logical view of this collection of data is to use an entity relationship model. The database records details (attributes) of particular items (entities) and the relationships between the different types of entities. For example, for the stock control area of an application, you would have Parts, Purchase Orders, Customers, and Customer Orders (entities). Each entity would have attributes–the Part would have a Part No, Name, Unit Price, Unit Quantity, and so on.

These entities would also have relationships between them, for example a Customer would be related to orders placed, which would be related to the part that had been ordered, and so on. Figure 14 on page 24 illustrates an entity relationship model.

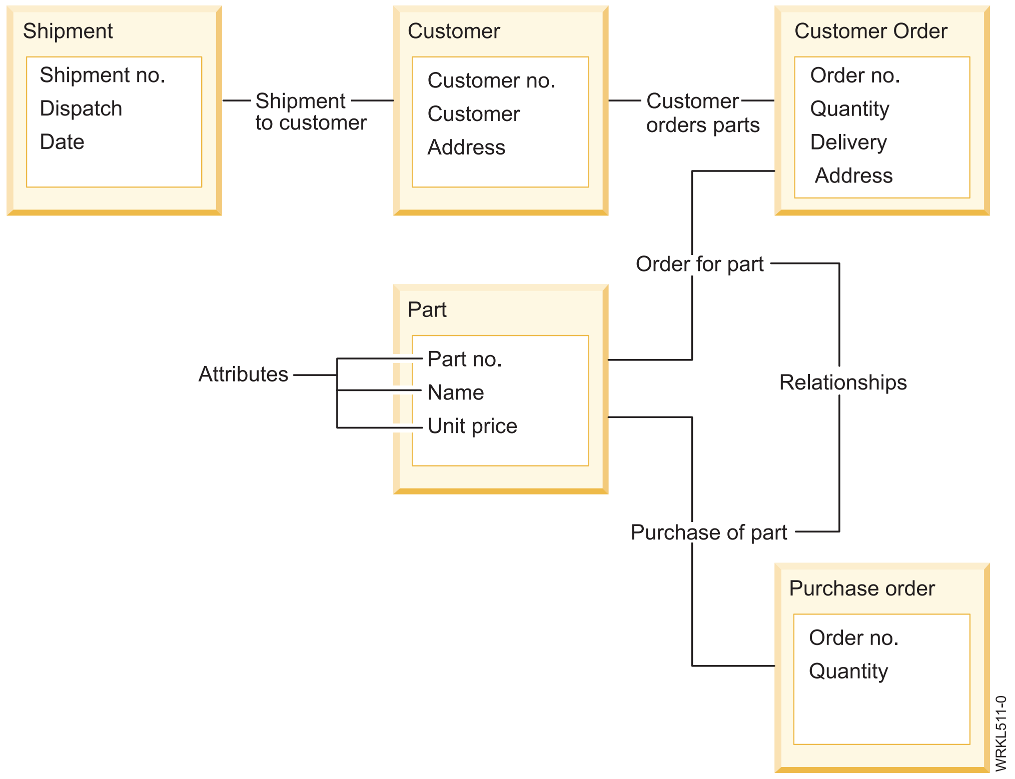


Figure 14. Entities, attributes, and relationships

A database management system (DBMS), such as the IMS Database Manager (IMS/DB) component or the DB2 product, provides a method for storing and using the business data in the database.

Why use a database?

A database has advantages over a flat file system.

When computer systems were first developed, the data was stored on individual files that were unique to an application or even a small part of an individual application. However, a properly designed and implemented DBMS provides many advantages over a flat file PDS system:

- It reduces the application programming effort.
- It manages more efficiently the creation and modification of, and access to, data than a non-DBMS system. As you know, if new data elements need to be added to a file, then all applications that use that file must be rewritten, even those that do not use the new data element. This need not happen when using a DBMS. Although many programmers have resorted to ″tricks″ to minimize this application programming rewrite task, it still requires effort.
- It provides a greater level of data security and confidentiality than a flat file system. Specifically, when accessing a logical record in a flat file, the application can see **all** data elements–including any confidential or privileged data. To minimize this, many customers have resorted to putting sensitive data into a separately managed file, and linking the two as necessary. This may cause data consistency issues.

  With a DBMS, the sensitive data can be isolated in a separate segment (in IMS/DB) or View (in DB2) that prevents unauthorized applications from seeing it. But these data elements are an integral part of the logical record!

However, the same details might be stored in several different places; for example, the details of a customer might be in both the ordering and invoicing application. This causes a number of problems:

- Because the details are stored and processed independently, details that are supposed to be the same (for example, a customer's name and address), might be inconsistent in the various applications.
- When common data has to be changed, it must be changed in several places, causing a high workload. If any copies of the data are missed, it results in the problems detailed in the previous point.
- There is no central point of control for the data to ensure that it is secure, both from loss and from unauthorized access.
- The duplication of the data wastes space on storage media.

The use of a database management system such as IMS/DB or DB2 to implement the database also provides additional advantages. The DBMS:

- Allows multiple tasks to access and update the data simultaneously, while preserving database integrity. This is particularly important where large numbers of users are accessing the data through an online application.
- Provides facilities for the application to update multiple database records and ensures that the application data in the various records remains consistent even if an application failure occurs.
- Is able to put confidential or sensitive data in a separate segment (in IMS) or table (in DB2). In contrast, in a PDS or VSAM flat file, the application program gets access to every data element in the logical record. Some of these elements might contain data that should be restricted.
- Provides utilities that control and implement backup and recovery of the data, preventing loss of vital business data.
- Provides utilities to monitor and tune access to the data.
- Is able to change the structure of the logical record (by adding or moving data fields). Such changes usually require that every application that accesses the VSAM or PDS file must be reassembled or recompiled, even if it does not need the added or changed fields. A properly designed data base insulates the application programmer from such changes.

Keep in mind, however, that the use of a database and database management system will not, in itself, produce the advantages detailed here. It also requires the proper design and administration of the databases, and development of the applications.

Who is the database administrator?

Database administrators (DBAs) are primarily responsible for specific databases in the subsystem. In some companies, DBAs are given the special group authorization, SYSADM, which gives them the ability to do almost everything in the DB2 subsystem, and gives them jurisdiction over all the databases in the subsystem. In other shops, a DBA's authority is limited to individual databases.

The DBA creates the hierarchy of data objects, beginning with the database, then table spaces, tables, and any indexes or views that are required. This person also sets up the referential integrity definitions and any necessary constraints.

The DBA essentially implements the physical database design. Part of this involves having to do space calculations and determining how large to make the physical data sets for the table spaces and index spaces, and assigning storage groups (also called **storgroups**).

There are many tools that can assist the DBA in these tasks. DB2, for example, provides the Administration Tool and the DB2 Estimator. If objects increase in size, the DBA is able to alter certain objects to make changes.

The DBA can be responsible for granting authorizations to the database objects, although sometimes there is a special security administration group that does this.

The centralization of data and control of access to this data is inherent to a database management system. One of the advantages of this centralization is the availability of consistent data to more than one application. As a consequence, this dictates tighter control of that data and its usage.

Responsibility for an accurate implementation of control lies with the DBA. Indeed, to gain the full benefits of using a centralized database, you must have a central point of control for it. Because the actual implementation of the DBA function is dependent on a company's organization, we limit ourselves to a discussion of the roles and responsibilities of a DBA. The group fulfilling the DBA role will need experience in both application and systems programming.

In a typical installation, the DBA is responsible for:
- Providing the standards for, and the administration of, databases and their use
- Guiding, reviewing, and approving the design of new databases
- Determining the rules of access to the data and monitoring its security
- Ensuring database integrity and availability, and monitoring the necessary activities for reorganization backup and recovery
- Approving the operation of new programs with existing production databases, based on results of testing with test data.

In general, the DBA is responsible for the maintenance of current information about the data in the database. Initially, this responsibility might be carried out using a manual approach. But it can be expected to grow to a scope and complexity sufficient to justify, or necessitate, the use of a data dictionary program.

The DBA is not responsible for the actual content of databases. This is the responsibility of the user. Rather, the DBA enforces procedures for accurate, complete, and timely update of the databases.

How is a database designed?

The process of database design, in its simplest form, is the structuring of the data elements for the various applications, in such an order that (1) each data element is readily available by the various applications, now and in the foreseeable future, (2) the data elements are efficiently stored, and (3) controlled access is enforced for those data elements with specific security requirements.

A number of different models for databases have been developed over the years (such as hierarchical, relational, or object) so that there is no consistent vocabulary for describing the concepts involved.

Entities

A database contains information about entities. An **entity** is something that:

- Can be uniquely defined.
- We might collect substantial information about, now or in the future.

In practice, this definition is limited to the context of the applications and business under consideration. Examples of entities are parts, projects, orders, customers, trucks, etc. It should be clear that defining entities is a major step in the database design process. The information we store in databases about entities is described by data attributes.

Data attributes

A **data attribute** is a unit of information that specifies a fact about an entity. For example, suppose the entity is a part. Name=Washer, Color=Green, and Weight=143 are three facts about that part. Thus these are three data attributes.

A data attribute has a name and a value. A data attribute name tells the kind of fact being recorded; the value is the fact itself. In this example, Name, Color, and Weight are data attribute names; while Washer, Green and 143 are values. A value must be associated with a name to have a meaning.

An **occurrence** is the value of a data attribute for a particular entity. An attribute is always dependent on an entity. It has no meaning by itself. Depending on its usage, an entity can be described by one single data attribute, or more. Ideally, an entity should be uniquely defined by one single data attribute, for example, the order number of an order. Such a data attribute is called the **key** of the entity. The key serves as the identification of a particular entity occurrence, and is a special attribute of the entity. Keys are not always unique. Entities with equal key values are called **synonyms**.

For instance, the full name of a person is generally not a unique identification. In such cases we have to rely on other attributes such as full address, birthday, or an arbitrary sequence number. A more common method is to define a new attribute that serves as the unique key, for example, employee number.

Entity relationships

The entities identified will also have connections between them, called **relationships**. For example, an order might be for a number of parts. Again these relationships only have meaning within the context of the application and business. These relationships can be one-to-one (that is, one occurrence of an entity relates to a single occurrence of another entity), one-to-many (one occurrence of an entity relates to many occurrences of another entity), or many-to-many (many occurrences of one entity have a relationship with many occurrences of another entity).

Relationships can also be **recursive**, that is, an entity can have a relationship with other occurrences of the same entity. For example a part, say a fastener, may consist of several other parts: bolt, nut, and washer.

Application functions

Data itself is not the ultimate goal of a database management system. It is the application processing performed on the data that is important. The best way to represent that processing is to take the smallest application unit representing a user interacting with the database–for example, one single order, one part's inventory status. In the following sections we call this an **application function**.

Functions are processed by application programs. In a batch system, large numbers of functions are accumulated into a single program (that is, all orders of a day), then processed against the database with a single scheduling of the desired application program. In the online system, just one or two functions may be grouped together into a single program to provide one iteration with a user.

Although functions are always distinguishable, even in batch, some people prefer to talk about programs rather than functions. But a clear understanding of functions is mandatory for good design, especially in a DB environment. Once you have identified the functional requirements of the application, you can decide how to best implement them as programs using CICS or IMS. The function is, in some way, the individual use of the application by a particular user. As such, it is the focal point of the DB system.

Access paths

Each function bears in its input some kind of identification with respect to the entities used (for example, the part number when accessing a Parts database). These are referred to as the access paths of that function. In general, functions require random access, although for performance reasons sequential access is sometimes used. This is particularly true if the functions are batched, and if they are numerous relative to the database size, or if information is needed from most database records. For efficient random access, each access path should utilize the entities key.

What is a database management system?

A database management system (or DBMS) is essentially nothing more than a computerized data-keeping system. Users of the system are given facilities to perform several kinds of operations on such a system for either manipulation of the data in the database or the management of the database structure itself. Database Management Systems (DBMSs) are categorized according to their data structures or types.

There are several types of databases that can be used on a mainframe to exploit z/OS: inverted list, hierarchic, network, or relational.

Mainframe sites tend to use a hierarchical model when the data **structure** (not data values) of the data needed for an application is relatively static. For example, a Bill of Material (BOM) database structure always has a high level assembly part number, and several levels of components with subcomponents. The structure usually has a component forecast, cost, and pricing data, and so on. The structure of the data for a BOM application rarely changes, and new data elements (not values) are rarely identified. An application normally starts at the top with the assembly part number, and goes down to the detail components.

Hierarchical and relational database systems have common benefits. RDBMS has the additional, significant advantage over the hierarchical DB of being non-navigational. By **navigational**, we mean that in a hierarchical database, the application programmer must know the structure of the database. The program must contain specific logic to navigate from the root segment to the desired child segments containing the desired attributes or elements. The program must still access the intervening segments, even though they are not needed.

The remainder of this section discusses the relational database structure.

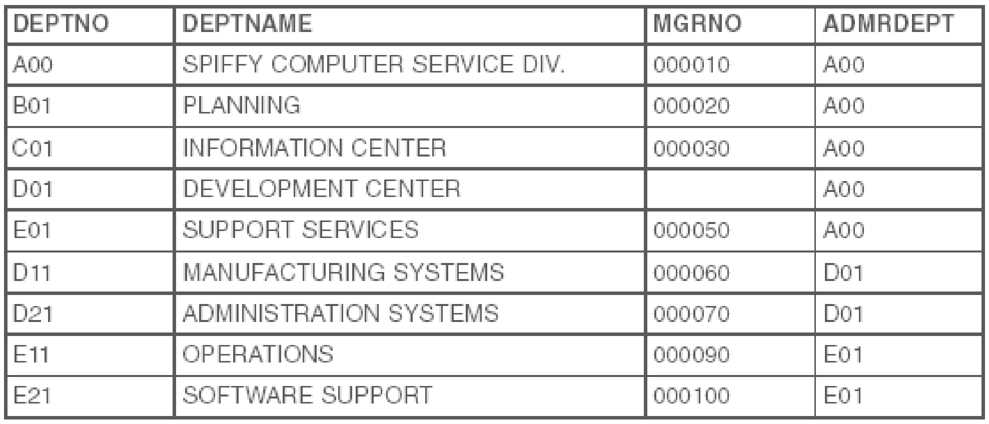
What structures exist in a relational database?

Relational databases include the following structures:

Database
A database is a logical grouping of data. It contains a set of related table spaces and index spaces. Typically, a database contains all the data that is associated with one application or with a group of related applications. You could have a payroll database or an inventory database, for example.

Table A table is a logical structure made up of rows and columns. Rows have no fixed order, so if you retrieve data you might need to sort the data. The order of the columns is the order specified when the table was created by the database administrator. At the intersection of every column and row is a specific data item called a value, or, more precisely, an atomic value. A table is named with a high-level qualifier of the owner's user ID followed by the table name, for example TEST.DEPT or PROD.DEPT. There are three types of tables:

- A base table that is created and holds persistent data
- A temporary table that stores intermediate query results
- A results table that is returned when you query tables.



DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
B01	PLANNING	000020	A00
C01	INFORMATION CENTER	000030	A00
D01	DEVELOPMENT CENTER		A00
E01	SUPPORT SERVICES	000050	A00
D11	MANUFACTURING SYSTEMS	000060	D01
D21	ADMINISTRATION SYSTEMS	000070	D01
E11	OPERATIONS	000090	E01
E21	SOFTWARE SUPPORT	000100	E01

Figure 15. Example of a DB2 table (department table)

In this table we use:

- Columns–The ordered set of columns are DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT. All the data in a given column must be of the same data type.
- Rows–Each row contains data for a single department.
- Values–At the intersection of a column and row is a value. For example, PLANNING is the value of the DEPTNAME column in the row for department B01.

Indexes

An index is an ordered set of pointers to rows of a table. Unlike the rows of a table that are not in a specific order, an index must always be maintained in order by DB2. An index is used for two purposes:

- For performance, to retrieve data values more quickly
- For uniqueness.

By creating an index on an employee's name, you can retrieve data more quickly for that employee than by scanning the entire table. Also, by creating a unique index on an employee number, DB2 will enforce the uniqueness of each value. A unique index is the only way DB2 can enforce uniqueness.

Creating an index automatically creates the **index space**, the data set that contains the index.

Keys A key is one or more columns that are identified as such in the creation of a table or index, or in the definition of referential integrity.

