IBM

Introduction to the New Mainframe: z/VSE Basics

Basic mainframe concepts and architecture

z/VSE fundamentals for students and beginners

Mainframe hardware and peripheral devices

Wolfgang Bosch
Hans Joachim Ebert
Helmut Hellner
Jerry Johnston
Wilhelm Mild
Ingolf Salm
Joerg Schmidbauer
Martin Walbruehl

Redbooks

ibm.com/redbooks

International Technical Support Organization

Introduction to the New Mainframe: z/VSE Basics

March 2007

SG24-7436-00

Note: Before using this information and the product it supports, read the information in “Notices” on page 463.

First Edition (March 2007)

This edition applies to the current version of the IBM z/VSE operating system as of March 2007.

Contents

Preface

This IBM Redbooks publication is based on the book *Introduction to the New Mainframe: z/OS Basics*, SG24-6366. It provides students of information systems technology with the background knowledge and skills necessary to begin using the basic facilities of a mainframe computer.

For optimal learning, students are assumed to have successfully completed an introductory course in computer system concepts, such as computer organization and architecture, operating systems, data management, or data communications. They should also have successfully completed courses in one or more programming languages, and be PC literate.

This textbook can also be used as a prerequisite for courses in advanced topics or for internships and special studies. It is not intended to be a complete text covering all aspects of mainframe operation, nor is it a reference book that discusses every feature and option of the mainframe facilities.

Others who will benefit from this course include experienced data processing professionals who have worked with non-mainframe platforms, or who are familiar with some aspects of the mainframe but want to become knowledgeable with other facilities and benefits of the mainframe environment.

As we go through this course, we suggest that the instructor alternate between text, lecture, discussions, and hands-on exercises. Many of the exercises are cumulative, and are designed to show the student how to design and implement the topic presented. The instructor-led discussions and hands-on exercises are an integral part of the course material, and can include topics not covered in this textbook.

In this course, we use simplified examples and focus mainly on basic system functions. Hands-on exercises are provided throughout the course to help students explore the mainframe style of computing.

At the end of this course, you will know:

- Basic concepts of the mainframe, including its usage and architecture
- Fundamentals of z/VSE, a System z entry mainframe operating system
- An understanding of mainframe workloads and the major middleware applications in use on mainframes today
- The basis for subsequent course work in more advanced, specialized areas of z/VSE, such as system administration or application programming

How this text is organized

This text is organized in four parts, as follows:

- Part 1, “Introduction to z/VSE and the mainframe environment” on page 1, provides an overview of the types of workloads commonly processed on the mainframe, such as batch jobs and online transactions. This part of the text helps students explore the user interfaces of z/VSE, a System z entry mainframe operating system. Discussion topics include VSE/ICCF and Interactive Interface, job control language, file structures, and batch processing. Special attention is paid to the users of mainframes and to the evolving role of mainframes in today’s business world.
- Part 2, “Application programming on z/VSE” on page 193, introduces the tools and utilities for developing a simple program to run on z/VSE. This part of the text guides the student through the process of application design, choosing a programming language, and using a runtime environment.
- Part 3, “Online workloads for z/VSE” on page 261, examines the major categories of interactive workloads processed by z/VSE, such as transaction processing, database management, and Web-serving. This part includes discussions of several popular middleware products, including DB2, CICS, and WebSphere Application Server.
- Part 4, “System programming on z/VSE” on page 369, provides topics to help the student become familiar with the role of the z/VSE system programmer. This part of the text includes discussions of system libraries, starting and stopping the system, security, and network communications. Also provided is an overview of mainframe hardware systems, including processors and I/O devices.

In this text, we use simplified examples and focus mainly on basic system functions. Hands-on exercises are provided throughout the text to help students explore the mainframe style of computing. Exercises include entering work into the system, checking its status, and examining the output of submitted jobs.

How each chapter is organized

Each chapter follows a common format:

- Objectives for the student
- Topics that teach a central theme related to mainframe computing
- Summary of the main ideas of the chapter
- A list of key terms introduced in the chapter

- Questions for review to help students verify their understanding of the material
- Topics for further discussion to encourage students to explore issues that extend beyond the chapter objectives
- Hands-on exercises to help students reinforce their understanding of the material

About the authors

This book based on the book *Introduction to the New Mainframe: z/OS Basics,* SG24-6366. The z/VSE text was produced by the VSE development team in Boeblingen, Germany and specialists from around the world:

Wolfgang Bosch is a Software Engineer in Germany and has worked with VSE for 17 years. His areas of experience are language products (such as COBOL, PL/I, and C) and CICS. Since joining the development team in 1998 he has specialized in Language Environment for z/VSE. He is engaged in development of Web tools for VSE and was recently involved in a multi-platform application development initiative via Rational and the new VisualAge Generator EGL Plug-in for VSE product.

Hans Joachim Ebert is a retired Senior Systems Engineer in Germany. He holds a degree in Informatics in 1966 and joined IBM in 1968, working initially in the DOS-related area. Since 1972 he has worked with CICS. His activities include technical sales and capacity planning for System z and performance analysis for VSE systems, with a focus on CICS as a main component of VSE. He has written extensively on CICS, VSE Hints & Tips, VSE performance, and CICS Web Support.

Helmut Hellner is a VSE developer in the IBM Development Laboratory, Boeblingen, Germany. He studied Computer Science at the University of Stuttgart, graduating in 1981. After several years of working with OS/390 and VM he joined the VSE development and service team. His focus is on security in z/VSE. Helmut also collected input from the different authors and integrated it in this book.

G. M. (Jerry) Johnston is a Senior Advisor to the Boeblingen lab. He has 40 years of experience in the IT field as an SE, a processor planning manager, and a software development manager. For the past few years he has been associated with VSE. Jerry holds an MSME degree from Georgia Tech and an MBA from the Wharton School of the University of Pennsylvania.

Wilhelm Mild is a z/VSE Solution Architect in the VSE Team in IBM Laboratory in Boeblingen Germany. After earning his degree in Computing Science, he dedicated more than a decade to VSAM development and data management, which is documented in Redbooks he worked on. He designs connectors for VSE and solutions for heterogeneous environments with VSE. Wilhelm teaches workshops and speaks at many international conferences and customer events.

Ingolf Salm is the z/VSE Lead Architect. Since 1981, he has been a member of the VSE development team located in Boeblingen, Germany. His favorite project was the implementation of the *turbo* z/VSE dispatcher. Besides z/VSE, he also works on Linux™ projects.

Joerg Schmidbauer is a VSE developer in the IBM Development Laboratory, Boeblingen, Germany. He has worked for IBM since 1989 in VSE Development. Besides VSE Connector-related work, he works on VSE hardware cryptographic support. He speaks at many customer events.

Martin Walbruehl is a VSE developer in the IBM Development Laboratory, Boeblingen, Germany. He studied Mathematics and Computer Science at the Rheinische-Friedrich-Wilhelm-University of Bonn, graduating in 1989. After several years of working in VSE system test, he joined VSE development for VSE/POWER in 1995. Today he works in VSE service and in VSE/POWER development.

This book based on the book *Introduction to the New Mainframe: z/OS Basics,* SG24-6366, which was produced by the International Technical Support Organization, Poughkeepsie Center:

Mike Ebbers has worked with mainframe systems at IBM for 32 years. For part of that time, he taught hands-on mainframe classes to new hires just out of college. Mike currently creates IBM Redbooks, a popular set of product documentation that can be found at:

http://www.ibm.com/redbooks

Wayne O'Brien is an Advisory Software Engineer at IBM Poughkeepsie. Since joining IBM in 1988, he has developed user assistance manuals and online help for a wide variety of software products. Wayne holds a Master of Science degree in Technical Communications from Rensselaer Polytechnic Institute (RPI) of Troy, New York.

Bill Ogden is a retired IBM Senior Technical Staff Member. He holds BSEE and MS (Computer Science) degrees and has worked with mainframes since 1962 and with z/OS since it was known as OS/360 Release 1/2. Since joining the ITSO in 1978, Bill has specialized in encouraging users new to the operating system and associated hardware.