Primary key

A table can only have one primary key because it defines the entity. There are two requirements for a primary key:

1. It must have a value, that is, it cannot be null.
2. It must be unique, that is, it must have a unique index defined on it.

Unique key

We already know that a primary key must be unique, but it is possible to have more than one unique key in a table. In our EMP table example, the employee number is defined as the primary key and is therefore unique. If we also had a social security value in our table, hopefully that value would be unique. To guarantee this, you could create a unique index on the social security column.

Foreign key

A foreign key is a key that is specified in a referential integrity constraint to make its existence dependent on a primary or unique key (parent key) in another table.

The example given is that of an employee's work department number relating to the primary key defined on the department number in the DEPT table. This constraint is part of the definition of the table.

What is DB2?

DB2 (Database 2™) is an IBM relational database management system.

Most table examples in this section can be found in the sample database that comes with the DB2 product on all platforms. We are using Version 8 in the screen captures. Therefore, the owner of our tables is DSN8810.

The elements that DB2 manages can be divided into two categories: Data structures that are used to organize user data, and system structures that are controlled by DB2. Data structures can be further broken down into **basic structures** and **schema structures**. Schema structures are fairly new objects that were introduced on the mainframe for compatibility within the DB2 family. A **schema** is a logical grouping of these new objects.

Data structures in DB2

DB2 data structures include views, table spaces, index spaces, and storage groups.

Views

A **view** is an alternative way of looking at the data in one or more tables. It is like an overlay that you would put over a transparency to only allow people to see certain aspects of the base transparency. For example, you can create a view on the department table to only let users have access to one particular department in order to update salary information. You don't want them to see the salaries in other departments. You create a view of the table that only lets the users see one department, and they use the view like a table. Thus, a view is used for security reasons. Most companies will not allow users to access their tables directly, but instead use a view to accomplish this. The users get access through the view. A view can also be used to simplify a complex query for less experienced users.

Table space

A table is just a logical construct. It is kept in an actual physical data set called a **table space**. Table spaces are storage structures and can contain one or more tables. A table space is named using the database name followed by the table space name, such as PAYROLL.ACCNT_RECV. There are three types of table spaces: Simple, Segmented, and Partitioned. For more detailed information, see *DB2 UDB for z/OS: SQL Reference*.

DB2 uses VSAM data sets. That is, each segment is a VSAM data set.

Index space

An **index space** is another storage structure that contains a single index. In fact, when you create an index, DB2 automatically defines an index space for you.

Storage groups

A **storage group** consists of a set of volumes on disks (DASD) that hold the data sets in which tables and indexes are actually stored.

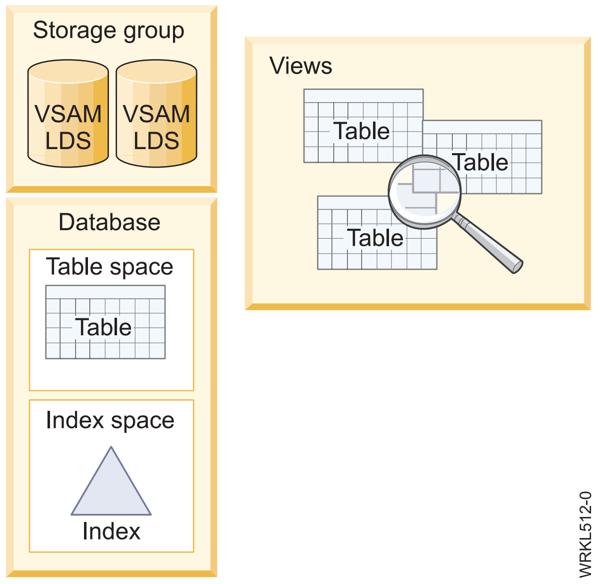


Figure 16. There is a hierarchy to the objects in a DB2 subsystem

DB2 schema structures

Schema structures include user-defined data types, user-defined functions, triggers, and large objects.

User-defined data type (UDT)

A UDT is a way for users to define their own data types above and beyond the usual character and numeric data types. However, UDTs are based upon the already existing DB2 data types. If you dealt in international currencies, you would most likely want to differentiate the various types of monies. With a UDT definition, you could define the EURO, based on the decimal data type, as a distinct data type in addition to YEN or US_DOLLAR. As a result, you could not add a YEN to a EURO since they are distinct data types.

User-defined function (UDF)

A UDF can be simply defined on an already existing DB2 function, such as rounding or averaging, or can be more complex and written as an application program that could be accessed by an SQL statement. In our international currency example, we could use a UDF to convert one currency value to another in order to do arithmetic functions.

Trigger

A **trigger** defines a set of actions that are executed when an insert, update, or delete operation occurs on a specific table. For example, let's say that every time you insert an employee into your EMP table, you also want to add one to an employee count that you keep in a company statistics table. You can define a trigger that will get "fired" when you do an insert into EMP. This firing will automatically add one to the appropriate column in the COMPANY_STATS table.

Large Object (LOB)

An LOB is a data type used by DB2 to manage unstructured data. There are three types of LOBs:

- Binary Large Objects (BLOBs) - These are used for photographs and pictures, audio and sound clips, and video clips.
- Character Large Objects (CLOBs) - These are used for large text documents.
- Double Byte Character Large Objects (DBCLOBs) - These are used for storing large text documents written in languages that require double-byte characters, such as Kanji.

LOBs are stored in special auxiliary tables that use a special LOB table space. In your EMP base table, text material such as a resume can be included for employees. Since this is a large amount of data, it is contained in its own table. A column in the EMP table, defined as a CLOB, would have a pointer to this special LOB auxiliary table that is stored in an LOB table space. Each column defined as an LOB would have its own associative auxiliary table and LOB table space.

Stored procedure

A **stored procedure** is a user-written application program that typically is stored and run on the server (but it can be run for local purposes as well).

Stored procedures were specifically designed for the client/server environment where the client would only have to make one call to the server, which would then run the stored procedure to access DB2 data and return the results. This eliminated having to make several network calls to run several individual queries against the database, which can be expensive.

You can think of a stored procedure as being somewhat like a subroutine that can be called to perform a set of related functions. It is an application program, but is defined to DB2 and managed by the DB2 subsystem.

System structures

System structures include the catalog and directory, buffer pools, and archive and active logs.

Catalog and directory

DB2 itself maintains a set of tables that contain metadata or data about all the DB2 objects in the subsystem. The **catalog** keeps information about all the objects, such as the tables, views, indexes, table spaces, and so on, while the **directory** keeps information about the application programs. The catalog can be queried to see the object information; the directory cannot.

When you create a user table, DB2 automatically records the table name, creator, its table space, and database in the catalog and puts this information in the catalog table called SYSIBM.SYSTABLES. All the columns defined in the table are automatically recorded in the SYSIBM.SYSCOLUMNS table.

In addition, to record that the owner of the table has authorization on the table, a row is automatically inserted into SYSIBM.SYSTABAUTH. Any indexes created on the table would be recorded in the SYSIBM.SYSINDEXES table.

Buffer pools

Buffer pools are areas of virtual storage in which DB2 temporarily stores pages of table spaces or indexes. They act as a cache area between DB2 and the physical disk storage device where the data resides. A data page is retrieved from disk and placed in a buffer pool page. If the needed data is already in a buffer, expensive I/O access to the disk can be avoided.

Active and archive logs

DB2 records all data changes and other significant events in a **log**. This information is used to recover data in the event of a failure, or DB2 can roll the changes back to a previous point in time. DB2 writes each log record to a data set called the **active log**.

When the active log is full, DB2 copies the contents to a disk or tape data set called the **archive log**. A bootstrap data set keeps track of these active and archive logs. DB2 uses this information in recovery scenarios, for system restarts, or for any activity that requires reading the log. A bootstrap data set allows for point-in-time recovery.

DB2 address spaces

DB2 is a multi-address space subsystem requiring a minimum of three address spaces: system services, database services, and lock manager services (IRLM).

In addition, Distributed Data Facility (DDF) is used to communicate with other DB2 Subsystems. Figure 17 shows these address spaces.

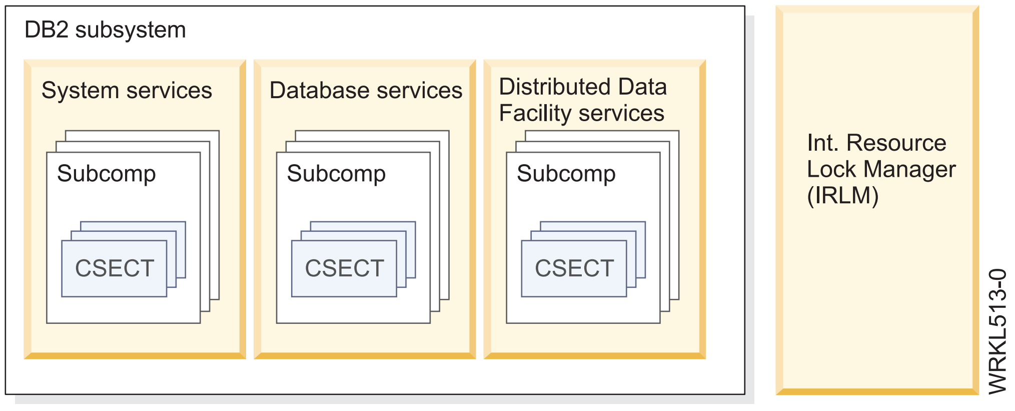


Figure 17. DB2 minimum address spaces

Using DB2 utilities

On z/OS, the DBA maintains database objects through a set of utilities and programs, which are submitted using JCL jobs. Usually a company will have a data set library for these jobs that DBAs copy and use. However, there are tools that will generate the JCL, such as the Administration Tool and the Utility option on the DB2I panel.

The utilities help the DBAs do their jobs. You could divide the utilities into categories:

- Data Organization utilities

After tables are created, the DBA uses the LOAD utility to populate them, with the ability to compress large amounts of data. There is the UNLOAD utility or the DSNTIAUL assembler program that can let the DBA move or copy data from one subsystem to another.

It is possible to keep the data in a certain order with the REORG utility. Subsequent insertions and loads can disturb this order, and the DBA must schedule subsequent REORGs based on reports from the RUNSTATS utility, which provides statistics and performance information. You can even run RUNSTATS against the system catalogs.

- Backup and Recovery utilities

  It is vital that a DBA take image copies of the data and the indexes with the COPY utility in order to recover data. A DBA can make a full copy or an incremental copy (only for data). Since recovery can only be done to a full copy, the MERGECOPY utility is used to merge incremental copies with a full one. The RECOVER utility can recover back to an image copy for a point-in-time recovery. More often, it is used to recover to an image copy, and then information from the logs, which record all data changes, is applied in order to recover forward to a current time. Without an image copy, an index can be recreated with REBUILD INDEX.
- Data consistency utilities

  One of the important data consistency utilities is the CHECK utility, which can be used to check and help correct referential integrity and constraint inconsistencies, especially after an additional population or after a recovery.

A typical use of utilities is to run RUNSTATS, then EXPLAIN, and then RUNSTATS again.

Using DB2 commands

Both the system administrator and the DBA use DB2 commands to monitor the subsystem.

The DB2I panel and the Administration Tool provide you with a means to easily enter these commands. The -DISPLAY DATABASE command displays the status of all table spaces and index spaces within a database. For example, without an image copy, your table can be put in a **copy pending** status, requiring that you run the COPY utility. There are several other display commands, such as DISPLAY UTILITY for the status of a utility job, or you can display buffer pool, thread, and log information.

There are also DSN commands that you can issue from a TSO session or batch job. However, these can be more simply entered using the options from the DB2I panel: BIND, DCLGEN, RUN, and so on. (In some shops, DBAs are responsible for binds, although these are usually done by programmers as part of the compile job.)

What is SQL?

Structured Query Language (SQL) is a high-level language that is used to specify what information a user needs without having to know how to retrieve it. The database is responsible for developing the access path needed to retrieve the data. SQL works at a set level, meaning that it is designed to retrieve one or more rows. Essentially, it is used on one or more tables and returns the result as a results table.

SQL has three categories based on the functionality involved:

- DML–Data manipulation language used to read and modify data

- DDL–Data definition language used to define, change, or drop DB2 objects
- DCL–Data control language used to grant and revoke authorizations

Several tools can be used to enter and execute SQL statements. Here we focus on SPUFI, which stands for SQL Processing Using File Input. SPUFI is part of the DB2 Interactive (DB2I) menu panel, which is a selection from your ISPF panel when DB2 is installed. (This, of course, depends on how your system people set up your system's menu panels.)

SPUFI is most commonly used by database administrators. It allows you to write and save one or more SQL statements at a time. DBAs use it to grant or revoke authorizations; sometimes even to create objects, when that needs to be done urgently. SPUFI is also often used by developers to test their queries. This way they are sure that the query returns exactly what they want.

Another tool that you might encounter on the mainframe is the Query Management Facility (QMF™), which allows you to enter and save just one SQL statement at a time. QMF's main strength is its reporting facility. (QMF includes a governor function to cap the amount of CPU that might be consumed by a poorly constructed or runaway query.) It enables you to design flexible and reusable report formats, including graphs. In addition, it provides a Prompted Query capability that helps users unfamiliar with SQL to build simple SQL statements. Another tool is the Administration Tool, which has SPUFI capabilities as well as a query building facility.

Figure 18 shows how SQL is entered using SPUFI. It is the very first selection on the DB2I panel. Note that the name of this DB2 subsystem is DB8H.

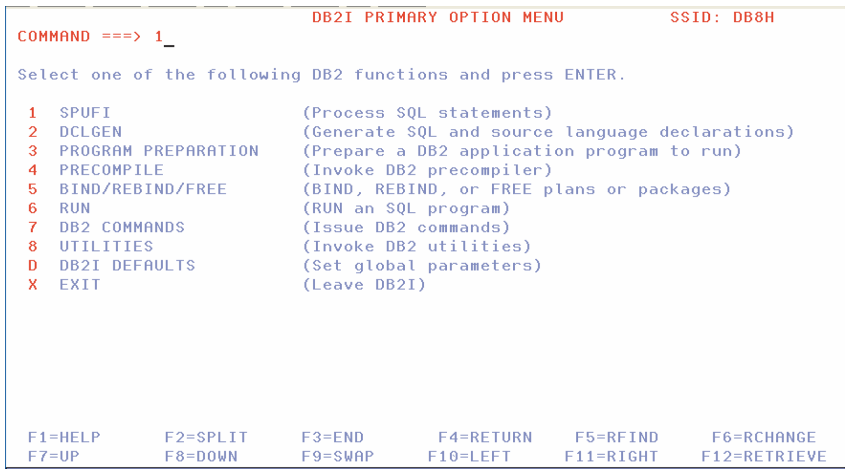
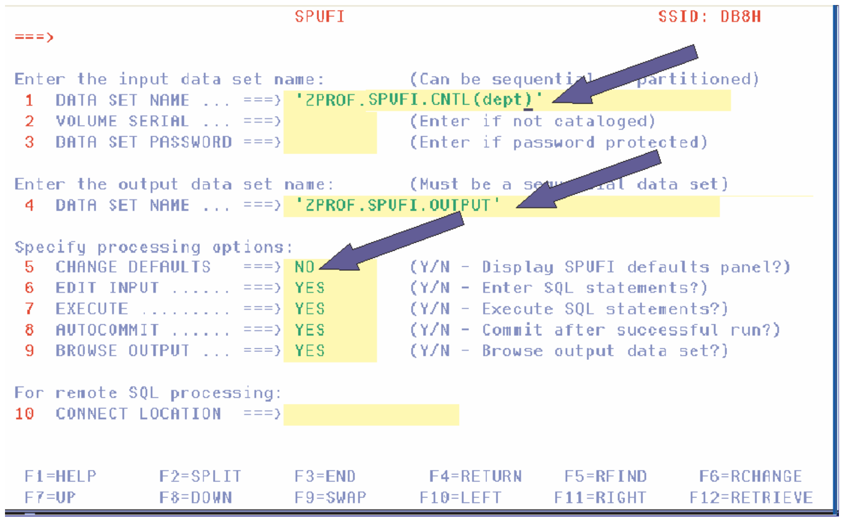

```
                             DB2I PRIMARY OPTION MENU          SSID: DB8H
COMMAND ===> 1_

Select one of the following DB2 functions and press ENTER.

 1  SPUFI                  (Process SQL statements)
 2  DCLGEN                 (Generate SQL and source language declarations)
 3  PROGRAM PREPARATION    (Prepare a DB2 application program to run)
 4  PRECOMPILE             (Invoke DB2 precompiler)
 5  BIND/REBIND/FREE       (BIND, REBIND, or FREE plans or packages)
 6  RUN                    (RUN an SQL program)
 7  DB2 COMMANDS           (Issue DB2 commands)
 8  UTILITIES              (Invoke DB2 utilities)
 D  DB2I DEFAULTS          (Set global parameters)
 X  EXIT                   (Leave DB2I)






 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 18. Entering SQL using SPUFI

SPUFI uses file input and output, so it is necessary to have two data sets pre-allocated:

- The first, which can be named ZPROF.SPUFI.CNTL, is typically a partitioned data set in order to keep or save your queries as members. A sequential data set would write over your SQL.
- The output file, which can be named ZPROF.SPUFI.OUTPUT, must be sequential, which means your output is written over for the next query. If you want to save it, you must rename the file, using the ISPF menu edit facilities.

In Figure 19 you can see how that fits in.

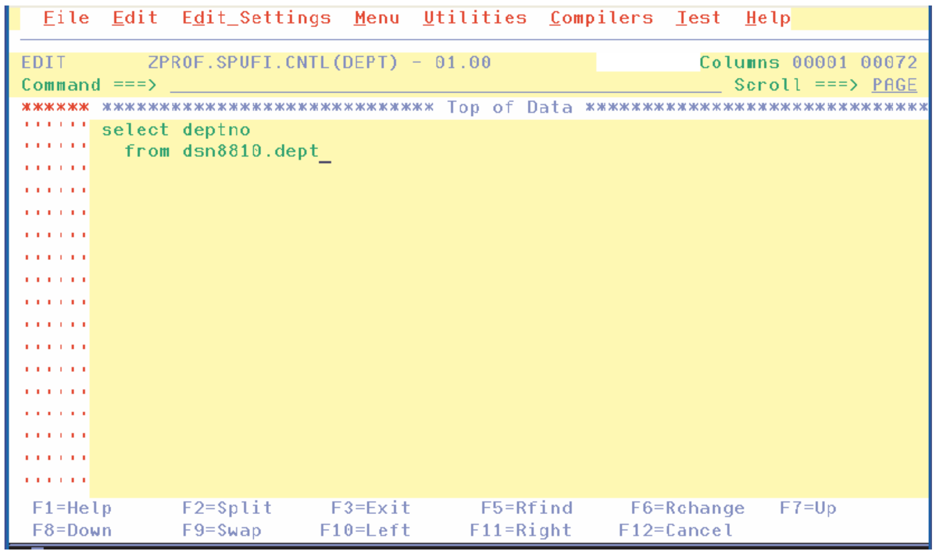


```
                            SPUFI                           SSID: DB8H
===>

Enter the input data set name:        (Can be sequential or partitioned)
 1  DATA SET NAME ... ===> 'ZPROF.SPUFI.CNTL(dept)'
 2  VOLUME SERIAL ... ===>            (Enter if not cataloged)
 3  DATA SET PASSWORD ===>            (Enter if password protected)

Enter the output data set name:       (Must be a sequential data set)
 4  DATA SET NAME ... ===> 'ZPROF.SPUFI.OUTPUT'

Specify processing options:
 5  CHANGE DEFAULTS   ===> NO         (Y/N - Display SPUFI defaults panel?)
 6  EDIT INPUT ...... ===> YES        (Y/N - Enter SQL statements?)
 7  EXECUTE ......... ===> YES        (Y/N - Execute SQL statements?)
 8  AUTOCOMMIT ...... ===> YES        (Y/N - Commit after successful run?)
 9  BROWSE OUTPUT ... ===> YES        (Y/N - Browse output data set?)

For remote SQL processing:
10  CONNECT LOCATION  ===>


 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 19. Assigning SPUFI data sets

Notice option 5, which you can change to YES temporarily to see the default values. One value you might want to change is the maximum number of rows retrieved.

With option 5 at NO, if you press the Enter key, SPUFI will put you in the input file, ZPROF.SPUFI.CNTL(DEPT), in order to enter or edit an SQL statement. By entering `recov on` in the command and pressing Enter, the warning on top of the screen will disappear. This option is part of the profile. The screen is shown in Figure 20.



```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

 EDIT       ZPROF.SPUFI.CNTL(DEPT) - 01.00                 Columns 00001 00072
 Command ===> ____________________________________________    Scroll ===> PAGE
 ****** ***************************** Top of Data ******************************
 ''''''  select deptno
 ''''''    from dsn8810.dept_
 ''''''
 ''''''
 ''''''
 ''''''
 ''''''
 ''''''
 ''''''
 ''''''
 ''''''
 ''''''
 ''''''
 ''''''
 ''''''
 ''''''
 ''''''
  F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
  F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

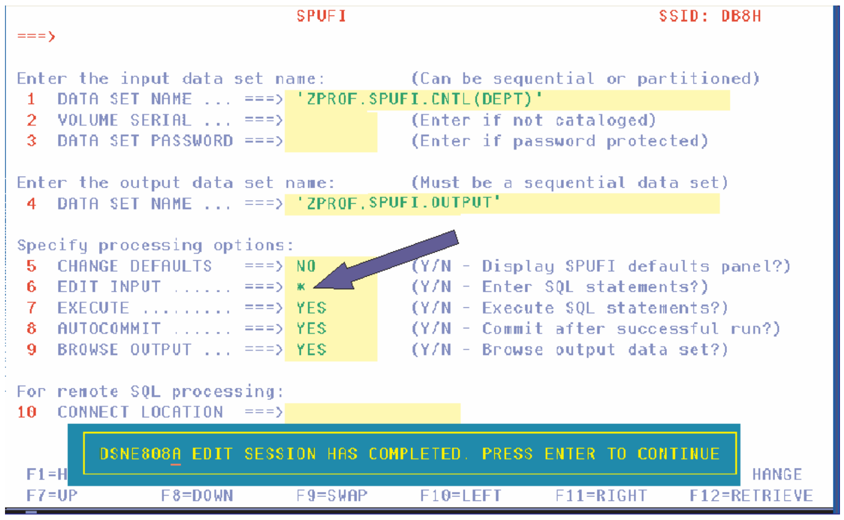
Figure 20. Editing the input file

If your profile is set to CAPS ON, the SQL statement you have just typed will normally change to capital letters at the Enter. But this is not needed.

Notice that we have mentioned DSN8810.DEPT as table name. This is the qualified name of the table, since we want to use the sample tables, which are created by user DSN8810.

If you enter just one SQL statement, you do not need to use the SQL terminator, which is a semi-colon (;), since this is specified in the defaults (but you can change this; remember option 5 of the previous screen). However, if you enter more than one SQL statement, you need to use a semicolon at the end of each statement to indicate that you have more than one.

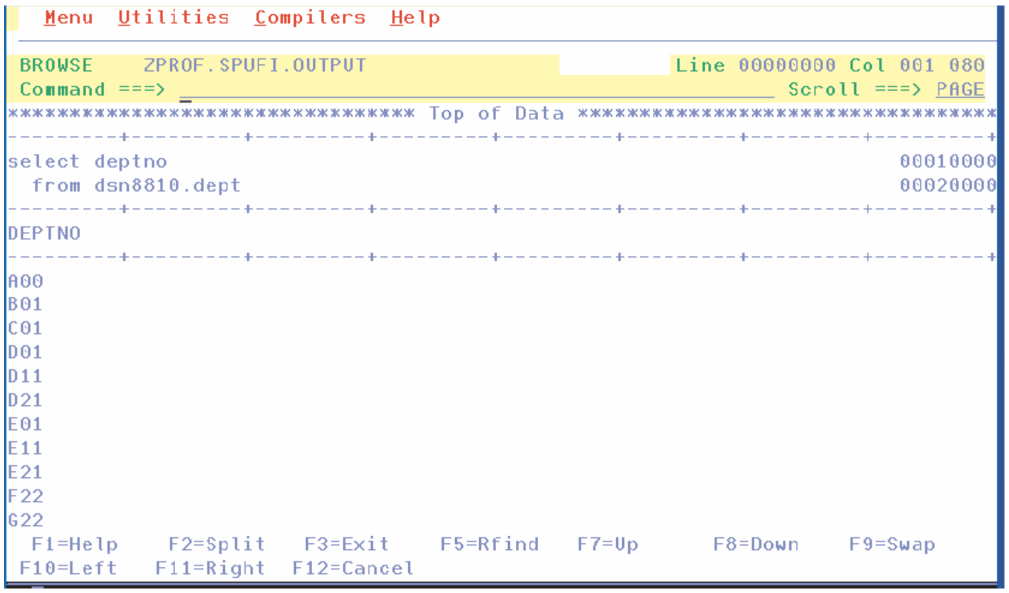
At this point, you need to go back to the first panel of SPUFI by pressing the F3 key. You will then see Figure 21.



```
                         SPUFI                                  SSID: DB8H
===>

Enter the input data set name:        (Can be sequential or partitioned)
 1  DATA SET NAME ... ===> 'ZPROF.SPUFI.CNTL(DEPT)'
 2  VOLUME SERIAL ... ===>            (Enter if not cataloged)
 3  DATA SET PASSWORD ===>            (Enter if password protected)

Enter the output data set name:       (Must be a sequential data set)
 4  DATA SET NAME ... ===> 'ZPROF.SPUFI.OUTPUT'

Specify processing options:
 5  CHANGE DEFAULTS   ===> NO         (Y/N - Display SPUFI defaults panel?)
 6  EDIT INPUT ...... ===> *          (Y/N - Enter SQL statements?)
 7  EXECUTE ......... ===> YES        (Y/N - Execute SQL statements?)
 8  AUTOCOMMIT ...... ===> YES        (Y/N - Commit after successful run?)
 9  BROWSE OUTPUT ... ===> YES        (Y/N - Browse output data set?)

For remote SQL processing:
10  CONNECT LOCATION  ===>

  DSNE808A EDIT SESSION HAS COMPLETED. PRESS ENTER TO CONTINUE
 F1=H                                                                HANGE
 F7=UP        F8=DOWN       F9=SWAP      F10=LEFT      F11=RIGHT     F12=RETRIEVE
```

Figure 21. Returning to the first SPUFI panel

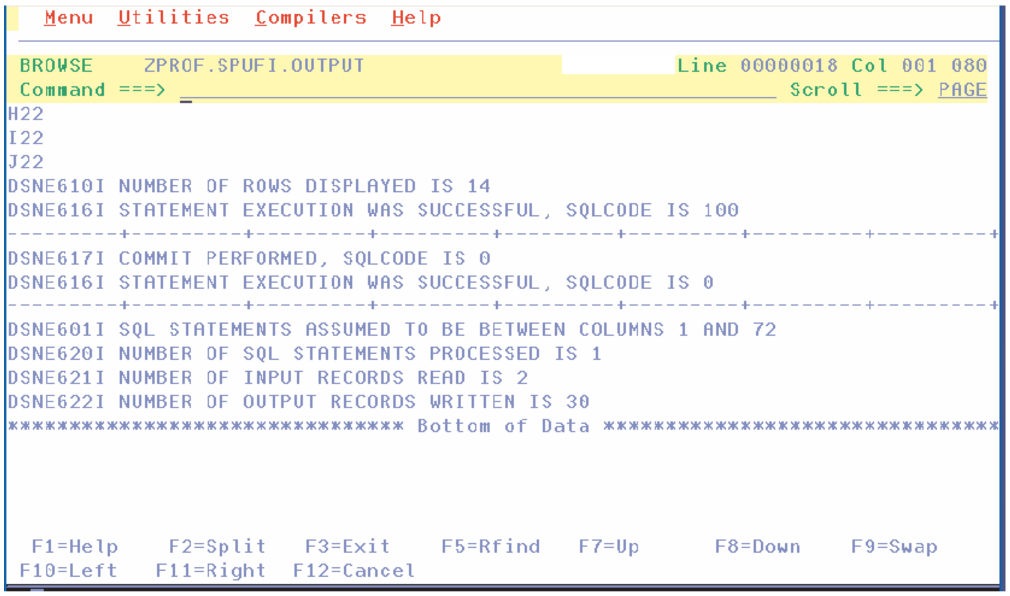
Notice that there is an asterisk (*) for option 6 since you just finished editing your SQL. At this point, if you press Enter, you will execute your SQL statement and you will automatically be put into your output file, since BROWSE OUTPUT is set to YES. The first part of the output is shown in Figure 22 on page 39.



```
   Menu  Utilities  Compilers  Help
 BROWSE    ZPROF.SPUFI.OUTPUT                           Line 00000000 Col 001 080
 Command ===> _                                                Scroll ===> PAGE
********************************* Top of Data **********************************
---------+---------+---------+---------+---------+---------+---------+---------+
select deptno                                                           00010000
  from dsn8810.dept                                                     00020000
---------+---------+---------+---------+---------+---------+---------+---------+
DEPTNO
---------+---------+---------+---------+---------+---------+---------+---------+
A00
B01
C01
D01
D11
D21
E01
E11
E21
F22
G22
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

Figure 22. First part of the SPUFI query results

To get the second (and in this case, final) result screen, press F8, and you will see Figure 23.



```
   Menu  Utilities  Compilers  Help
 BROWSE    ZPROF.SPUFI.OUTPUT                           Line 00000018 Col 001 080
 Command ===> _                                                Scroll ===> PAGE
H22
I22
J22
DSNE610I NUMBER OF ROWS DISPLAYED IS 14
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE617I COMMIT PERFORMED, SQLCODE IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE601I SQL STATEMENTS ASSUMED TO BE BETWEEN COLUMNS 1 AND 72
DSNE620I NUMBER OF SQL STATEMENTS PROCESSED IS 1
DSNE621I NUMBER OF INPUT RECORDS READ IS 2
DSNE622I NUMBER OF OUTPUT RECORDS WRITTEN IS 30
******************************** Bottom of Data ********************************



  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

Figure 23. Second result screen

Notice that you have a result table with just one column. This is what was specified in SELECT, just DEPTNO. We have retrieved the DEPTNO from all the (14) rows in the table. There are a few messages. One gives the number of rows retrieved. Another indicates that SQLCODE (an SQL return code indicating success or not) is 100, which means end of file, therefore no more results to show.

Related reading: For more information about SQL, see the IBM publication, *DB2 UDB for z/OS: SQL Reference*. You can find this and other related publications at the z/OS Internet Library Web site:

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

Application programming for DB2

SQL is not a full programming language, but it is necessary for accessing and manipulating data in a DB2 database. It is a 4GL nonprocedural language that was developed in the mid 1970s to use with DB2. SQL can either be used dynamically with an interpretive program like SPUFI, or it can be imbedded and compiled or assembled in a host language.

So how do you write an application program that accesses DB2 data?

To do this, SQL is embedded in the source code of a programming language, such as Java, Smalltalk, REXX™, C, C++, COBOL, Fortran, PL/I, and high-level Assembler. There are two categories of SQL statements that can be used in a program: static and dynamic.

- Static

  SQL refers to complete SQL statements that are written in the source code. In the program preparation process, DB2 develops access paths for the statements, and these are recorded in DB2. The SQL never changes from one run to another, and the same determined access paths are used without DB2 having to create them again, a process that can add overhead.

  Note: All SQL statements must have an access path.
- Dynamic

  SQL refers to SQL statements that are only partially or totally unknown when the program is written. Only when the program runs does DB2 know what the statements are and is able to determine the appropriate access paths. These do not get recorded since the statements can change from one run to another. An example of this is SPUFI. SPUFI is actually an application program that accepts dynamic SQL statements. These are the SQL statements that you enter in the input file. Each time you use SPUFI, the SQL can change, so special SQL preparation statements are embedded in the application to handle this.

DB2 program preparation: the flow

We now concentrate on Static SQL, to get an idea of the processes involved when using DB2. We also want to add that it may seem complex, but each action has a good reason for being there.

The traditional program preparation process, compile and linkedit, must have some additional steps to prepare SQL because compilers do not recognize SQL. These steps, including compile and linkedit, can be done with the DB2I panel, although the whole process is usually done in one JCL jobstream except for DCLGEN. Use Figure 24 on page 42 to follow the explanations.

DCLGEN

DCLGEN is a way to automatically generate your source definitions for the DB2 objects that will be used in your program. This is set up in a member of a DCLGEN library that can optionally be included in your source program. If you do not include it, you must manually code the definitions. The DB2 database administrator usually creates these, based on the company's rules. During this phase, you need a running DB2 system, because the definitions are taken from the DB2 catalog.

PRECOMPILE

Because compilers cannot handle SQL, the precompile step comments out the SQL statements and leaves behind a CALL statement to DB2. This passes some parameters such as host variable addresses (to retrieve data into), statement numbers, and (very importantly!) a modified timestamp called a **consistency token** (but often referred to as the timestamp). During this phase, you do not need a running DB2 system--everything is done without accessing DB2.

The precompiler identifies the SQL by special beginning and ending flags that must be included for each SQL statement. The beginning flag, EXEC SQL, is the same for all programming languages. The ending flag differs. COBOL uses END-EXEC. (period), while C and other languages use a semi-colon. Here is a COBOL example:

```
EXEC SQL
     SELECT EMPNO, LASTNAME
      INTO :EMPNO, :LASTNAME
      FROM EMP
END-EXEC.
```

In this example, EMPNO and LASTNAME are retrieved into host variables, which are preceded by a colon. Host variables (HVs) are variables defined in the “host” language (COBOL, PL/1, and so on), the language that embeds the SQL. During the DCLGEN phase, a set of these variables are also defined. The HV name is here the same as the column name, which is not a requirement–it can be any name with a datatype compatible with the columns datatype.

After the precompile, our program is divided into two parts:

- The modified source code; this is the original source code, were the SQL is commented out and replaced by CALLs.
- The database request module (DBRM), which is usually a member of a PDS library and contains the SQL statements of the program.

The modified source code is passed on to the compiler to be compiled and linkedited to create an executable load module, just like any program that does not contain SQL.

By the way, you can embed any type of SQL into your program: DML, DDL, and DCL, as long as the authorization rules are respected.

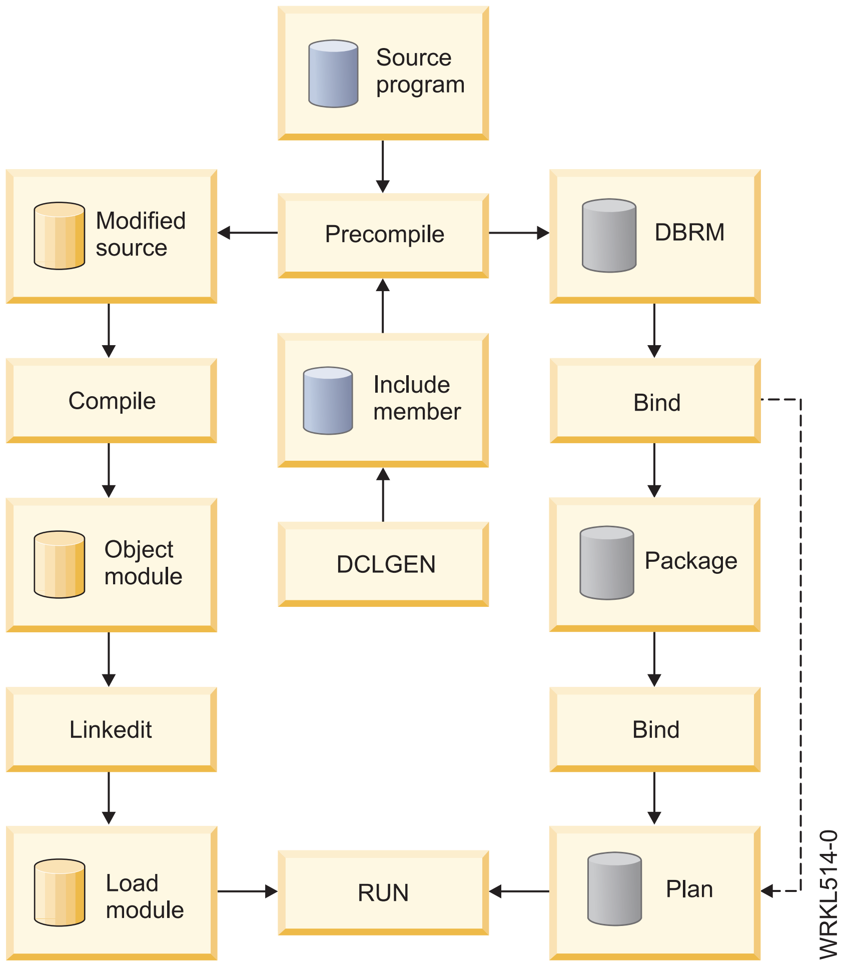


Figure 24. Program preparation flow

BIND

BIND can be thought of as the DB2 equivalent compile process for the DBRM. Bind does three things:

- Checks your syntax for errors.
- Checks authorization.
- Most importantly, it determines the access paths for your statements. DB2 has a component called the **optimizer**, which assesses all the different ways that your data can be accessed, such as scanning an entire table, using an index, which index, etc. It weighs the costs of each and picks the least. It is referred to as a cost-based optimizer (as opposed to a rule-based optimizer).

The SQL with its access path (and the consistency token/timestamp) is stored as a package in a DB2 directory. Other information, such as package information and the actual SQL, is stored in the catalog. The bind creates the executable SQL code for one application in a package. Now DB2 has all the information it needs to get to the requested data for this program.

Often programs call subroutines, which also contain SQL calls. Each of these subroutines then also has a package. You need to group all DB2 information together. Therefore, we need another step: another bind, but this time to create a **plan**.

Even if you are not using a subroutine, you still need to create a plan. The plan may contain more information than just your program info. This is common practice: The plan contains all packages of one project and every run uses the same plan.

To be complete, we need to add that originally the DBRMs were bound straight into the plan (they are called instream DBRMs). However, if there is one small change to one of the programs, you need to rebind the whole plan. The same needs to be done when an index is added.

During this binding process, as you know, DB2 updates its directory and catalog. Updating means preventing other people from updating (data is locked from them), so most other actions against DB2 were nearly impossible. To avoid this constraint, packages were introduced. Now you only need to rebind the one package, so the duration of the update is very short, and impact on other users is almost zero. There are still plans around with instream DBRMs, although most companies choose to convert them into packages.

Plans are unique to the mainframe environment. Other platforms do not use them.

RUN

When you execute your application program, the load module is loaded into main storage. When an SQL statement is encountered, the CALL to DB2, which replaced the SQL statement, passes its parameters to DB2. One of those is the consistency token. This token, or timestamp, is also in the package. The packages in the specified plan of DB2 are then searched for the corresponding timestamp, and the appropriate package is loaded and executed. So, for the run, you need to specify the plan name as a parameter.

One last note: The result of an SQL statement is usually a result set (more than one row). An application program can only deal with one record, or row, at a time. There is a special construction added to DB2, called a **cursor** (essentially a pointer), which allows you, in your embedded SQL, to fetch, update, or delete one row at a time, from your result set.

Related reading: To learn more, see the IBM publication, *DB2 UDB for z/OS: Application Programming and SQL Guide*.

Functions and structure of the IMS Database Manager

The IMS Database Manager provides a central point for the control and access to application data.

A database management system (DBMS) provides facilities for business application transactions or processes to access stored information. The role of a DBMS is to provide the following functions:
- Allow access to the data for multiple users from a single copy of the data.
- Control concurrent access to the data so as to maintain integrity for all updates.
- Minimize hardware device and operating system access method dependencies.
- Reduce data redundancy by maintaining only one copy of the data.

IMS provides a full set of utility programs to provide all these functions within the IMS product. This section describes the various types of z/OS address spaces and their relationships with each other.

The core of an IMS subsystem is the **control region**, running in one z/OS address space. This has a number of dependent address spaces running in other regions that provide additional services to the control region, or in which the IMS application programs run.

In addition to the control region, some applications and utilities used with IMS run in separate batch address spaces. These are separate from an IMS subsystem and its control region, and have no connection with it.

For historical reasons, some documents describing IMS use the term region to describe a z/OS address space, for example, IMS Control Region. In this course we use the term region wherever this is in common usage. You can take the term region as being the same as a z/OS address space.

Figure 25 illustrates the IMS DB/DC subsystem. If you want more details, we refer you to *An Introduction to IMS* (ISBN 0-13-185671-5).

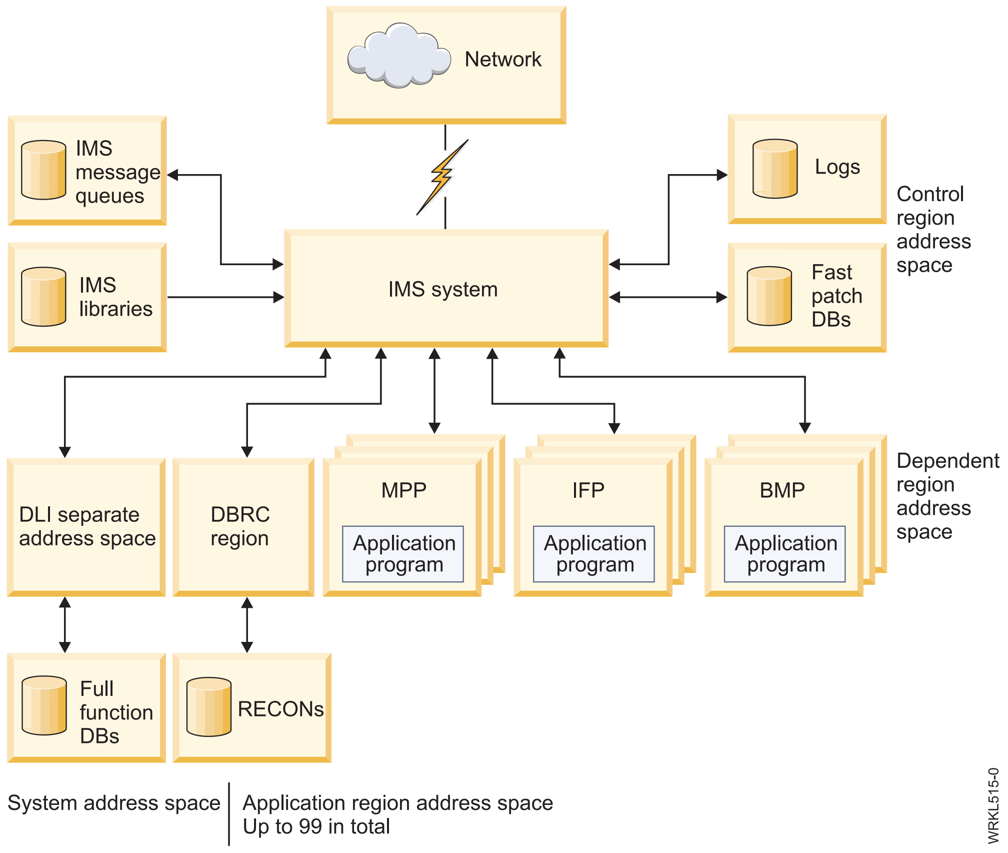


Figure 25. Structure of the IMS DB/DC subsystem

The IMS hierarchical database model

IMS uses a hierarchical model as the basic method for storing data, which is a pragmatic way of storing the data and implementing the relationships between the various types of entities.

In this model, the individual entity types are implemented as segments in a hierarchical structure. The hierarchical structure is determined by the designer of the database, based on the relationships between the entities and the access paths required by the applications.

Note that in the IMS program product itself, the term database is used slightly differently from its use in other DBMSs. In IMS, a database is commonly used to describe the implementation of one hierarchy, so that an application would normally access a large number of IMS databases. Compared to the relational model, an IMS database is approximately equivalent to a table.

DL/I allows a wide variety of data structures. The maximum number of segment types is 255 per hierarchical data structure. A maximum of 15 segment levels can be defined in a hierarchical data structure. There is no restriction on the number of occurrences of each segment type, except as imposed by physical access method limits.

Sequence to access the segments

The sequence of traversing the hierarchy is top to bottom, left to right, front to back (for twins).

Segment code numbers do not take twins into account and sequential processing of a database record is in hierarchic sequence. All segments of a database record are included so twins do have a place in hierarchic sequences. Segments may contain sequence fields that determine the order in which they are stored and processed.

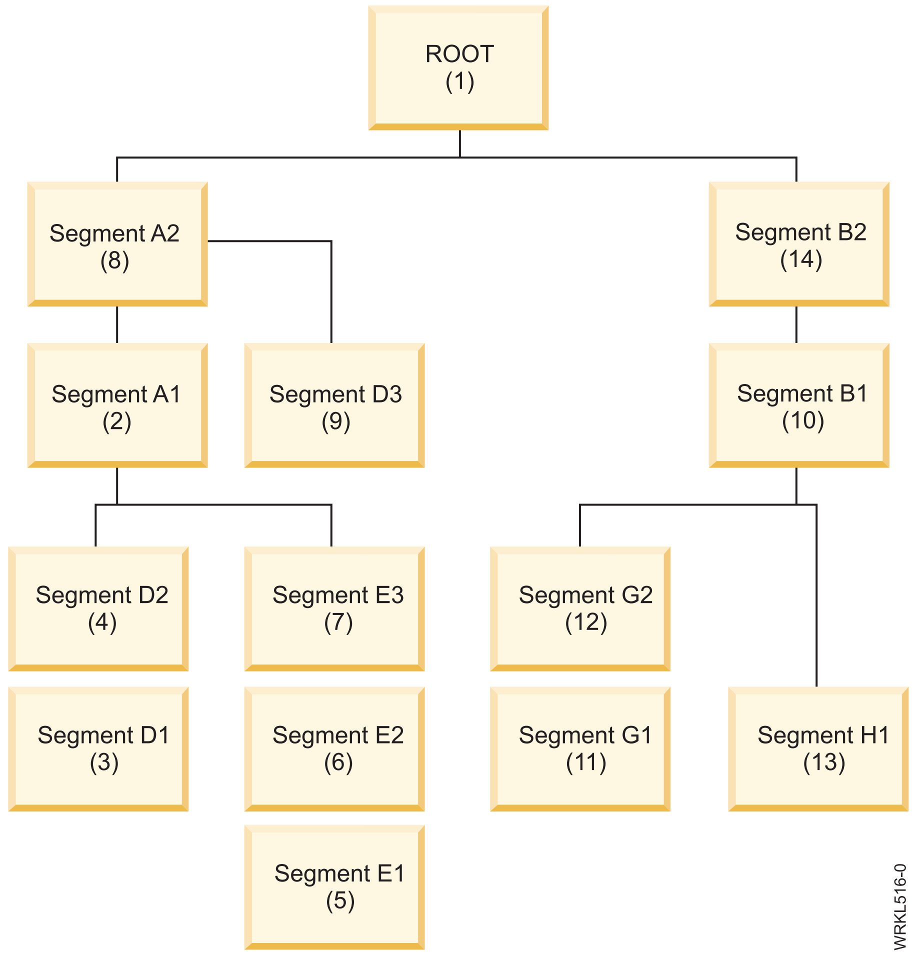


Figure 26. The sequence

The hierarchical data structure in Figure 26 describes the data of one database record as seen by the application program. It does not represent the physical storage of the data. The physical storage is of no concern to the application program.

The basic building element of a hierarchical data structure is the parent/child relationship between segments of data, also illustrated in Figure 26.

IMS use of z/OS services

IMS is designed to make the best use of the features of the z/OS operating system. This includes:

- It runs in multiple address spaces.

  IMS subsystems (except for IMS/DB batch applications and utilities) normally consist of a control region address space, dependent address spaces providing system services, and dependent address spaces for application programs. Running in multiple address spaces gives the following advantages:

  – Maximizes use of CPUs when running on a multiple-processor CPC.

  – Address spaces can be dispatched in parallel on different CPUs.

  – Isolates the application programs from the IMS systems code, and reduces outages from application failures.

- It runs multiple tasks in each address space.

IMS, particularly in the control regions, creates multiple z/OS subtasks for the various functions to be performed. This allows other IMS subtasks to be dispatched by z/OS while one IMS subtask is waiting for system services.

- It uses z/OS cross-memory services to communicate between the various address spaces making up an IMS subsystem. It also uses the z/OS Common System Area (CSA) to store IMS control blocks that are frequently accessed by the address spaces making up the IMS subsystem. This minimizes the overhead of running in multiple address spaces.
- It uses the z/OS subsystem feature to detect when dependent address spaces fail, to prevent cancellation of dependent address spaces, and to interact with other subsystems such as DB2 and WebSphere MQ.
- It can make use of a z/OS sysplex. Multiple IMS subsystems can run on the z/OS systems making up the sysplex and access the same IMS databases. This provides:
  - Increased availability–z/OS systems and IMS subsystems can be switched in and out without interrupting the service.
  - Increased capacity–The multiple IMS subsystems can process far greater volumes.

Evolution of IMS

Initially, all IMS/DB online applications used IMS/TM as the interface to the database. However, with the growing popularity of DB2, many customers began to develop online applications using DB2 as a database, next to their existing good applications. That is why you see a lot of mixed environments in the real world.

IMS online example

In our travel agent example, examples of IMS transactions could be in the part of the airline company:

- Some of the batch may be to update daily, such as the payments executed by travel agents and other customers.
- Another batch part may be the reminders to send out to the travel agents and other customers to make some payment.
- Checking whether reservations are made (and paid) can be an online application.
- Checking whether there are available seats.

Chapter 3. z/OS HTTP Server

As enterprises move many of their applications to the Web, mainframe organizations face the complexity of enabling and managing new Web-based workloads in addition to more traditional workloads, such as batch processing. z/OS HTTP Server serves static and dynamic Web pages. HTTP Server has the same capabilities as any other Web server, but it also has some features that are z/OS-specific.

Note: This section is about z/OS HTTP Server. Other sections are about WebSphere Application Server on z/OS and Messaging and queuing. These sections use IBM products in the examples, but many such middleware products exist in the marketplace today.

What is z/OS HTTP Server?

z/OS HTTP Server serves static and dynamic Web pages. You can run HTTP Server in any of three modes, with each offering advantages for handling Web-based workloads:

Stand-alone server
This mode is typically used for HTTP Server-only implementations (simple Web sites). Its main role is to provide a limited exposure to the Internet.

Scalable server
This mode is typically used for interactive Web sites, where the traffic volume increases or declines dynamically. It is intended for a more sophisticated environment, in which servlets and JSPs are invoked.