Acknowledgements

The following people are gratefully acknowledged for their contribution to this project:

Manuela Buettcher, VSE Development, Boeblingen, Germany
Mark Evans, VisualAge Generator and EGL Development, Raleigh, USA
Ingo Franzki, VSE Development, Boeblingen, Germany
Heinz Hagedorn, VSE Development, Boeblingen, Germany
Helmut Hanselmann, VSE Development, Boeblingen, Germany
Garry Hasler, LE for z/VSE Development, Perth, Australia
Helga Hoff, VSE Development, Boeblingen, Germany
Martin Kahnt, VSE Development, Boeblingen, Germany
Antoinette Kaschner, VSE Development, Boeblingen, Germany
Dagmar Kruse, Technical Sales z/VSE, Munich, Germany
Torsten Roeber, SW Support DB2, Mainz, Germany
Elke Schaefer, VSE Development, Boeblingen, Germany
Hans-Ulrich Schmidt, VSE Development, Boeblingen, Germany
Gerhard Schneidt, VSE Development, Boeblingen, Germany
Heiko Schnell, VSE Development, Boeblingen, Germany
Heinz Schulz, VSE Development, Boeblingen, Germany
Richard Steimle, VSE Development, Boeblingen, Germany
Ed Wilhelm, VSE Development, Boeblingen, Germany

Special thanks to the editors at the ITSO center in Poughkeepsie, New York.

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review form found at:

  `ibm.com/redbooks`

- Send your comments in an e-mail to:

  `redbook@us.ibm.com`

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Part 1

Introduction to z/VSE and the mainframe environment

Welcome to mainframe computing. We begin this text with an overview of the mainframe computer and its place in today's information technology (IT) organization. We explore the reasons why public and private enterprises throughout the world rely on the mainframe as the foundation of large-scale computing. We discuss the types of workloads that are commonly associated with the mainframe, such as batch jobs and online or interactive transactions, and the unique manner in which this work is processed by the mainframe operating system—z/VSE.

Throughout this text, we pay special attention to the people who use mainframes and to the role of the new mainframe in today's business world.

1

Introduction to the new mainframe

Objective: As a technical professional in the world of mainframe computing, you will need to understand how mainframe computers support your company's IT infrastructure and business goals. You will also need to know the job titles of the various members of your company's mainframe support team.

After completing this chapter, you will be able to:

- List ways in which the mainframe of today challenges the traditional thinking about centralized computing versus distributed computing.
- Explain how businesses make use of mainframe processing power, the typical uses of mainframes, and how mainframe computing differs from other types of computing.
- Outline the major types of workloads for which mainframes are best suited.
- Name five jobs or responsibilities that are related to mainframe computing.
- Identify four mainframe operating systems.

1.1 The new mainframe

e-business
The transaction of business over an electronic medium such as the Internet.

Today, mainframe computers play a central role in the daily operations of most of the world's largest corporations, including many Fortune 1000 companies. While other forms of computing are used extensively in various business capacities, the mainframe occupies a coveted place in today's e-business environment. In banking, finance, health care, insurance, public utilities, government, and a multitude of other public and private enterprises, the mainframe computer continues to form the foundation of modern business.

The long-term success of mainframe computers is without precedent in the information technology (IT) field. Periodic upheavals shake world economies and continuous—often wrenching—change in the Information Age has claimed many once-compelling innovations as victims in the relentless march of progress. As emerging technologies leap into the public eye, many are just as suddenly rendered obsolete by some even newer advancement. Yet today, as in every decade since the 1960s, mainframe computers and the mainframe *style* of computing dominate the landscape of large-scale business computing.

Why has this one form of computing taken hold so strongly among the world's largest corporations? In this chapter we look at the reasons why mainframe computers continue to be the popular choice for large-scale business computing.

1.2 The S/360: A turning point in mainframe history

When did mainframe computers come into being? The origin of mainframe computers dates back to the 1950s, if not earlier. In those days, mainframe computers were not just the largest computers—they were the *only* computers, and few businesses could afford them.

System/360
The first general purpose computer, introduced in 1964.

Mainframe development occurred in a series of *generations* starting in the 1950s. First generation systems, such as the IBM 705 in 1954 and the IBM 1401 in 1959, were a far distant from the enormously powerful machines that were to follow, but they clearly had characteristics of mainframe computers. These computers were sold as business machines and served then—as they do now—as the central data repository in a corporation's data processing center. [1]

In the 1960s, the course of computing history changed dramatically when mainframe manufacturers began to standardize the hardware and software they offered to customers. The introduction of the IBM System/360 (or S/360) in 1964

[1] According to the IBM product brochure, typical customer uses for a 1401 were "payroll, railroad freight car accounting, public utility customer accounting, inventory control, and accounts receivable."

signaled the start of the third generation: the first general-purpose computers. Earlier systems such as the 1401 were dedicated as either commercial or scientific computers. The revolutionary S/360 could perform both types of computing, as long as the customer, a software company, or a consultant provided the programs to do so. In fact, the name S/360 refers to the architecture's wide scope: 360 degrees to cover the entire circle of possible uses.

With standardized mainframe computers to run their workloads, customers could, in turn, write business applications that did not need specialized hardware or software. Moreover, customers were free to upgrade to newer and more powerful processors without concern for compatibility problems with their existing applications. The first wave of customer business applications were mostly written in Assembler, COBOL, FORTRAN, or PL/I (programming language 1), and a substantial number of these older programs are still in use today.

In the decades since the 1960s, mainframe computers have steadily grown to achieve enormous processing capabilities. The new mainframe has an unrivaled ability to serve concurrent users by the tens of thousands, manage a lot of data, and reconfigure hardware and software resources to accommodate changes in workload—all from a single point of control.

1.3 An evolving architecture

An *architecture* is a set of defined terms and rules that are used as instructions to build products. In computer science, an architecture describes the organizational structure of a system. An architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts. Parts that interact through interfaces include classes, components, and subsystems.

Architecture describes the organizational structure of a system.

Starting with the first large machines, which arrived on the scene in the 1960s and became known as *Big Iron* (in contrast to smaller departmental systems), each new generation of mainframe computers has included improvements in one or more of the following areas of the architecture:[2]

- More and faster processors
- More physical memory and greater memory addressing capability
- Dynamic capabilities for upgrading both hardware and software

[2] Since the introduction of the S/360 in 1964, IBM has significantly extended the platform roughly every ten years: System/370 in 1970, System/370 Extended Architecture (370-XA) in 1983, Enterprise Systems Architecture/390 (ESA/390) in 1990, and z/Architecture in 2000. For more information about earlier mainframe hardware systems, see Appendix A, “A brief look at IBM mainframe history” on page 581.

- Increased automation of hardware error checking and recovery
- Enhanced devices for input/output (I/O) and more and faster paths (channels) between I/O devices and processors
- More sophisticated I/O attachments, such as network adapters with extensive inboard processing
- A greater ability to divide the resources of one machine into multiple, logically independent and isolated systems, each running its own operating system using the virtualization method of computing

Despite the continual change, mainframe computers remain the most stable, secure, and compatible of all computing platforms. The latest models can handle the most advanced and demanding customer workloads, yet continue to run applications that were written in the 1970s or earlier.

How can a technology change so much, yet remain so stable? It can by evolving to meet new challenges. In the early 1990s, the client/server model of computing, with its distributed nodes of less powerful computers, emerged to challenge the dominance of mainframe computers. Industry pundits predicted a swift end for the mainframe computer and called it a *dinosaur*. In response, mainframe designers did what they have always done when confronted with changing times and a growing list of user requirements: they designed new mainframe computers to meet the demand. IBM, as the leading manufacturer of mainframe computers, code-named its then-current machine T-Rex.

With the expanded functions and added tiers of data processing capabilities such as Web-serving, autonomics, disaster recovery, and grid computing, the mainframe computer is poised to ride the next wave of growth in the IT industry. Mainframe manufacturers such as IBM are once again reporting annual sales growth in the double digits.

"I predict that the last mainframe will be unplugged on March 15, 1996."
—Stewart Alsop, Infoworld, March 1991

And the evolution continues. While the mainframe computer has retained its traditional, central role in the IT organization, that role is now defined to include being the primary hub in the largest distributed networks. In fact, the Internet itself is based largely on numerous, interconnected mainframe computers serving as major hubs and routers.