Multiple servers
This mode uses a combination of stand-alone and scalable servers to improve scalability and security throughout the system. For example, a stand-alone server could be used as a gateway to scalable servers, and the gateway could verify the user authentication of all requests, and reroute requests to the other servers.

Serving static Web pages on z/OS

With a Web server on z/OS, such as HTTP Server, the serving of static Web pages is similar to Web servers on other platforms. The user sends an HTTP request to HTTP Server to obtain a specific file. HTTP Server retrieves the file from its file repository and sends it to the user, along with information about the file (such as mime type and size) in the HTTP header.

HTTP Server has a major difference from other Web servers, however. Because z/OS systems encode files in EBCDIC, the documents on z/OS must first be converted to the ASCII format typically used on the Internet (binary documents such as pictures need not be converted).

HTTP Server performs these conversions, thus saving the programmer from performing this step. However, the programmer must use FTP to load documents on the server. That is, the programmer specifies ASCII as the FTP transport format to have the file converted from EBCDIC. For binary transfers, the file is not converted.

Serving dynamic Web pages on z/OS

Dynamic Web pages are an essential part of Web-based commerce. Every kind of interaction and personalization requires dynamic content. When a user fills out a form on a Web site, for example, the data in the form must be processed, and feedback must be sent to the user.

Two approaches for serving dynamic Web pages on z/OS are:

- Using CGI for dynamic Web pages
- Using the plug-in interface.

Using CGI for dynamic Web pages

One way to provide dynamic Web pages is through the Common Gateway Interface (CGI), which is part of the HTTP protocol. CGI is a standard way for a Web server to pass a Web user's HTTP request to an application. GCI generates the output and passes it back to HTTP Server, which sends it back to the user in an HTTP response (Figure 27).

CGI is not limited to returning only HTML pages; the application can also create plain text documents, XML documents, pictures, PDF documents, and so on. The mime type must reflect the content of the HTTP response.

CGI has one major disadvantage, which is that each HTTP request requires a separate address space. This causes a lack of efficiency when there are many requests at a time.

To avoid this problem, FastCGI was created. Basically, the HTTP Server FastCGI plug-in is a program that manages multiple CGI requests in a single address space, which saves many program instructions for each request. More information about HTTP Server plug-ins is provided in "Using the plug-in interface."

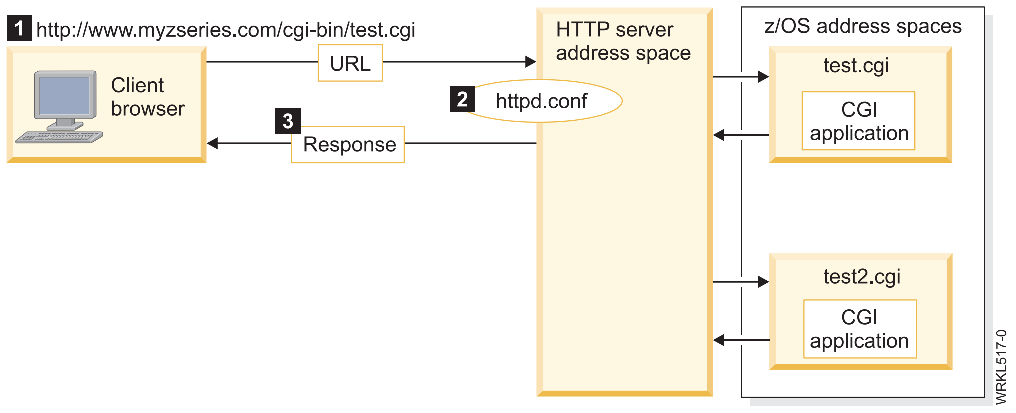


Figure 27. How the CGI works

Using the plug-in interface

Another way of providing dynamic content is by using the plug-in interface of HTTP Server, which allows one of several products to interface with HTTP Server. Here, for example, are some ways in which HTTP Server can pass control to WebSphere;

- WebSphere plug-in, same address space

Figure 28 shows a simple configuration in which no J2EE server is needed. This servlet can connect to CICS or IMS, or to DB2 through JDBC. However, coding business logic inside servlets is not recommended.

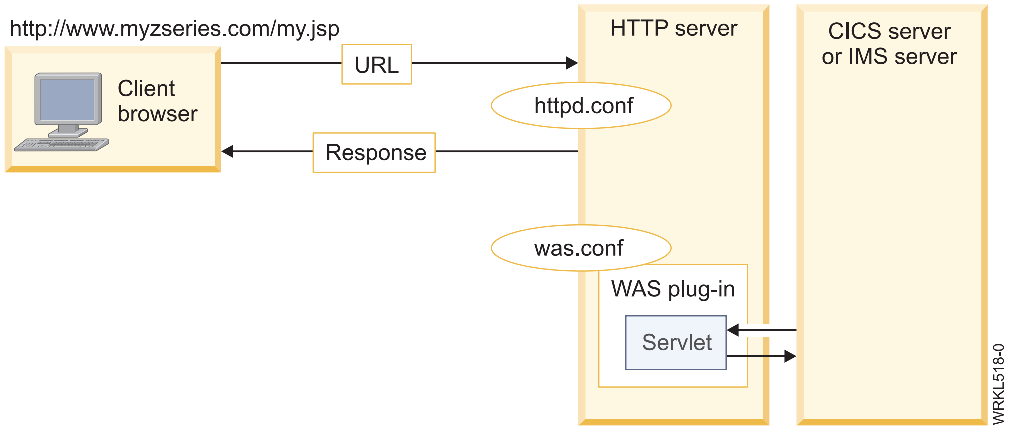


Figure 28. Accessing servlets using the WebSphere plug-in

- Web container inside HTTP Server, separate EJB container

  Figure 29 shows a more usable configuration in which the servlets run in a different address space than the EJBs, so the EJBs are invoked from remote calls. The EJBs then get information from other servers.

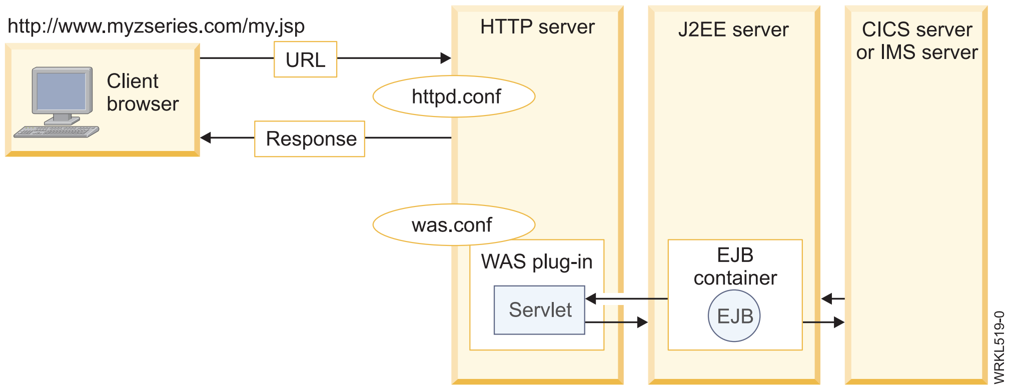


Figure 29. Accessing EJBs from a WebSphere plug-in

- Separate J2EE server with both Web container and EJB container

  In addition to running your servlets locally within the WebSphere plug-in, you can also use the WebSphere plug-in to run servlets remotely in a Web container, as shown in Figure 30 on page 52. This allows you to localize your servlets and EJBs to the same z/OS address space, so that no remote EJB calls are required.

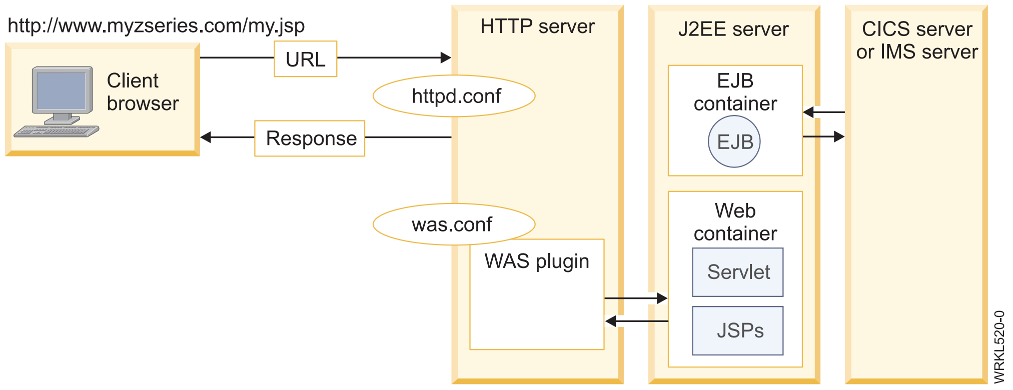


Figure 30. Accessing servlets in a Web container using the WebSphere plug-in

If you are using WebSphere Application Server, HTTP Server might not be needed, yet there are several ways in which HTTP Server can interact with WebSphere Application Server.

z/OS HTTP Server capabilities

HTTP Server provides capabilities similar to other Web servers, but with some functions specific to z/OS as well.

The z/OS-specific functions can be grouped as follows:
- Basic functions
- Security functions
- File caching.

z/OS HTTP Server basic functions

HTTP Server basic functions include EBCDIC/ASCII file access, performance and usage monitoring, tracing and logging, server side include (SSI), Simple Network Management Protocol (SNMP) Management Information Base (MIB), cookies support, Multi-Format Processing, persistent connections, and virtual hosts.

- EBCDIC/ASCII file access

  The server accesses files and converts them, if needed, from EBCDIC to ASCII encoding.
- Performance and usage monitoring

  As part of the z/OS features, HTTP Server can produce system management facilities (SMF) records that the system programmer can retrieve later to do performance and usage analysis. (SMF is an optional feature of z/OS that provides you with the means for gathering and recording information that can be used to evaluate system usage for accounting, charge-back, and performance tuning.)
- Tracing and logging

  HTTP Server comes with a complete set of logging, tracing, and reporting capabilities that allow you to keep track of every HTTP request.
- Server Side Include (SSI)

  Server Side Include allows you to insert information into documents (static or dynamic) that the server sends to the clients. This could be a variable (like the ″Last modified″ date), the output of a program, or the content of another file. Enabling this function, but not using it, can have a serious performance impact.

- Simple Network Management Protocol (SNMP) Management Information Base (MIB)

  HTTP Server provides an SNMP MIB and SNMP subagent, so you can use any SNMP-capable network management system to monitor your server’s health, throughput, and activity. It can then notify you if specified threshold values are exceeded.
- Cookies support

  Because HTTP is a stateless protocol, a state can be added with the help of cookies, which store information on the client’s side. This support is useful for multiple Web pages, for example to achieve customized documents or for banner rotation.
- Multi-Format Processing

  This feature is used for personalization of Web pages. The browser sends header information along with the request, including the **accept header**. This information includes the language of the user. HTTP Server can make use of the contents of the accept header to select the appropriate document to return to the client.
- Persistent connections

  With the help of this HTTP/1.1-specific feature, not every request has to establish a new connection. Persistent connections stay “alive” for a certain amount of time to enable the use of a given connection to another request.
- Virtual hosts

  Virtual hosts allow you to run one Web server while making it appear to clients as if you are running several. This is achieved by the use of different DNS names for the same IP and/or different IP addresses bound to the same HTTP Server.

z/OS HTTP Server security functions

HTTP Server security functions include thread level security, HTTPS/SSL support, LDAP support, certificate authentication, and proxy support.

- Thread level security

  An independent security environment can be set for each thread running under HTTP Server, which basically means that every client connecting to the server will have its own security environment.
- HTTPS/SSL support

  HTTP Server has full support for the Secure Socket Layer (SSL) protocol. HTTPS uses SSL as a sublayer under the regular HTTP layer to encrypt and decrypt HTTP requests and HTTP responses. HTTPS uses port 443 for serving instead of HTTP port 80.
- LDAP support

  The Lightweight Data Access Protocol (LDAP) specifies a simplified way to retrieve information from an X.500-compliant directory in an asynchronous, client/server type of protocol.
- Certificate authentication

  As part of the SSL support, HTTP Server can use certificate authentication and act as a certificate authority.
- Proxy support

  HTTP Server can act as a proxy server. You cannot, however, use the Fast Response Cache Accelerator (FRCA).

z/OS file caching

Performance can be significantly increased by using certain file caching (buffering) possibilities.

- HTTP Server caching HFS files
- HTTP Server caching z/OS data sets
- z/OS UNIX caching HFS files
- Fast Response Cache Accelerator (FRCA).

WebSphere HTTP Server plug-in code

The WebSphere HTTP Server plug-in is code that runs inside various Web servers: IBM HTTP Server, Apache, IIS, Sun Java System. Requests are passed over to the plug-in, where they are handled based on a configuration file.

The plug-in is code supplied with WebSphere that runs inside various HTTP servers. Those HTTP servers may be the IBM HTTP Server on z/OS. As workload comes into the HTTP Server, directives in the HTTP Server's configuration file (httpd.conf) are used to make a decision: is the work request coming in something the HTTP Server handles or is it something that's to be passed over the plug-in itself.

Once inside the plug-in, the logic that acts upon the request is determined by the plug-in's configuration file, not the HTTP Server's. That configuration file is by default called the plugin-cfg.xml file. Information on which of the backed application servers the request is to go to is defined in this file. This file is something that is created by WebSphere Application Server and doesn't necessarily need modifying, although you have the flexibility to do so.

Note: In general, plug-ins provide functionality extensions for HTTP Server. Figure 31 on page 55 shows one example of its use, although there are many different plug-ins that can be configured to assist in customization of your Web environment. Another popular plug-in is the Lightweight Directory Access Protocol Server (LDAP) used for security authentication.

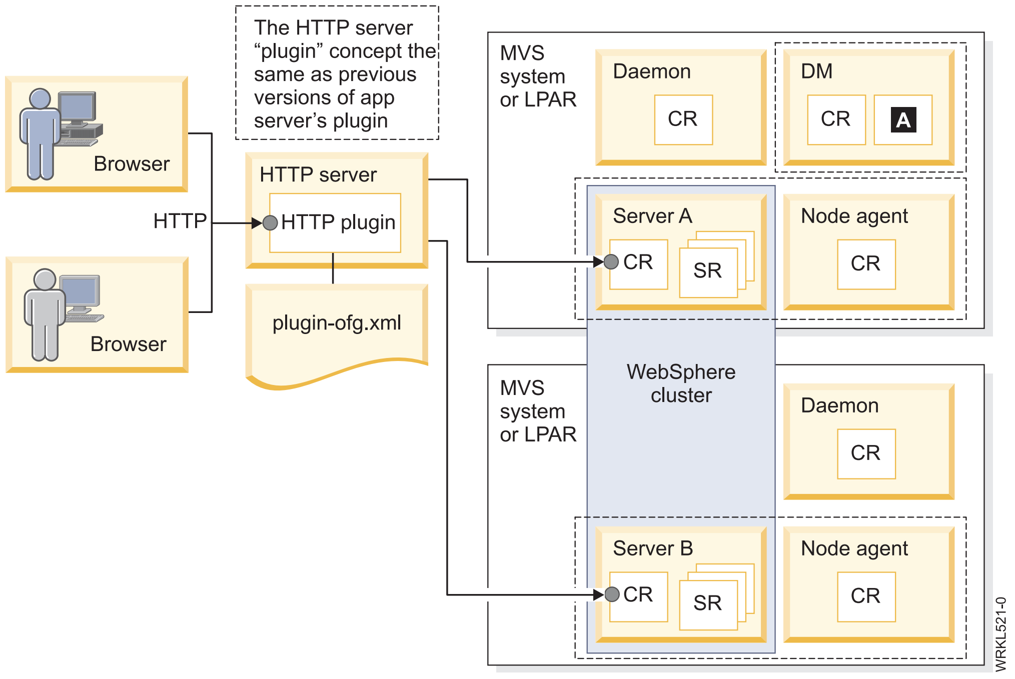


Figure 31. Example of a plug-in

Chapter 4. WebSphere Application Server on z/OS

As enterprises move many of their applications to the Web, mainframe organizations face the complexity of enabling and managing new Web-based workloads in addition to more traditional workloads, such as batch processing. WebSphere Application Server is a comprehensive, sophisticated, Java 2 Enterprise Edition (J2EE) and Web services technology-based application system.

Note: This section is about WebSphere Application Server on z/OS. Other sections are about z/OS HTTP Server and Messaging and queuing. These sections use IBM products in the examples, but many such middleware products exist in the marketplace today.

What is WebSphere Application Server for z/OS?

WebSphere Application Server on z/OS is the J2EE implementation conforming to the current Software Development Kit (SDK) specification supporting applications at an API level. As mentioned, it is a Java Application deployment and run-time environment built on open standards-based technology supporting all major functions such as servlets, Java server pages (JSPs), and Enterprise Java Beans (EJBs) including the latest technology integration of services and interfaces.

The application server runtime is highly integrated with all inherent features and services offered on z/OS. The application server can interact with all major subsystems on the operating system including DB2, CICS, and IMS. It has extensive attributes for security, performance, scalability and recovery. The application server also uses sophisticated administration and tooling functions, thus providing seamless integration into any data center or server environment.