As the image of the mainframe computer continues to evolve, you might ask: Is the mainframe computer a self-contained computing environment, or is it one part of the puzzle in distributed computing? The answer is that the new mainframe is both: a self-contained processing center, powerful enough to process the largest and most diverse workloads in one secure *footprint*, and one that is just as effective when implemented as the primary server in a corporation's distributed server farm. In effect, the mainframe computer is the definitive server in the client/server model of computing.

1.4 Mainframes in our midst

Despite the predominance of mainframes in the business world, these machines are largely invisible to the general public, the academic community, and indeed many experienced IT professionals. Instead, other forms of computing attract more attention, at least in terms of visibility and public awareness. That this is so is perhaps not surprising. After all, who among us needs direct access to a mainframe? And, if we did, where would we find one to access? The truth, however, is that we are *all* mainframe users, whether we realize it or not (more on this later).

Most of us with some personal computer (PC) literacy and sufficient funds can purchase a notebook computer and quickly put it to good use—running software, browsing Web sites, and perhaps even writing papers for college professors to grade. With somewhat greater effort and technical prowess, we can delve more deeply into the various facilities of a typical Intel®-based workstation and learn its capabilities through direct, hands-on experience—with or without help from any of a multitude of readily available information sources in print or on the Web.

Mainframes, however, tend to be hidden from the public eye. They do their jobs dependably—indeed, with almost total reliability—and are highly resistant to most forms of insidious abuse that afflict PCs, such as e-mail-borne viruses and *trojan horses*. By performing stably, quietly, and with negligible downtime, mainframes are the example by which all other computers are judged. But at the same time, this lack of attention tends to allow them to fade into the background.

Furthermore, in a typical customer installation, the mainframe shares space with many other hardware devices: external storage devices, hardware network routers, channel controllers, and automated tape library *robots*, to name a few. The new mainframe is physically no larger than many of these devices and generally does not stand out from the crowd of peripheral devices.

So, how can we explore the mainframe's capabilities in the real world? How can we learn to interact with the mainframe, learn its capabilities, and understand its importance to the business world? Major corporations are eager to hire new mainframe professionals, but there is a catch: Some previous experience would help.

Would we even know a mainframe if we saw one, given that these machines have evolved to flourish in the twenty-first century IT organization? What we need is an experienced guide to lead us on a *dinosaur safari*, which is where this textbook comes in.

1.5 What is a mainframe?

First, let us tackle the terminology. Today, computer manufacturers do not always use the term *mainframe* to refer to mainframe computers. Instead, most have taken to calling any commercial-use computer—large or small—a *server*, with the mainframe simply being the largest type of server in use today. IBM, for example, refers to its latest mainframe as the IBM System z9 server. We use the term mainframe in this text to mean computers that can support thousands of applications and input/output devices to simultaneously serve thousands of users.

Server farm
A very large collection of servers.

Servers are proliferating. A business might have a large server collection that includes transaction servers, database servers, e-mail servers, and Web servers. Very large collections of servers are sometimes called *server farms* (in fact, some data centers cover areas measured in *acres*). The hardware required to perform a server function can range from little more than a cluster of rack-mounted personal computers to the most powerful mainframes manufactured today.

A mainframe is the central data repository, or *hub,* in a corporation's data processing center, linked to users through less powerful devices such as workstations or terminals. The presence of a mainframe often implies a centralized form of computing, as opposed to a distributed form of computing. Centralizing the data in a single mainframe repository saves customers from having to manage updates to more than one copy of their business data, which increases the likelihood that the data is current.

The distinction between centralized and distributed computing, however, is rapidly blurring as smaller machines continue to gain in processing power and mainframes become ever more flexible and multi-purpose. Market pressures require that today's businesses continually reevaluate their IT strategies to find better ways of supporting a changing marketplace. As a result, mainframes are now frequently used in combination with networks of smaller servers in a multitude of configurations. The ability to dynamically reconfigure a mainframe's hardware and software resources (such as processors, memory, and device connections), while applications continue running, further underscores the flexible, evolving nature of the modern mainframe.

Platform
A computer architecture (hardware and software).

While mainframe hardware has become harder to pigeon-hole, so, too, have the operating systems that run on mainframes. Years ago, in fact, the terms defined each other: a mainframe was any hardware system that ran a major IBM operating system.[3] This meaning has been blurred in recent years because these operating systems can be run on very small systems.

[3] The name was also traditionally applied to large computer systems that were produced by other vendors.

Computer manufacturers and IT professionals often use the term *platform* to refer to the hardware and software that are associated with a particular computer architecture. For example, a mainframe computer and its operating system (and their predecessors[4]) are considered a platform. UNIX® on a Reduced Instruction Set Computer (RISC) system is considered a platform somewhat independently of exactly which RISC machine is involved. Personal computers can be seen as several different platforms, depending on which operating system is being used.

So, let us return to our question now: What is a mainframe? Today, the term *mainframe* can best be used to describe a *style* of operation, applications, and operating system facilities. To start with a working definition, "a mainframe is what businesses use to host the commercial databases, transaction servers, and applications that require a greater degree of security and availability than is commonly found on smaller-scale machines."

Early mainframe systems were housed in enormous, room-sized metal boxes or frames, which is probably how the term mainframe originated. The early mainframe required large amounts of electrical power and air-conditioning, and the room was filled mainly with I/O devices. Also, a typical customer site had several mainframes installed, with most of the I/O devices connected to all of the mainframes. During their largest period, in terms of physical size, a typical mainframe occupied 2,000 to 10,000 square feet (600 to 3000 square meters). Some installations were even larger than this.

Mainframe
A large computer system that is used to host the databases, transaction servers, and applications that require a great degree of security and availability.

Starting around 1990, mainframe processors and most of their I/O devices became physically smaller, while their functionality and capacity continued to grow. Mainframe systems today are much smaller than earlier systems—about the size of a large refrigerator.

In some cases, it is now possible to run a mainframe operating system on a PC that emulates a mainframe. Such emulators are useful for developing and testing business applications before moving them to a mainframe production system.

[4] IBM System/390 (S/390) refers to a specific series of machines, which have been superseded by the IBM System z machines. Nevertheless, many S/390 systems are still in use. Therefore, keep in mind that although we discuss the System z systems in this course, almost everything discussed also applies to S/390 machines. One major exception is 64-bit addressing, which is used only with System z.

Clearly, the term mainframe has expanded beyond merely describing the physical characteristics of a system. Instead, the word typically applies to some combination of the following attributes:

- Compatibility with mainframe operating systems, applications, and data
- Centralized control of resources
- Hardware and operating systems that can share access to disk drives with other systems, with automatic locking and protection against destructive simultaneous use of disk data
- A *style* of operation, often involving dedicated operations staff who use detailed *operations procedure books* and highly organized procedures for backups, recovery, training, and disaster recovery at an alternative location
- Hardware and operating systems that routinely work with hundreds or thousands of simultaneous I/O operations

As the performance and cost of hardware resources such as central processing unit (CPU) power and external storage media improve, and the number and types of devices that can be attached to the CPU increase, the operating system software can more fully take advantage of the improved hardware. Also, continuing improvements in software functionality help drive the development of each new generation of hardware systems.

1.6 Who uses mainframe computers?

So, who uses mainframes? Just about *everyone* has used a mainframe computer at one point or another. If you ever used an automated teller machine (ATM) to interact with your bank account, you used a mainframe.

Today, mainframe computers play a central role in the daily operations of most of the world's largest corporations. While other forms of computing are used extensively in business in various capacities, the mainframe occupies a coveted place in today's e-business environment. In banking, finance, health care, insurance, utilities, government, and a multitude of other public and private enterprises, the mainframe computer continues to be the foundation of modern business.

Until the mid-1990s, mainframes provided the *only* acceptable means of handling the data processing requirements of a large business. These requirements were then (and are often now) based on large and complex batch jobs, such as payroll and general ledger processing.

The mainframe owes much of its popularity and longevity to its inherent reliability and stability, a result of careful and steady technological advances that have been made since the introduction of the System/360 in 1964. No other computer architecture can claim as much continuous, evolutionary improvement, while maintaining compatibility with previous releases.

Because of these design strengths, the mainframe is often used by IT organizations to host the most important, *mission-critical* applications. These applications typically include customer order processing, financial transactions, production and inventory control, payroll, as well as many other types of work.

One common impression of a mainframe's user interface is the 80x24-character *green screen* terminal, named for the old cathode ray tube (CRT) monitors from years ago that glowed green. In reality, mainframe interfaces today look much the same as those for personal computers or UNIX systems. When a business application is accessed through a Web browser, there is often a mainframe computer performing crucial functions *behind the scenes*.

Many of today's busiest Web sites store their production databases on a mainframe host. New mainframe hardware and software products are ideal for Web transactions because they are designed to allow huge numbers of users and applications to rapidly and simultaneously access the same data without interfering with each other. This security, scalability, and reliability is critical to the efficient and secure operation of contemporary information processing.

Corporations use mainframes for applications that depend on scalability and reliability. For example, a banking institution could use a mainframe to host the database of its customer accounts, for which transactions can be submitted from any of thousands of ATM locations worldwide.

Businesses today rely on the mainframe to:

- Perform large-scale transaction processing (thousands of transactions per second)[5].
- Support thousands of users and application programs concurrently accessing numerous resources.
- Handle large-bandwidth communication.

The roads of the information superhighway often lead to a mainframe.

[5] The latest series of IBM mainframe computers, the IBM System z9 109 (also known as the z9-109), can process a staggering *one billion* transactions per day.

1.7 Factors contributing to mainframe use

The reasons for mainframe use are many, but most generally fall into one or more of the following categories:

- Reliability, availability, and serviceability
- Security
- Scalability
- Continuing compatibility
- Evolving architecture

Let us look at each of these categories in more detail.

1.7.1 Reliability, availability, and serviceability

The *reliability*, *availability*, and *serviceability* (or RAS) of a computer system have always been important factors in data processing. When we say that a particular computer system *exhibits RAS characteristics*, we mean that its design places a high priority on the system remaining in service at all times. Ideally, RAS is a central design feature of all aspects of a computer system, including the applications.

RAS has become accepted as a collective term for many characteristics of hardware and software that are prized by mainframe users. The terms are defined as follows:

Reliability	The system's hardware components have extensive self-checking and self-recovery capabilities. The system's software reliability is a result of extensive testing and the ability to make quick updates for detected problems.
Availability	The system can recover from a failed component without impacting the rest of the running system. This applies to hardware recovery (the automatic replacing of failed elements with spares) and software recovery (the layers of error recovery that are provided by the operating system).
Serviceability	The system can determine why a failure occurred. Allows for the replacement of hardware and software elements while impacting as little of the operational system as possible. This term also implies well-defined units of replacement, either hardware or software.

Availability
The ability to recover from the failure of a component without impacting the rest of the running system.

A computer system is available when its applications are available. An available system is one that is reliable (that is, it rarely requires downtime for upgrades or repairs). And, if the system is brought down by an error condition, it must be serviceable (that is, easy to fix within a relatively short period of time).

Mean time between failure (MTBF) refers to the availability of a computer system. The new mainframe and its associated software have evolved to the point that customers often experience months or even *years* of system availability between system downtimes. Moreover, when the system is unavailable because of an unplanned failure or a scheduled upgrade, this period is typically very short. The remarkable availability of the system in processing the organization's mission-critical applications is vital in today's 24-hour, global economy. Along with the hardware, mainframe operating systems exhibit RAS through such features as storage protection and a controlled maintenance process.

Beyond RAS, a state-of-the-art mainframe system might be said to provide *high availability* and *fault tolerance*. Redundant hardware components in critical paths, enhanced storage protection, a controlled maintenance process, and system software designed for unlimited availability all help to ensure a consistent, highly available environment for business applications in the event that a system component fails. Such an approach allows the system designer to minimize the risk of having a *single point of failure* undermine the overall RAS of a computer system.

1.7.2 Security

One of a firm's most valuable resources is its data: customer lists, accounting data, employee information, and so on. This critical data needs to be securely managed and controlled, and, simultaneously, made available to those users authorized to see it. The mainframe computer has extensive capabilities to simultaneously share, but still protect, the firm's data among multiple users.

In an IT environment, data security is defined as protection against unauthorized access, transfer, modification, or destruction, whether accidental or intentional. To protect data and to maintain the resources necessary to meet the security objectives, customers typically add a sophisticated security manager product to their mainframe operating system. The customer's security administrator often bears the overall responsibility for using the available technology to transform the company's security policy into a usable plan.

A secure computer system prevents users from accessing or changing any objects on the system, including user data, except through system-provided interfaces that enforce authority rules. The new mainframe can provide a very secure system for processing large numbers of parallel transaction and applications that access critical data. We discuss the mainframe security in Chapter 17, "Security on z/VSE" on page 483.

1.7.3 Scalability

It has been said that the only constant is *change.* Nowhere is that statement truer than in the IT industry. In business, positive results can often trigger a growth in IT infrastructure to cope with increased demand. The degree to which the IT organization can add capacity without disruption to normal business processes or without incurring excessive overhead (nonproductive processing) is largely determined by the *scalability* of the particular computing platform.

Scalability
The ability of a system to retain performance levels when adding processors, memory, and storage.

By scalability, we mean the ability of the hardware, software, or a distributed system to continue to function well as it is changed in size or volume (for example, the ability to retain performance levels when adding processors, memory, and storage). A scalable system can efficiently adapt to work with larger or smaller networks performing tasks of varying complexity.

As a company grows in employees, customers, and business partners, it usually needs to add computing resources to support business growth. One approach is to add more processors of the same size, with the resulting overhead in managing this more complex setup. Alternatively, a company can consolidate its many smaller processors into fewer, larger systems. Using a mainframe system, many companies have significantly lowered their total cost of ownership (TCO), which includes not only the cost of the machine (its hardware and software), but the cost to run it.

Mainframes exhibit scalability characteristics in both hardware and software, with the ability to run multiple copies of the operating system software on a single hardware box. The scalability of the mainframe allows allocating as much processor power as needed to individual operating systems. This process can be done on demand by using the most efficient virtualization layer for mainframes, which is z/VM (see also 1.10.2, “z/VM” on page 29).

1.7.4 Continuing compatibility

Compatibility
The ability of a system to run software requiring new hardware instructions and to run older software requiring the original hardware instructions.

Mainframe customers tend to have a very large financial investment in their applications and data. Some applications have been developed and refined over decades. Some applications were written many years ago, while others may have been written very recently. The ability of an application to work in the system or its ability to work with other devices or programs is called *compatibility*.

The need to support applications of varying ages imposes a strict compatibility demand on mainframe hardware and software, which have been upgraded many times since the first System/360 mainframe computer was shipped in 1964. Applications *must* continue to work properly. Thus, much of the design work for new hardware and system software revolves around this compatibility requirement.

The overriding need for compatibility is also the primary reason why many aspects of the system work as they do, for example, the syntax restrictions of the job control language (JCL) that is used to control batch jobs. Any new design enhancements made to JCL must preserve compatibility with older jobs so that they can continue to run without modification. The desire and need for continuing compatibility is one of the defining characteristics of mainframe computing.

Absolute compatibility across decades of changes and enhancements is not possible, of course, but the designers of mainframe hardware and software make it a top priority. When an incompatibility is unavoidable, the designers typically warn users *at least a year* in advance that software changes might be needed.

1.8 Typical mainframe workloads

Most mainframe workloads fall into one of two categories: batch processing or online transaction processing, which includes Web-based applications (Figure 1-1).