WebSphere Application Server is an e-business application deployment environment. It is built on open standards-based technology such as CORBA, HTML, HTTP, IIOP, and J2EE-compliant Java technology standards for servlets, Java Server Pages (JSP) technology, and Enterprise Java Beans (EJB), and it supports all Java APIs needed for J2EE compliance.

The Controller Address Space will automatically start a servant region as work arrives. As shown in Figure 32, an application server instance is composed of a controller region (CR) and one or more servant regions (SRs).

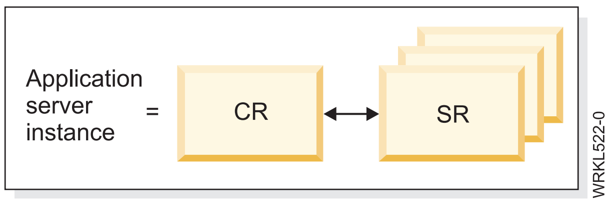


Figure 32. An application server instance

The application server on z/OS supports two types of configurations: Base and Network Deployment. Each configuration uses essentially the same architectural hierarchy, comprised of **servers**, **nodes** and **cells**. However, cells and nodes play an important role only in the Network Deployment configuration.

Servers

A server is the primary runtime component; this is where your application actually executes. The server provides containers and services that specialize in enabling the execution of specific Java application components. Each application server runs in its own Java Virtual Machine (JVM).

Depending on the configuration, servers might work separately or in combination, as follows:

- In a Base configuration, each application server functions as a separate entity. There is no workload distribution or common administration among the application servers.
- In a Network Deployment configuration, multiple application servers are maintained from a central administration point.

In addition, you can cluster application servers for workload distribution.

Note: A special type of application server called a JMS Server is not covered in this material.

Nodes (and node agents)

A node is a logical grouping of WebSphere-managed server processes that share common configuration and operational control. A node is generally associated with one physical installation of the application server.

As you move up to the more advanced application server configurations, the concepts of configuring multiple nodes from one common administration server and workload distribution among nodes are introduced. In these centralized management configurations, each node has a node agent that works with a Deployment Manager to manage administration processes.

Cells

A cell is a grouping of nodes into a single administrative domain. In the Base configuration, a cell contains one node. That node may have multiple servers, but the configuration files for each server are stored and maintained individually (XML-based).

With the Network Deployment configuration, a cell can consist of multiple nodes, all administered from a single point. The configuration and application files for all nodes in the cell are centralized into a cell master configuration repository. This centralized repository is managed by the Deployment manager process and synchronized out to local copies held on each of the nodes.

In the address spaces used for the application server, there is a concept of **containers**, which provide runtime separation between the various elements that execute. A single container, known as an EJB container, is used to run Enterprise Java Beans. Another container, known as the Web container, is used to execute Web-related elements such as HTML, GIF files, servlets, and Java server pages (JSPs). Together, they make up the application server runtime within the JVM.

J2EE application model on z/OS

The J2EE Application Model on z/OS is exactly the same as on other platforms.

The J2EE Application Model on z/OS follows the SDK specification, exhibiting the following qualities:

- Functional–satisfies user requirements
- Reliable–performs under changing conditions
- Usable–enables easy access to application functions
- Efficient–uses system resources wisely
- Maintainable–can be modified easily
- Portable–can be moved from one environment to another.

WebSphere Application Server on z/OS supports four major models of application design: Web-based computing, integrated enterprise computing, multithreading distributed business computing, and service-oriented computing. All these design models focus on separating the application logic from the underlying infrastructure; that is, the physical topology and explicit access to the information system are distinct from the programming model for the application.

The J2EE programming model supported by WebSphere Application Server for z/OS makes it easier to build applications for new business requirements because it separates the details from the underlying infrastructure. It provides for the deployment of the component and service-oriented programming model offered by J2EE.

Running WebSphere Application Server on z/OS

WebSphere Application Server runs as a standard subsystem on z/OS. Therefore, it inherits all the characteristics of mainframe qualities and functionality that accompany that platform, such as its unique capacity for running hundreds of heterogeneous workloads concurrently, and meeting service level objectives as defined by the user.

Consolidation of workloads

A mainframe can be used to consolidate workloads from many individual servers. Therefore, if there is a large administration overhead or a physical capacity concern of many individual servers, the mainframe can take on the role of a single server environment managing those workloads. It can present a single view of administration, performance and recovery for applications that harness the mainframe's services during execution.

Several application servers can easily be migrated into one logical partition of a mainframe's resources, thus providing ease of management and monitoring. Consolidation also allows for instrumentation and metric gathering, resulting in easier capacity analysis.

WebSphere for z/OS security

The combination of zSeries hardware- and software-based security, along with incorporated J2EE security, offers significant defense against possible intrusions. The product security is a layered architecture built on top of the operating system platform, the Java Virtual Machine (JVM), and Java2 security.

WebSphere Application Server for z/OS integrates infrastructure and mechanisms to protect sensitive J2EE resources and administrative resources addressing the enterprise from an end-to-end security perspective based on industry standards.

The open architecture possesses secure connectivity and interoperability with all mainframe Enterprise Information Systems, which includes:

- CICS Transaction Server (TS)
- DB2
- Lotus® Domino®
- IBM Directory

WebSphere Application Server integrates with RACF® and WebSEAL Secure Proxy (Trusted Association Interceptor), providing a unified, policy-based and permission-based model for securing all Web resources and Enterprise Java Bean components, as defined in the J2EE specification.

Continuous availability and WebSphere Application Server for z/OS

WebSphere for z/OS uses the zSeries platform's internal error detection and correction internal capabilities. WebSphere for z/OS has recovery termination management that detects, isolates, corrects and recovers from software errors. WebSphere for z/OS can differentiate and prioritize work based on service level agreements. It offers clustering capability as well as the ability to make non-disruptive changes to software components, such as resource managers.

In a critical application, WebSphere for z/OS can implement a failure management facility of z/OS called automatic restart manager or ARM. This facility can detect application failures, and restart servers when failures occur. WebSphere uses ARM to recover application servers (servants). Each application server running on a z/OS system is registered with an ARM restart group.

WebSphere for z/OS can implement a feature called **clustering**. Clustering technology is used extensively in high availability solutions involving WebSphere, as shown in Figure 33 on page 61.

A cluster consists of multiple copies of the same component with the expectation that at least one of the copies will be available to service a request. In general, the cluster works as a unit where there is some collaboration among the individual copies to ensure that the request can be directed toward a copy that is capable of servicing the request.

Designers of a high availability solution participate in establishing a service level as they determine the number and placement of individual members of clusters. WebSphere for z/OS provides management for some of the clusters needed to create the desired service level. Greater service levels of availability can be obtained as WebSphere clusters are supplemented with additional cluster technologies.

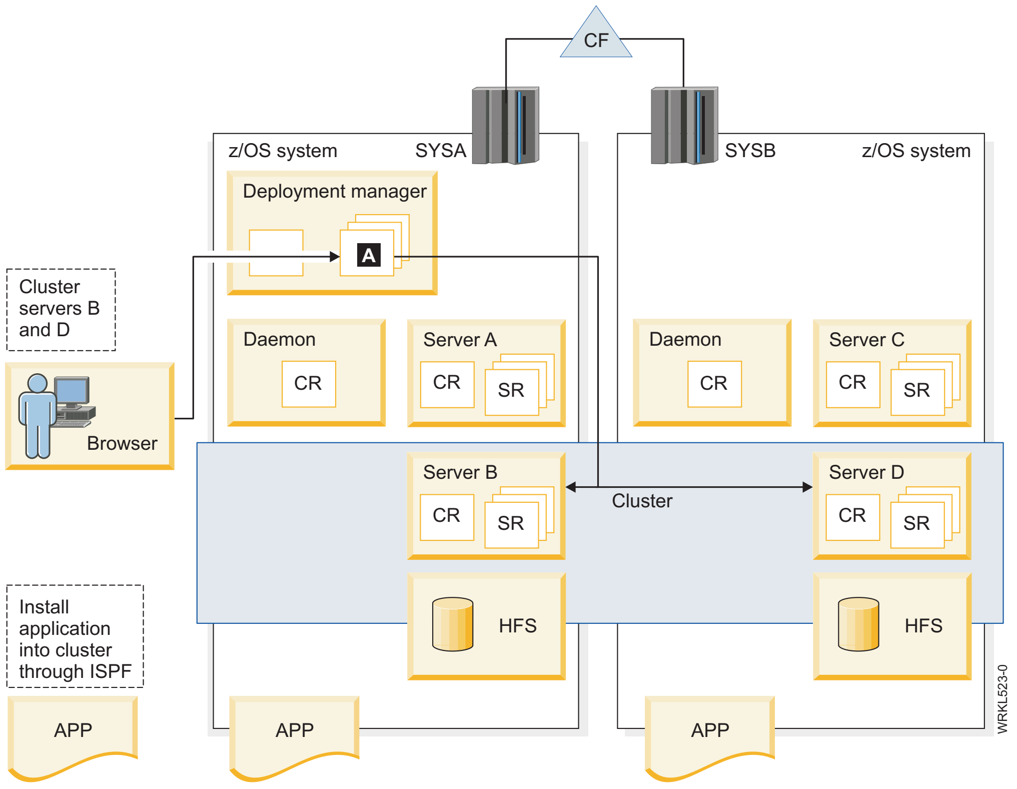


Figure 33. Clustering of servers in a cell

A WebSphere Application Server cluster is composed of individual cluster members, with each member containing the same set of applications. In front of a WebSphere Application Server cluster is a workload distributor, which routes the work to individual members.

Clusters can be vertical within an LPAR (that is, two or more members residing in a z/OS system) or they can be placed horizontally across LPARs to obtain the highest availability in the event an LPAR containing a member has an outage.

Workload in this case can still be taken on from the remaining cluster members. Also within these two configurations, it is possible to have a hybrid in which the cluster is composed of vertical and horizontal members (see Figure 34 on page 62).

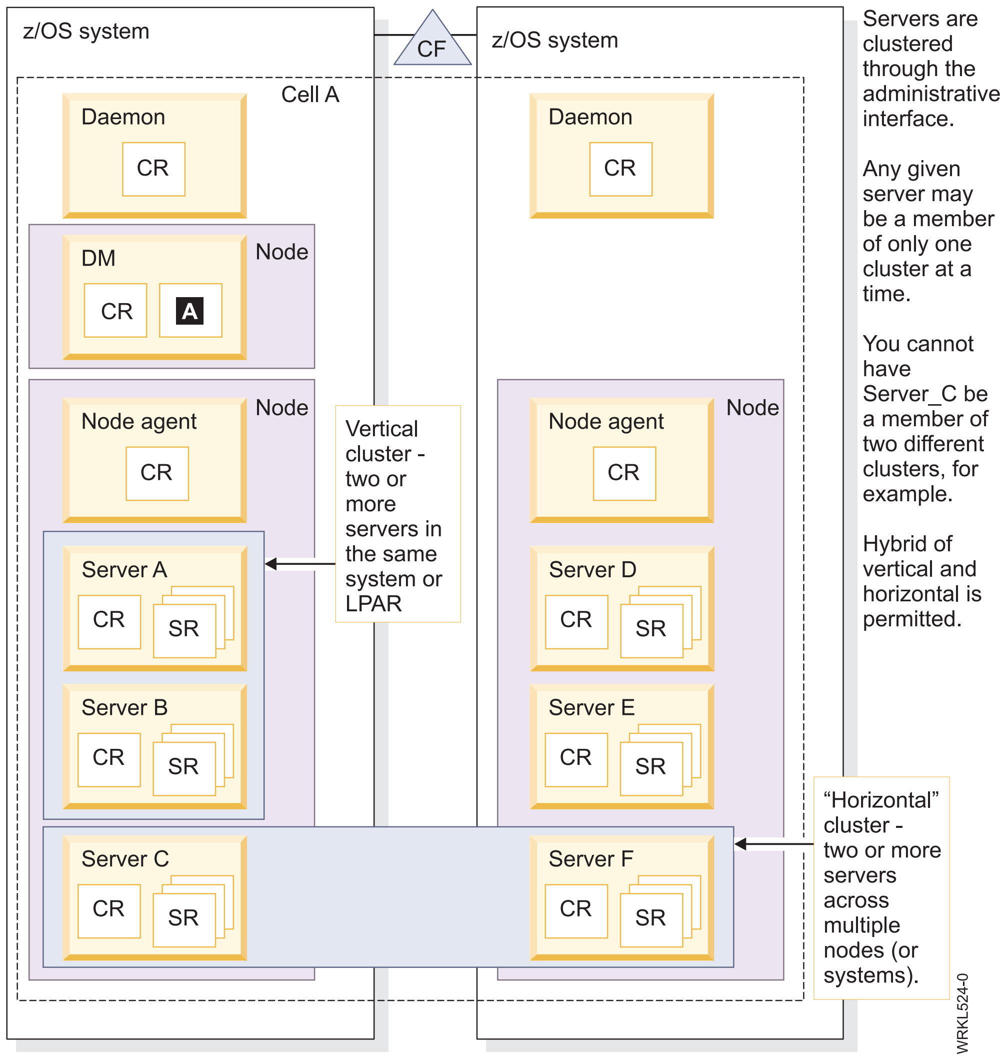

z/OS system
CF
z/OS system
Cell A
Daemon
CR
Daemon
CR
DM
CR
A
Node
Node agent
CR
Node
Vertical cluster - two or more servers in the same system or LPAR
Node agent
CR
Node
Server A
CR
SR
Server B
CR
SR
Server C
CR
SR
Server D
CR
SR
Server E
CR
SR
Server F
CR
SR
"Horizontal" cluster - two or more servers across multiple nodes (or systems).
Servers are clustered through the administrative interface.
Any given server may be a member of only one cluster at a time.
You cannot have Server_C be a member of two different clusters, for example.
Hybrid of vertical and horizontal is permitted.
WRKL524-0

Figure 34. Vertical and horizontal clusters

You might wonder when to use vertical clustering as opposed to horizontal clustering. You might use vertical clustering to check the dispatching efficiency of a single system. In a vertical cluster, the servers compete with each other for resources.

Performance and WebSphere Application Server for z/OS

Performance is highly dependent on application design and coding, regardless of the power of the runtime platform–a defectively written application will perform just as poorly on z/OS as it would on another platform. That said, WebSphere Application Server for z/OS uses the mainframe qualities in hardware, and software characteristics incorporating Workload Management schemes, dynamic LPAR configuration, and Parallel Sysplex® functionality.

Specifically, WebSphere Application Server for z/OS uses the three distinct functions of z/OS workload management (WLM):

- Routing

  WLM routing services are used to direct clients to servers on a specific system based on measuring current system utilization, known as the Performance Index (PI).

- Queuing

  The WLM queuing service is used to dispatch work requests from a Controller Region to one or more Server Regions. It is possible for a Work Manager to register with WLM as a Queuing Manager. This tells WLM that this server would like to use WLM-managed queues to direct work to other servers, which allows WLM to manage server spaces to achieve the specified performance goals established for the work.
- Prioritize

  The application server provides for starting and stopping Server Regions to set work priority. This allows WLM to manage application server instances in order to achieve goals specified by the business.

WLM maintains a performance index (PI) for each service class period to measure how actual performance varies from the goal. Because there are several types of goals, WLM needs some way of comparing how well or poorly work in one service class is doing compared to other work. A service class (SC) is used to describe a group of work within a workload with equivalent performance characteristics.

Application server configuration on z/OS