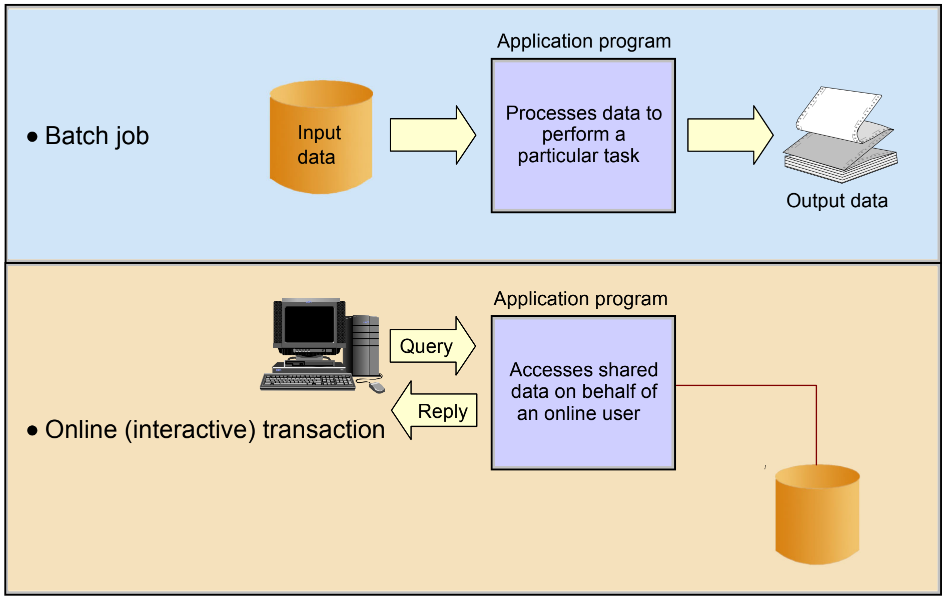

Batch job
Input data
Application program
Processes data to perform a particular task
Output data
Online (interactive) transaction
Query
Reply
Application program
Accesses shared data on behalf of an online user

Figure 1-1   Typical mainframe workloads

These workloads are discussed in several chapters in this text. The following sections provide an overview.

1.8.1 Batch processing

One key advantage of mainframe systems is their ability to process a high amount of data from high-speed storage devices and produce valuable output. For example, mainframe systems make it possible for banks and other financial institutions to perform end-of-quarter processing and produce reports that are necessary to customers (for example, quarterly stock statements or pension statements) or to the government (for example, financial results). With mainframe systems, retail stores can generate and consolidate nightly sales reports for review by regional sales managers.

Batch processing The running of jobs on the mainframe without user interaction.

The applications that produce these statements are *batch* applications. That is, they are processed on the mainframe without user interaction. A *batch job* is submitted on the computer, reads and processes data in bulk (perhaps terabytes of data), and produces output, such as customer billing statements. An equivalent concept can be found in a UNIX script file or a Windows® command file, but a z/VSE batch workload might process multiple parallel jobs with millions of records.

While batch processing is possible on distributed systems, it is not as commonplace as it is on mainframes because distributed systems often lack:

- Sufficient data storage
- Available processor capacity, or *cycles*
- Ability of mass parallel input/output (I/O) workload

Mainframe operating systems are typically equipped with sophisticated job scheduling software that allows data center staff to submit, manage, and track the execution and output of batch jobs[6].

Batch processes typically have the following characteristics:

- Large amounts of input data are processed and stored (perhaps terabytes or more), large numbers of records are accessed, and a large volume of output is produced.
- Immediate response time is usually not a requirement. However, batch jobs often must complete within a *batch window* (a period of less-intensive online activity), as prescribed by a *service level agreement* (SLA).
- Information is generated about large numbers of users or data entities (for example, customer orders or a retailer’s stock on hand).

[6] In the early days of the mainframe, punched cards were often used to enter jobs into the system for execution. *Keypunch operators* used card punches to enter data, and decks of cards (or batches) were produced. These were fed into card readers, which read the jobs and data into the system. As you can imagine, this process was cumbersome and error-prone. Nowadays, it is possible to transfer the equivalent of punched card data to the mainframe in a PC text file. We discuss various ways of introducing work into the mainframe in Chapter 7, “Batch processing and VSE/POWER” on page 177.

- A scheduled batch process can consist of the execution of hundreds or thousands of jobs in a pre-established sequence.

During batch processing, multiple types of work can be generated. Consolidated information such as profitability of investment funds, scheduled database backups, processing of daily orders, and updating of inventories are common examples. Figure 1-2 on page 18 shows a number of batch jobs running in a typical mainframe environment.

As shown in Figure 1-2 on page 18, consider the following elements at work in the scheduled batch process:

1. At night, numerous batch jobs running programs and utilities are processed. These jobs consolidate the results of the online transactions that take place during the day.
2. The batch jobs generate reports of business statistics.
3. Backups of critical files and databases are made before and after the batch window.
4. Reports with business statistics are sent to a specific area for analysis the next day.
5. Reports with exceptions are sent to the branch offices.
6. Monthly account balance reports are generated and sent to all bank customers.
7. Reports with processing summaries are sent to the partner credit card company.

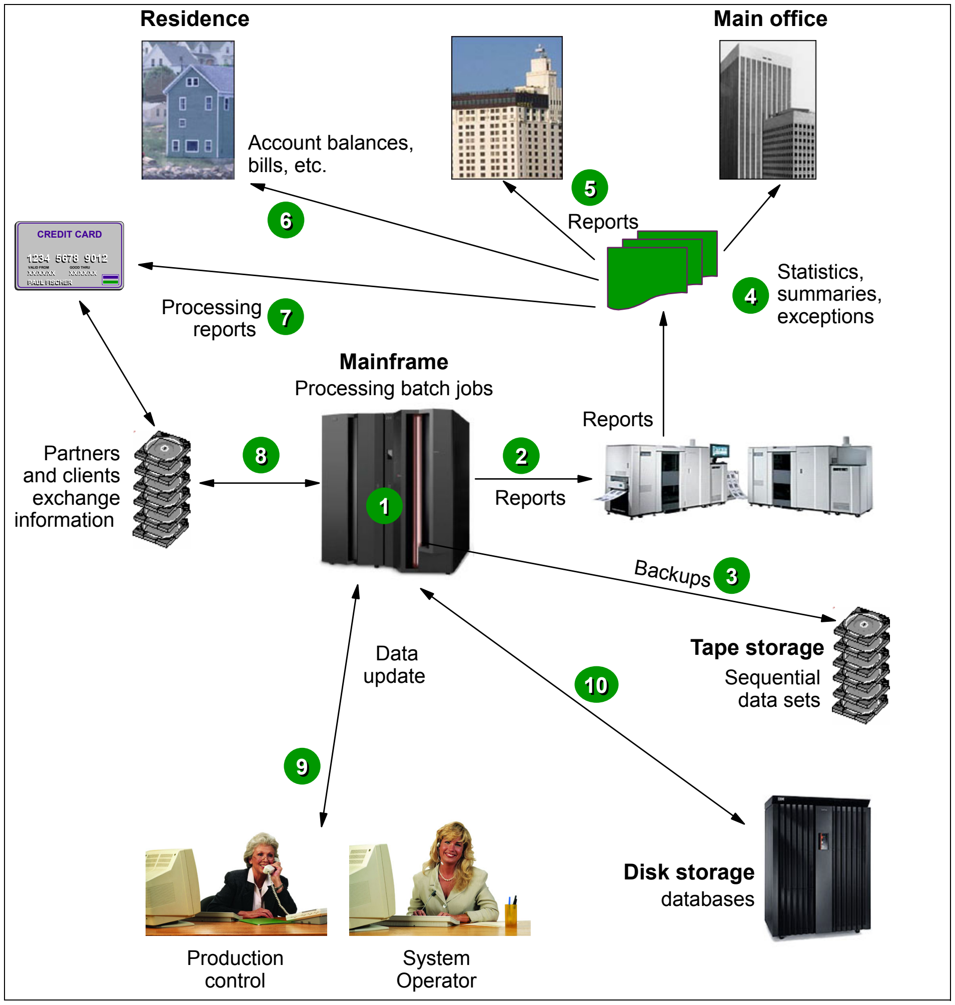


Figure 1-2 Typical batch use

8. A credit card transaction report is received from the partner company.
9. In the production control department, the operations area is monitoring the messages on the system console and the execution of the jobs.
10. Jobs and transactions are reading or updating the database (the same one that is used by online transactions), and many files are written to tape.

1.8.2 Online transaction processing

Transaction processing that occurs interactively with the end user is referred to as *online transaction processing,* or OLTP. Typically, mainframes serve a vast number of *transaction systems*. These systems are often mission-critical applications that businesses depend on for their core functions. Transaction systems must be able to support an unpredictable number of concurrent users and transaction types. Most transactions are executed in short time periods—fractions of a second in some cases.

One of the main characteristics of a transaction system is that the interactions between the user and the system are very short. The user will perform a complete business transaction through short interactions, with immediate response time required for each interaction. These systems are currently supporting mission-critical applications. Therefore, continuous availability, high performance, and data protection and integrity are required.

Online transactions are familiar to most people. Examples include:

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

Online transaction processing (OLTP) Transaction processing that occurs interactively with the end user.

For example, inside a bank branch office or on the Internet, customers use online services when checking an account balance or directing fund balances.

In fact, an online system performs many of the same functions as 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

Some industry uses of mainframe-based online systems include:

- Banks - ATMs, teller systems for customer service
- Insurance - agent systems for policy management and claims processing
- Travel and transport - airline reservation systems
- Manufacturing - inventory control, production scheduling
- Government - tax processing, license issuance and management

How might the end users in these industries interact with their mainframe systems? Multiple factors can influence the design of a company's transaction processing system, including:

- Number of users interacting with the system at any one time.
- Number of *transactions per second* (TPS).
- Availability requirements of the application. For example, must the application be available 24 hours a day, seven days a week, or can it be brought down briefly one night each week?

Before personal computers and intelligent workstations became popular, the most common way to communicate with online mainframe applications was with 3270 terminals. These devices were sometimes known as *dumb* terminals, but they had enough intelligence to collect and display a full screen of data rather than interacting with the computer for each keystroke, saving processor cycles. The characters were green on a black screen, so the mainframe applications were nicknamed *green screen* applications.

Based on these factors, user interactions vary from installation to installation. With applications now being designed, many installations are reworking their existing mainframe applications to include Web browser-based interfaces for users. This work sometimes requires new application development, but can often be done with vendor software purchased to *re-face* the application. Here, the end user often does not realize that there is a mainframe behind the scenes.

In this text, there is no need to describe the process of interacting with the mainframe through a Web browser, as it is exactly the same as any interaction a user would have through the Web. The only difference is the machine at the other end.

Online transactions usually have the following characteristics:

- A small amount of input data, a few stored records accessed and processed, and a small amount of data as output
- Immediate response time, usually less than one second
- Large numbers of users involved in large numbers of transactions simultaneously
- Round-the-clock availability of the transactional interface to the user
- Assurance of security for transactions and user data

In a bank branch office, for example, customers use online services when checking an account balance or making an investment.

Figure 1-3 shows a series of common online transactions using a mainframe.

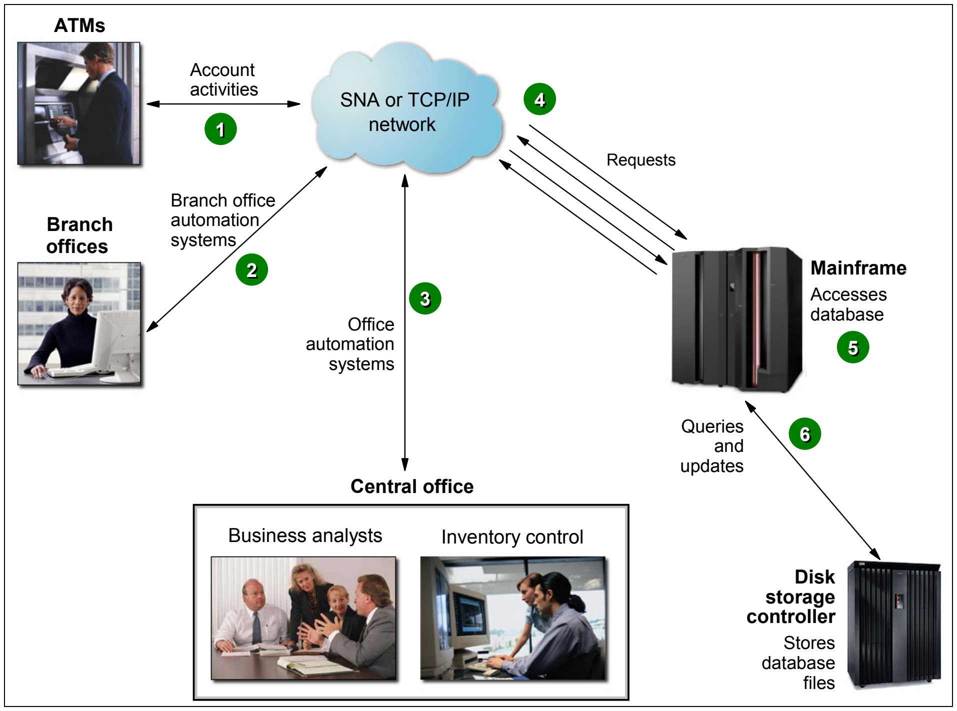


Figure 1-3 Typical online use

Figure 1-3 shows:

1. A customer uses an ATM, which presents a user-friendly interface for various functions (withdrawal, query account balance, deposit, transfer, or cash advance from a credit card account).
2. Elsewhere in the same private network, a bank employee in a branch office performs operations such as consulting, fund applications, and money ordering.
3. At the bank's central office, business analysts tune transactions for improved performance. Other staff use specialized online systems for office automation to perform customer relationship management, budget planning, and stock control.
4. All requests are directed to the mainframe computer for processing.

5. Programs running on the mainframe computer perform updates and inquires to the database management system (for example, DB2).
6. Specialized disk storage systems store the database files.

1.9 Roles in the mainframe world

Mainframe systems are designed to be used by large numbers of people. Most of those who interact with mainframes are end users (people who use the applications that are hosted on the system). However, because of the large number of end users, applications running on the system, and the sophistication and complexity of the system software that supports the users and applications, a variety of roles are needed to operate and support the system.

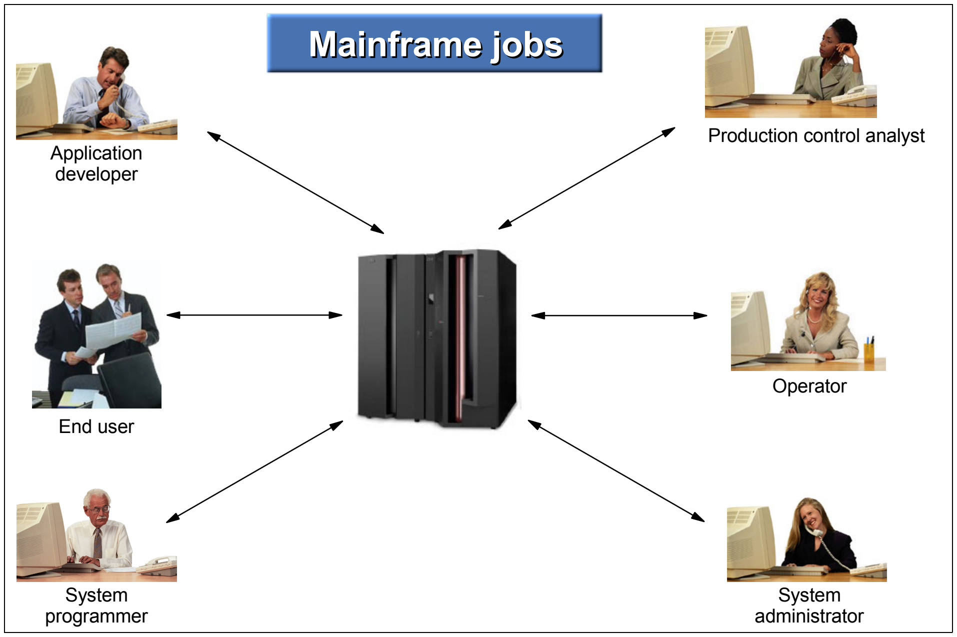


Figure 1-4 Who is who in the mainframe world?

In the IT field, these roles are referred to by a number of different titles. This text uses the following:

- System programmers
- System administrators
- Application designers and programmers
- System operators
- Production control analysts

In a distributed systems environment, many of the same roles are needed as in the mainframe environment. However, the job responsibilities are often not as well defined. Since the 1960s, mainframe roles have evolved and expanded to provide an environment in which the system software and applications can function smoothly and effectively and serve many thousands of users efficiently. While it may seem that the size of the mainframe support staff is large and unwieldy, the numbers become comparatively small when one considers the number of users supported, the number of transactions run, and the high business value of the work that is performed on the mainframe.