An application server configuration on z/OS includes base server node and the Network Deployment Manager.

Base server node

The base application server node is the simplest operating structure in the Application Server for z/OS. It consists of an application server and a daemon server (one node and one cell), as shown in Figure 35 on page 64. All of the configuration files and definitions are kept in the HFS directory structure created for this base application server. The daemon server is a special server with one controller region. The system architecture of WebSphere for z/OS calls for one daemon server per cell per system or LPAR.

Each base application server node contains administration for its own cell domain and a separate repository for its configuration. Therefore, you can have many base application servers, each isolated from the others, having their own administration policy for their specific business needs.

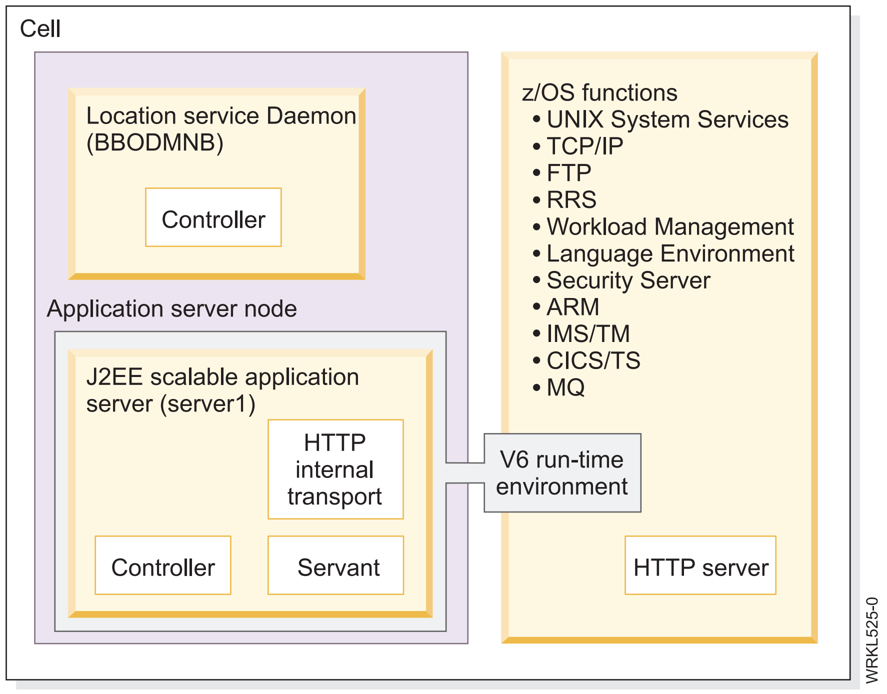


Figure 35. Base server node

Network Deployment Manager

Network Deployment Manager (Figure 36 on page 65) is an extension to the base application server. It allows the system to administer multiple application servers from one centralized location. Here, application servers are attached to nodes, and multiple nodes belong to a cell. With the Deployment Manager, horizontally and vertically scaled systems, as well as distributed applications, can be easily administered.

The Network Deployment Manager also manages the repositories on each node, performing such tasks as creating, maintaining, and removing the repositories. The system uses an extract/modify method to update the configuration.

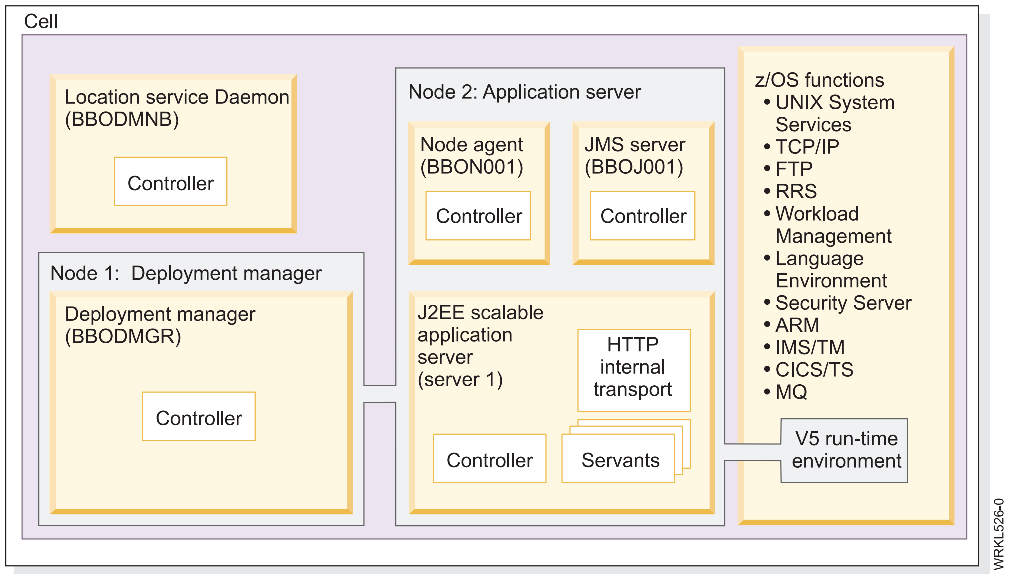


Figure 36. Network Deployment Manager

WebSphere Application Server connectors for Enterprise Information Systems

The ability of applications to interface with resources outside of the application server process and to use those resources efficiently has always been an important requirement. Equally important is the ability for vendors to plug in their own solutions for connecting to and using their resources.

An application might require access to many types of resources, which may or may not be located in the same machine as the application. Therefore, access to a resource begins with a connection that is a pathway from an application to a resource, which might be another transaction manager or database manager.

Java program access to a broad range of back-end resources is performed through a resource adapter. This is a system-level software driver that plugs into an application server and enables a Java application to connect to various back-end resources.

The following considerations are common to all connections:

- Creating a connection can be expensive. Setting up a connection can take a long time when compared to the amount of time the connection is actually used.
- Connections must be secure. This is often a joint effort between the application and the server working with the resource.
- Connections must perform well. Performance can be critical to the success of an application, and it is a function of the application's overall performance.
- Connections must be monitorable and have good diagnostics. The quality of the diagnostics for a connection depends on the information regarding the status of the server and the resource.
- Methods for connecting to and working with a resource. Different database architectures require different means for access from an application server.

- Quality of service, which becomes a factor when accessing resources outside of the application server. The application might require the ACID (Atomicity, Consistency, Isolation, and Durability) properties that can be obtained when using data in managing a transaction.

Enterprise resources are often older resources that were developed over time by a business and are external to the application server process. Each type of resource has its own connection protocol and proprietary set of interfaces to the resource. Therefore, the resource has to be adapted in order for it to be accessible from a JVM process as contained in an application server.

WebSphere Application Server has facilities to interface with other z/OS subsystems such as CICS, DB2 and IMS. This is done through a resource adapter and a connector. Accessing back-end Enterprise Information Systems (EIS) extends the functionality of the application server into existing business functions, providing enhanced capabilities.

The J2EE Connector Architecture defines the contracts between the application, the connector, and the application server where the application is deployed. The application has a component called the **resource adapter**. This is contained within the application code handling the interface to the connector which the application developer creates.

From a programming perspective, this means that programmers can use a single unified interface to obtain data from the EIS. The resource adapter will sort out the different elements and provide a programming model that is independent of the actual EIS behavior and communication requirements.

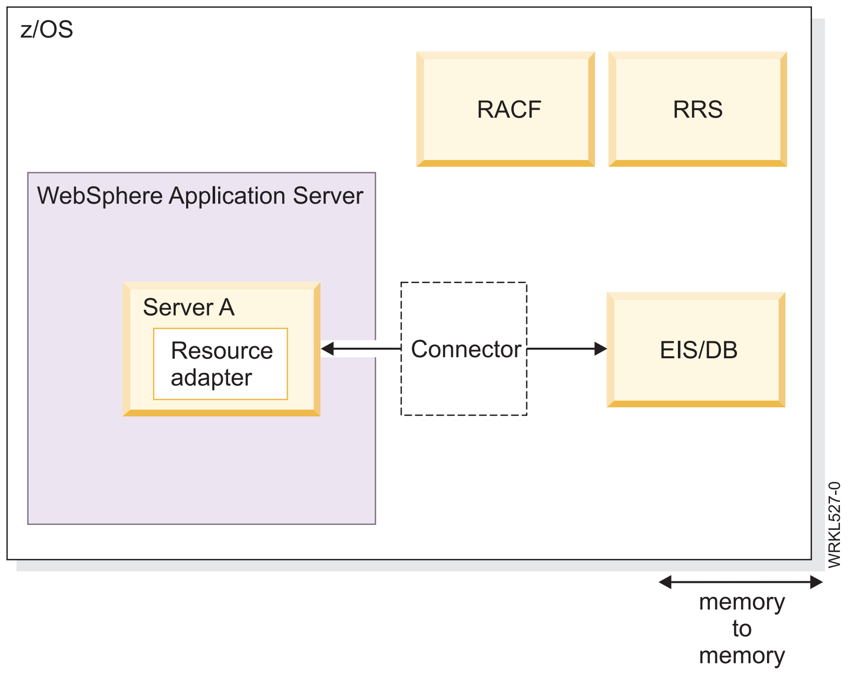


Figure 37. Basic architecture of a connector to an EIS

Java Connector Architecture (JCA) is a standard that provides a uniform method for talking to different kinds of EIS systems from an application server. JCA is designed to facilitate sharing of data and to integrate new J2EE applications with legacy or other heterogeneous systems.

WebSphere for z/OS supports any JCA-compliant resource adapters, but does not provide any of its own, with the exceptions of the built-in JMS adapter, and the relational resource adapter for database management systems, such as DB2.

z/OS Web application connectors

Web applications on z/OS can use connectors to access data in various mainframe middleware products.

This topic briefly describes the following commonly used connectors, which are used to access data in CICS, IMS, and DB2:
- CICS Transaction Gateway
- IMS Connect
- DB2 JDBC.

CICS Transaction Gateway

Customer Information Control System (CICS) has the CICS Transaction Gateway (CTG) to connect from the application server to CICS. CTG provides the interface between Java and CICS application transactions. It is a set of client and server software components incorporating the services and facilities to access CICS from the application server. CTG uses special APIs and protocols in servlets or EJBs to request services and functions of the CICS Transaction Manager.

IMS Connect

IMS Connect is the connector TCP/IP server that enables an application server client to exchange messages with IMS Open Transaction Manager Access (OTMA). This server provides communication links between TCP/IP clients and IMS databases. It supports multiple TCP/IP clients accessing multiple databases. To protect information that is transferred through TCP/IP, IMS Connect provides Secure Sockets Layer (SSL) support.

IMS Connect can also perform router functions between application server clients and local option clients with databases and IMSplex resources. Request messages received from TCP/IP clients using TCP/IP connections, or local option clients using the z/OS Program Call (PC), are passed to a database through cross-system coupling facility (XCF) sessions. IMS Connect receives response messages from the database and then passes them back to the originating TCP/IP or local option clients.

IMS Connect supports TCP/IP clients communicating with socket calls, but it can also support any TCP/IP client that communicates with a different input data stream format. User-written message exits can execute in the IMS Connect address space to convert the z/OS installation's message format to OTMA message format before IMS Connect sends the message to IMS. The user-written message exits also convert OTMA message format to the installation's message format before sending a message back to IMS Connect. IMS Connect then sends output to the client.

DB2 JDBC

The Java Database Connectivity (JDBC) is an application programming interface (API) that the Java programming language uses to access different forms of tabular data, as well as some hierarchical systems such as IMS. JDBC specifications were developed by Sun Microsystems together with relational database providers such as Oracle and IBM to ensure portability of Java applications across database platforms.

This interface does not necessarily fall into the category of "connector" because there is no separate address space required for its implementation. The interface is a Java construct that looks like a Java class but does not provide an implementation of its methods. For JDBC, the actual implementation of the JDBC interface is provided by the database vendor as a "driver." This provides portability because all access using the JDBC is through standard calls with standard parameters. Thus an application can be coded with little regard to the database being used, because all of the platform-dependent code is stored in the JDBC drivers.

As a result, JDBC must be flexible with regard to what functionality it does and does not provide, solely based on the fact that different database systems have different levels of functionality. JDBC drivers provide the physical code that implements the objects, methods, and data types defined in the specification. JDBC standards define four types of drivers, numbered 1 through 4. The distinction between them is based on how the driver is physically implemented and how it communicates with the database.

z/OS supports only Type 2 and Type 4 drivers, as follows:

- Type 2

  The JDBC API calls platform- and database-specific code to access the database. This is the most common driver type used, and offers the best performance. However, because the driver code is platform-specific, a different version has to be coded (by the database vendor) for each platform.
- Type 4

  A Type 4 driver is fully written in Java, and accesses the target database directly using the protocol of the database itself. (In the case of DB2, this is DRDA®.) Because the driver is fully written in Java, it can be ported to any platform that supports that DBMS protocol without change, thus allowing applications to also use it across platforms without change.

  A Java application, running under WebSphere Application Server, talks to the (Universal) Type 4 JDBC driver that supports two-phase commit, and the driver talks directly to the remote database server through DRDA. The Universal Type 4 driver implements DRDA Application Requester functionality.

To access DB2 on z/OS, IBM provides a Type 2 driver and a driver that combines Type 2 and Type 4 JDBC implementations. In general, JDBC Type 2 connectivity is used for Java programs that run on the same z/OS system with the target DB2 subsystem. JDBC Type 4 connectivity is used for Java programs that run on a z/OS system other than that of the target DB2 subsystem.

Chapter 5. Messaging and queuing

As enterprises move many of their applications to the Web, mainframe organizations face the complexity of enabling and managing new Web-based workloads in addition to more traditional workloads, such as batch processing. IBM WebSphere MQ facilitates application integration by passing messages between applications and Web services.

Note: This section is about WebSphere MQ. Other sections are about z/OS HTTP Server and WebSphere Application Server on z/OS. These sections use IBM products in the examples, but many such middleware products exist in the marketplace today.

What is WebSphere MQ?

Most large organizations today have an inheritance of IT systems from various manufacturers, which often makes it difficult to share communications and data across systems. Many of these organizations also need to communicate and share data electronically with suppliers and customers–who might have other disparate systems. It would be handy to have a message handling tool that could receive from one type of system and send to another type.

IBM WebSphere MQ facilitates application integration by passing messages between applications and Web services. It is used on more than 35 hardware platforms and for point-to-point messaging from Java, C, C++ and COBOL applications. Three-quarters of enterprises that buy inter-application messaging systems buy WebSphere MQ. In the largest installation, over 250 million messages a day are transmitted.