This text is concerned mainly with the system programmer and application programmer roles in the mainframe environment. There are, however, several other important jobs involved in the *care and feeding* of the mainframe, and we touch on some of these roles to give you a better idea of what is going on behind the scenes.

Mainframe activities, such as the following, often require cooperation among the various roles:

- Installing and configuring system software
- Designing and coding new applications to run on the mainframe
- Introduction and management of new workloads on the system, such as batch jobs and online transaction processing
- Operation and maintenance of the mainframe software and hardware

In the following sections, we describe each role in more detail.

1.9.1 Who is the system programmer?

In a mainframe IT organization, the system programmer (or systems programmer) plays a central role. The system programmer installs, customizes, and maintains the operating system, and also installs or upgrades products that run on the system. The system programmer might be presented with the latest version of the operating system to upgrade the existing systems. Or, the installation might be as simple as upgrading a single program, such as a sort application.

System programmer The person who installs, customizes, and maintains the operating system.

The system programmer performs tasks such as the following:

- Planning hardware and software system upgrades and changes in configuration
- Training system operators and application programmers
- Automating operations
- Capacity planning

- Running installation jobs and scripts
- Performing installation-specific customization tasks
- Installing service to fix encountered problems or to prevent occurrence of already identified problems
- Integration-testing the new products with existing applications and user procedures
- System-wide performance tuning to meet required levels of service

The system programmer must be skilled at debugging problems with system software. These problems are often captured in a copy of the computer's memory contents called a *dump*, which the system produces in response to a failing software product, user job, or transaction. Armed with a dump and specialized debugging tools, the system programmer can determine where the components have failed. When the error has occurred in a software product, the system programmer works directly with the software vendor's support representatives to discover whether the problem's cause is known and whether a fix is available.

System programmers are also needed to install and maintain the *middleware* on the mainframe, such as data management systems and online transaction processing systems. Middleware is a software *layer* between the operating system and the end user or end-user application. It supplies major functions that are not provided by the operating system. Major middleware products such as DB2, DL/I, CICS, and MQ Series can be as complex as the operating system itself.

1.9.2 Who is the system administrator?

The distinction between *system programmer* and *system administrator* varies widely among mainframe sites. In smaller IT organizations, where one person might be called upon to perform several roles, the terms may be used interchangeably.

System administrator The person who maintains the critical business data that resides on the mainframe.

In larger IT organizations with multiple departments, the job responsibilities tend to be more clearly separated. System administrators perform more of the day-to-day tasks related to maintaining the critical business data that resides on the mainframe, while the system programmer focuses on maintaining the system itself. One reason for the separation of duties is to comply with auditing procedures, which often require that no one person in the IT organization be allowed to have unlimited access to sensitive data or resources. Examples of system administrators include the database administrator (DBA) and the security administrator.

While system programmer expertise lies mainly in the mainframe hardware and software areas, system administrators are more likely to have experience with the applications. They often interface directly with the application programmers and end users to make sure that the administrative aspects of the applications are met. These roles are not necessarily unique to the mainframe environment, but they are key to its smooth operation nonetheless.

In larger IT organizations, the system administrator maintains the system software environment for business purposes, including the day-to-day maintenance of systems to keep them running smoothly. For example, the database administrator must ensure the integrity of, and efficient access to, the data that is stored in the database management systems.

Other examples of common system administrator tasks can include:

- Installing software
- Adding and deleting users and maintaining user profiles
- Maintaining security resource access lists
- Managing storage devices and printers
- Managing networks and connectivity
- Monitoring system performance

In matters of problem determination, the system administrator generally relies on the software vendor support center personnel to diagnose problems, read dumps, and identify corrections for cases in which these tasks are not performed by the system programmer.

1.9.3 Who are the application designers and programmers?

The *application designer* and *application programmer* (or *application developer*) design, build, test, and deliver mainframe applications for the company's end users and customers. Based on requirements gathered from business analysts and end users, the designer creates a design specification from which the programmer constructs an application. The process includes several iterations of code changes and compilation, application builds, and unit testing.

During the application development process, the designer and programmer must interact with other roles in the enterprise. For example, the programmer often works on a team of other programmers who are building code for related application program modules. When completed, each module is passed through a testing process that can include function, integration, and system-wide tests. Following the tests, the application programs must be acceptance tested by the user community to determine whether the code actually satisfies the original user requirement.

In addition to creating new application code, the programmer is responsible for maintaining and enhancing the company's existing mainframe applications. In fact, this is often the primary job for many of today's mainframe application programmers. While mainframe installations still create new programs with Common Business Oriented Language (COBOL) or PL/I, languages such as Java™ have become popular for building new applications on the mainframe, just as they have on distributed platforms.

Widespread development of mainframe programs written in high-level languages such as COBOL and PL/I continues at a brisk pace, despite rumors to the contrary. Many thousands of programs are in production on mainframe systems around the world, and these programs are critical to the day-to-day business of the corporations that use them. COBOL and other high-level language programmers are needed to maintain existing code and make updates and modifications to existing programs. Also, many corporations continue to build new application logic in COBOL and other traditional languages, and IBM continues to enhance their high-level language compilers to include new functions and features that allow those languages to continue to take advantage of newer technologies and data formats.

We will look at the roles of application designer and application programmer in more detail in Part 2, “Application programming on z/VSE” on page 193.

1.9.4 Who is the system operator?

The *system operator* monitors and controls the operation of the mainframe hardware and software. The operator starts and stops system tasks, monitors the system consoles for unusual conditions, and works with the system programming and production control staff to ensure the health and normal operation of the systems.

Console messages can be so voluminous that operators often have a difficult time determining whether a situation is really a problem. In recent years, tools to reduce the volume of messages and automate message responses to routine situations have made it easier for operators to concentrate on unusual events that might require human intervention.

System operator The person who monitors and controls the operation of the mainframe hardware and software.

The operator is also responsible for starting and stopping the major subsystems, such as transaction processing systems, database systems, and the operating system itself. These *restart operations* are not nearly as commonplace as they once were, as the availability of the mainframe has improved dramatically over the years. However, the operator must still perform an orderly shutdown and startup of the system and its workloads, when it is required.

In case of a failure or an unusual situation, the operator communicates with system programmers, who assist the operator in determining the proper course of action, and with the production control analyst, who works with the operator to make sure that production workloads are completing properly.

1.9.5 Who is the production control analyst?

The *production control analyst* is responsible for making sure that batch workloads run to completion without error or delay. Some mainframe installations run interactive workloads for online users, followed by batch updates that run after the prime shift when the online systems are not running. While this execution model is still common, world-wide operations at many companies (with live, Internet-based access to production data) are finding the *daytime online/night time batch* model to be obsolete. Batch workloads continue to be a part of information processing, however, and skilled production control analysts play a key role.

Production control analyst The person who ensures that batch workloads run to completion without error or delay.

A common complaint about mainframe systems is that they are inflexible and hard to work with, specifically in terms of implementing changes. The production control analyst often hears this type of complaint, but understands that the use of well-structured rules and procedures to control changes—a strength of the mainframe environment—helps to prevent outages. In fact, one reason that mainframes have attained a strong reputation for high levels of availability and performance is that there are controls on change and it is difficult to introduce change without proper procedures.

1.9.6 What role do vendors play?

A number of vendor roles are commonplace in the mainframe shop. Because most mainframe computers are sold by IBM, and the operating systems and primary online systems are also provided by IBM, most vendor contacts are IBM employees. However, *independent software vendor* (ISV) products are also used in the IBM mainframe environment, and customers use *original equipment manufacturer* (OEM) hardware, such as disk and tape storage devices, as well.

Typical vendor roles are:

- Hardware support or customer engineer

  Hardware vendors usually provide on-site support for hardware devices. The IBM hardware maintenance person is often referred to as the *customer engineer* (CE). The CE provides installation and repair service for the mainframe hardware and peripherals. The CE usually works directly with the operations teams when hardware fails or new hardware is being installed.