Where data held on different databases on different systems must be kept synchronized, little is available in the way of protocols to coordinate updates and deletions and so on. Mixed environments are difficult to keep aligned; complex programming is often required to integrate them.

Message queues, and the software that manages them, such as IBM WebSphere MQ for z/OS, enable program-to-program communication. In the context of online applications, **messaging** and **queuing** can be understood as follows:

- Messaging means that programs communicate by sending each other messages (data), rather than by calling each other directly.
- Queuing means that the messages are placed on queues in storage, so that programs can run independently of each other, at different speeds and times, in different locations, and without having a logical connection between them.

WebSphere MQ synchronous communication

A figure shows the basic mechanism of program-to-program communication using a synchronous communication model.

In Figure 38 on page 70, program A prepares a message and puts it on Queue 1. Program B gets the message from Queue 1 and processes it. Both Program A and

Program B use an application programming interface (API) to put messages on a queue and get messages from a queue. The WebSphere MQ API is called the Message Queue Interface (MQI).

When Program A puts a message on Queue 1, Program B might not be running. The queue stores the message safely until Program B starts and is ready to get the message. Likewise, at the time Program B gets the message from Queue 1, Program A might no longer be running. Using this model, there is no requirement for two programs communicating with each other to be executing at the same time.

There is clearly a design issue, however, about how long Program A should wait before continuing with other processing. This design might be desirable in some situations, but when the wait is too long, it is not so desirable any more. Asynchronous communication is designed to handle those situations.

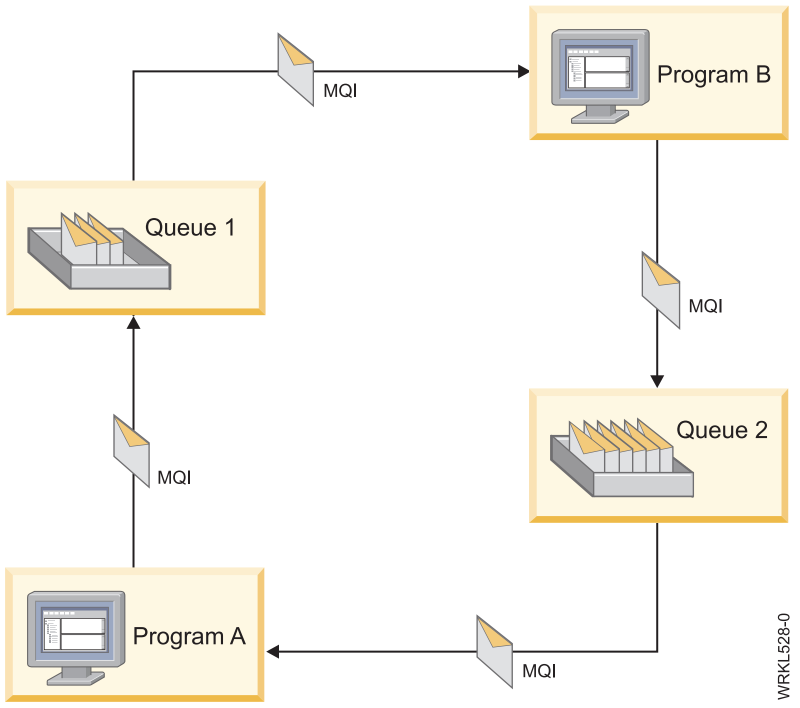


Figure 38. Synchronous application design model

WebSphere MQ asynchronous communication

A figure shows the basic mechanism of program-to-program communication using an asynchronous communication model.

Using the asynchronous model, Program A puts messages on Queue 1 for Program B to process, but it is Program C, acting asynchronously to Program A, which gets the replies from Queue 2 and processes them. Typically, Program A and Program C would be part of the same application. You can see the flow of this activity in Figure 39 on page 71.

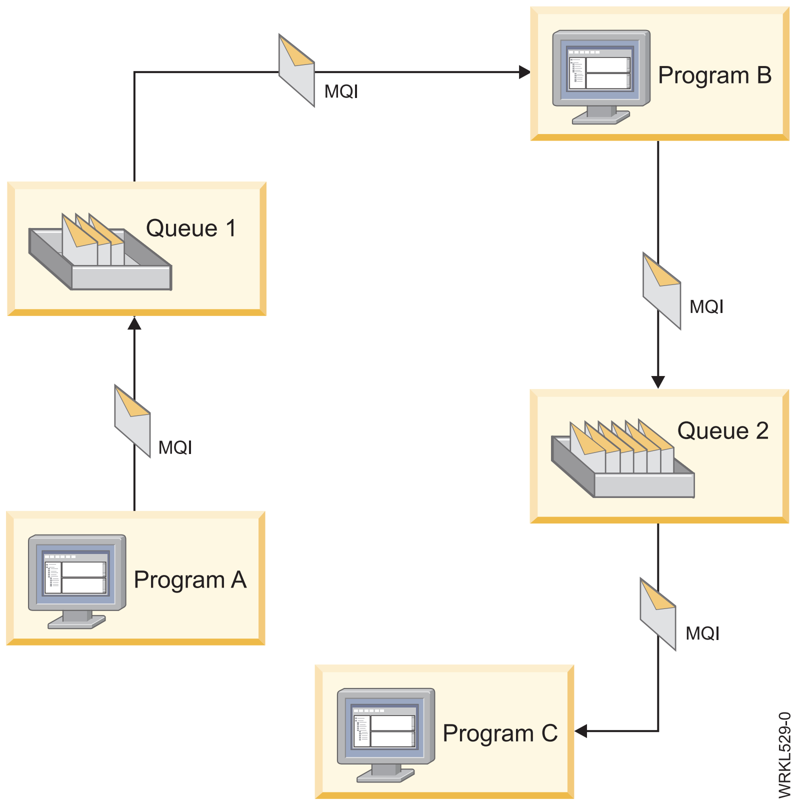


Figure 39. Asynchronous application design model

The asynchronous model is natural for WebSphere MQ. Program A can continue to put messages on Queue 1 and is not blocked by having to wait for a reply to each message. It can continue to put messages on Queue 1 even if Program B fails. If so, Queue 1 stores the messages safely until Program B is restarted.

In a variation of the asynchronous model, Program A could put a sequence of messages on Queue 1, optionally continue with some other processing, and then return to get and process the replies itself. This property of WebSphere MQ, in which communicating applications do not have to be active at the same time, is known as **time independence**.

WebSphere MQ message types

WebSphere MQ uses four types of messages.

Datagram
A message for which no response is expected.

Request
A message for which a reply is requested.

Reply A reply to a request message.

Report
A message that describes an event such as the occurrence of an error or a confirmation on arrival or delivery.

WebSphere MQ message queues and the queue manager

A message queue is used to store messages sent by programs. They are defined as objects belonging to the queue manager.

When an application puts a message on a queue, the queue manager ensures that the message is:
- Stored safely
- Is recoverable
- Is delivered once, and once only, to the receiving application.

This is true even if a message has to be delivered to a queue owned by another queue manager, and is known as the assured delivery property of WebSphere MQ.

Queue manager

The component of software that owns and manages queues is called a queue manager (QM). In WebSphere MQ, the message queue manager is called the MQM, and it provides messaging services for applications, ensures that messages are put in the correct queue, routes messages to other queue managers, and processes messages through a common programming interface called the Message Queue Interface (MQI).

The queue manager can retain messages for future processing in the event of application or system outages. Messages are retained in a queue until a successful completion response is received through the MQI.

There are similarities between queue managers and database managers. Queue managers own and control queues similar to the way that database managers own and control their data storage objects. They provide a programming interface to access data, and also provide security, authorization, recovery and administrative facilities.

There are also important differences, however. Databases are designed to provide long-time data storage with sophisticated search mechanisms, whereas queues are not designed for this. A message on a queue generally indicates that a business process is incomplete; it might represent an unsatisfied request, an unprocessed reply, or an unread report.

Types of message queues

Several types of message queues exist. In this text, the most relevant are the following:
- Local queue

  A queue is local if it is owned by the queue manager to which the application program is connected. It is used to store messages for programs that use the same queue manager. The application program doesn't have to run on the same machine as the queue manager.
- Remote queue

  A queue is remote if it is owned by a different queue manager. A remote queue is not a real queue; it is only the **definition** of a remote queue to the local queue manager. Programs cannot read messages from remote queues. Remote queues are associated with a transmission queue.
- Transmission queue

This local queue has a special purpose: it is used as an intermediate step when sending messages to queues that are owned by a different queue manager. Transmission queues are transparent to the application; that is, they are used internally by the queue manager initiation queue.

This is a local queue to which the queue manager writes (transparently to the programmer) a trigger message when certain conditions are met on another local queue, for example, when a message is put into an empty message queue or in a transmission queue. Two WebSphere MQ applications monitor initiation queues and read trigger messages, the trigger monitor and the channel initiator. The **trigger manager** can start applications, depending on the message. The **channel initiator** starts the transmission between queue managers.

- Dead-letter queue

  A queue manager (QM) must be able to handle situations when it cannot deliver a message, for example:

  – Destination queue is full.

  – Destination queue does not exist.

  – Message puts have been inhibited on the destination queue.

  – Sender is not authorized to use the destination queue.

  – Message is too large.

  – Message contains a duplicate message sequence number.

  When one of these conditions occurs, the message is written to the dead-letter queue. This queue is defined when the queue manager is created, and each QM should have one. It is used as a repository for all messages that cannot be delivered.

What is a WebSphere MQ channel?

A **channel** is a logical communication link. The conversational style of program-to-program communication requires the existence of a communications connection between each pair of communicating applications. Channels shield applications from the underlying communications protocols.

WebSphere MQ uses two kinds of channels:

- Message channel

  A message channel connects two queue managers through message channel agents (MCAs). A message channel is unidirectional, comprised of two message channel agents (a sender and a receiver) and a communication protocol. An MCA transfers messages from a transmission queue to a communication link, and from a communication link to a target queue. For bidirectional communication, it is necessary to define a **pair** of channels, consisting of a sender and a receiver.

- MQI channel

  An MQI channel connects a WebSphere MQ client to a queue manager. Clients do not have a queue manager of their own. An MQI channel is bidirectional.

How WebSphere MQ ensures transactional integrity

A business might require two or more distributed databases to be maintained in step. WebSphere MQ offers a solution involving multiple units of work acting asynchronously.

The top half of Figure 40 shows a two-phase commit structure, while the WebSphere MQ solution is shown in the lower half, as follows:

- The first application writes to a database, places a message on a queue, and issues a syncpoint to commit the changes to the two resources. The message contains data which is to be used to update a second database on a separate system. Because the queue is a remote queue, the message gets no further than the transmission queue within this unit of work. When the unit of work is committed, the message becomes available for retrieval by the sending MCA.
- In the second unit of work, the sending MCA gets the message from the transmission queue and sends it to the receiving MCA on the system with the second database, and the receiving MCA places the message on the destination queue. This is performed reliably because of the assured delivery property of WebSphere MQ. When this unit of work is committed, the message becomes available for retrieval by the second application.
- In the third unit of work, the second application gets the message from the destination queue and updates the database using the data contained in the message.

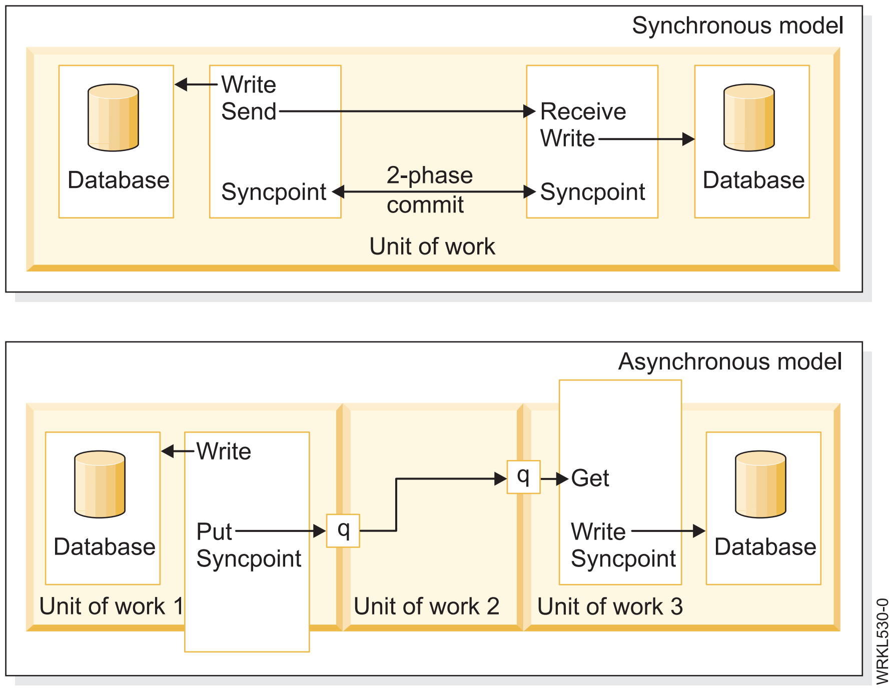


Figure 40. Data integrity

It is the transactional integrity of units of work 1 and 3, and the once and once only, assured delivery property of WebSphere MQ used in unit of work 2, which ensures the integrity of the complete business transaction.

If the business transaction is a more complex one, many units of work may be involved.

Example of messaging and queuing in WebSphere MQ

Now let’s return to the earlier example of a travel agency to see how messaging facilities play a role in booking a vacation. Assume that the travel agent must reserve a flight, a hotel room, and a rental car. All of these reservations must succeed before the overall business transaction can be considered complete.

With a message queue manager such as WebSphere MQ, the application can send several requests at once; it need not wait for a reply to one request before sending the next. A message is placed on each of three queues, serving the flight reservations application, the hotel reservations application, and the car rental application. Each application can then perform its respective task in parallel with the other two and place a reply message on the reply-to queue. The agent’s application waits for these replies and produces a consolidated answer for the travel agent.

Designing the system in this way can improve the overall response time. Although the application might normally process the replies only when they have all been received, the program logic may also specify what to do when only a partial set of replies is received within a given period of time.

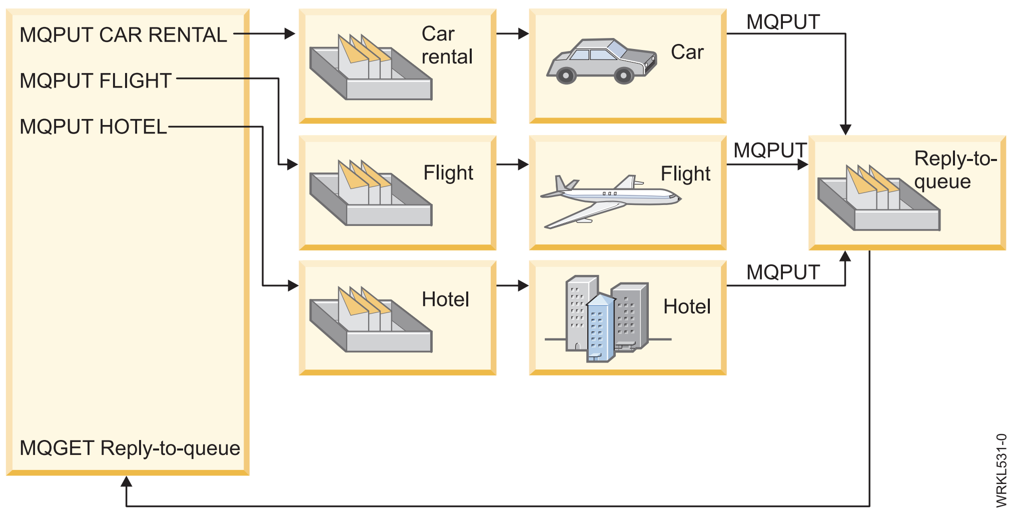


Figure 41. Parallel processing

Interfacing with CICS, IMS, batch, or TSO/E in WebSphere MQ

WebSphere MQ is available for a variety of platforms.

WebSphere MQ for z/OS includes several adapters to provide messaging and queuing support for:

- CICS: the WebSphere MQ-CICS bridge
- IMS: the WebSphere MQ-IMS bridge
- Batch or TSO/E.

Notices

Programming interface information

This publication documents information that is NOT intended to be used as Programming Interfaces of z/OS.

Trademarks

Printed in USA