- Software support

A number of vendor roles exist to support software products on the mainframe[7]. IBM has a centralized *Support Center* that provides entitled and extra-charge support for software defects or usage assistance. There are also information technology specialists and architects who can be engaged to provide additional pre-sales and post-sales support for software products, depending upon the size of the enterprise and the particular customer situation.

- Field technical sales support, systems engineer, or client representative

  For larger mainframe accounts, IBM and other vendors provide face-to-face sales support. The vendor representatives specialize in various types of hardware or software product families and call on the part of the customer organization that influences the product purchases. At IBM, the technical sales specialist is referred to as the field technical sales support (FTSS) person, or by the older term, systems engineer (SE).

  For larger mainframe accounts, IBM frequently assigns a client representative*,* who is attuned to the business issues of a particular industry sector, to work exclusively with a small number of customers. The client representative acts as the general *single point of contact* between the customer and the different organizations within IBM.

1.10 z/VSE and other mainframe operating systems

Much of this text is concerned with teaching you the fundamentals of z/VSE, which is one IBM mainframe operating system. We begin discussing z/VSE concepts in Chapter 2, "z/VSE overview" on page 35. It is useful for mainframe students, however, to have a working knowledge of other mainframe operating systems. One reason is that a given mainframe computer might run multiple operating systems. For example, the use of z/VSE, z/VM, and Linux on the same mainframe is common.

Mainframe operating systems are sophisticated products with substantially different characteristics and purposes, and each could justify a separate book for a detailed introduction. Besides z/VSE, four other operating systems dominate mainframe usage: z/VM, z/OS, Linux on System z, and z/TPF.

[7] This text does not examine the marketing and pricing of mainframe software. However, the availability and pricing of middleware and other licensed programs is a critical factor affecting the growth and use of mainframes.

1.10.1 z/VSE

Virtual Storage Extended (z/VSE) is popular with users of smaller mainframe computers. Some of these customers eventually migrate to z/OS when they grow beyond the capabilities of z/VSE.

Compared to z/OS, the z/VSE operating system provides a smaller, less complex base for batch processing and transaction processing. The design and management structure of z/VSE is excellent for running routine production workloads consisting of multiple batch jobs (running in parallel) and extensive, traditional transaction processing. In practice, most z/VSE users also have the z/VM operating system and use this as a general terminal interface for z/VSE application development and system management.

z/VSE was originally known as *Disk Operating System* (DOS), and was the first disk-based operating system introduced for the System/360 mainframe computers. DOS was seen as a temporary measure until OS/360 would be ready. However, some mainframe customers liked its simplicity (and small size) and decided to remain with it after OS/360 became available. DOS became known as DOS/VS (when it started using virtual storage), then VSE/SP, and later VSE/ESA, and most recently z/VSE. The name VSE is often used collectively to refer to any of the more recent versions.

1.10.2 z/VM

z/Virtual Machine (z/VM) has two basic components: a *control program* (CP) and a single-user operating system, CMS. As a control program, z/VM is a *hypervisor* because it runs other operating systems in the virtual machines it creates. Any of the IBM mainframe operating systems such as z/OS, Linux on System z, z/VSE, and z/TPF can be run as *guest systems* in their own virtual machines, and z/VM can run any combination of guest systems.

The control program artificially creates multiple virtual machines from the real hardware resources. To end users, it appears as if they have dedicated use of the shared real resources. The shared real resources include printers, disk storage devices, and the CPU. The control program ensures data and application security among the *guest systems.* The real hardware can be shared among the guests, or dedicated to a single guest for performance reasons. The system programmer allocates the real devices among the guests. For most customers, the use of guest systems avoids the need for larger hardware configurations.

z/VM's other major component is the *Conversational Monitor System* (CMS). This component of z/VM runs in a virtual machine and provides both an interactive end-user interface and the general z/VM application programming interface.

1.10.3 z/OS

The operating system z/OS[8] is popular with users of bigger mainframe computers. Some of these customers eventually migrated to z/OS when they grow beyond the capabilities of z/VSE. It is designed to offer a stable, secure, and continuously available environment for applications running on the mainframe.

z/OS today is the result of technological advancement. z/OS evolved from an operating system that could process a single program at a time to an operating system that can handle many thousands of programs and interactive users concurrently.

An extensive set of system facilities and unique attributes makes z/OS well suited for processing large, complex workloads, such as those that require many I/O operations, access to large amounts of data, or comprehensive security. Typical mainframe workloads include long-running applications that update millions of records in a database and online applications that can serve many thousands of users concurrently. Modern technologies like Web application Servers and Java workload enable z/OS to host modern global solutions. The reliability and scalability of z/OS allows the positioning of it as a central point of control for the entire enterprise, including heterogeneous platforms.

1.10.4 Linux for System z

Several (non-IBM) Linux distributions can be used on a mainframe. There are other generic names for these distributions:

- Linux for S/390 (uses 31-bit addressing and 32-bit registers)
- Linux for System z (uses 64-bit addressing and registers)

The phrase *Linux on System z* is used to refer to Linux running on an S/390 or System z, when there is no specific need to refer explicitly to either the 31-bit version or the 64-bit version. We assume that students are generally familiar with Linux, and therefore we mention only those characteristics that are relevant for mainframe usage. These include the following:

- Linux uses traditional count key data (CKD)[9] disk devices and SAN-connected SCSI type devices. Other mainframe operating systems can

[8] z/OS is designed to take advantage of the IBM System z architecture, or z/Architecture, which was introduced in 2000.
[9] CKD devices are formatted such that the individual data pieces can be accessed directly by the read head of the disk.

recognize these drives as Linux drives, but cannot use the data formats on the drives. That is, there is no sharing of data between Linux and other mainframe operating systems.

- Linux does not use 3270 display terminals, while all other mainframe operating systems use 3270s as their basic terminal architecture.[10] Linux uses X Window System based terminals or X-Window System emulators on PCs. It also supports typical ASCII terminals, usually connected through the *Telnet* protocol. The X-Window System is the standard for graphical interfaces in Linux. It is the middle layer between the hardware and the window manager.
- With the proper setup, a Linux system under z/VM can be quickly cloned to make another, separate Linux image. The z/VM emulated LAN can be used to connect multiple Linux images and to provide an external LAN route for them. Read-only file systems, such as a typical /usr file system, can be shared by Linux images.
- Linux on a mainframe operates with the ASCII character set, not the EBCDIC[11] form of stored data that is typically used on mainframes. Here, EBCDIC is used only when writing to such character-sensitive devices as displays and printers. The Linux drivers for these devices handle the character translation.

1.10.5 z/TPF

The z/Transaction Processing Facility (z/TPF) operating system is a special-purpose system that is used by companies with very high transaction volume, such as credit card companies and airline reservation systems. z/TPF was once known as Airline Control Program (ACP). It is still used by airlines and has been extended for other very large systems with high-speed, high-volume transaction processing requirements.

z/TPF can use multiple mainframes in a loosely coupled environment to routinely handle tens of thousands of transactions per second, while experiencing uninterrupted availability that is measured in years. Very large terminal networks, including special-protocol networks used by portions of the reservation industry, are common.

[10] There is a Linux driver for minimal 3270 operation, in very restrictive modes, but this is not commonly used. 3270 terminals were full-screen buffered non-intelligent terminals, with control units and data streams to maximize efficiency of data transmission.
[11] EBCDIC, which stands for extended binary coded decimal interchange code, is a coded character set of 256 8-bit characters that was developed for the representation of textual data. EBCDIC is not compatible with ASCII character coding. For a handy conversion table see Appendix D, “EBCDIC - ASCII table” on page 609.

1.11 Summary

Today, mainframe computers play a central role in the daily operations of most of the world's largest corporations, including many Fortune 1000 companies. While other forms of computing are used extensively in business in various capacities, the mainframe occupies a coveted place in today's e-business environment. In banking, finance, health care, insurance, utilities, government, and a multitude of other public and private enterprises, the mainframe computer continues to form the foundation of modern business.

The new mainframe owes much of its popularity and longevity to its inherent reliability and stability, a result of continuous technological advances since the introduction of the IBM System/360 in 1964. No other computer architecture in existence can claim as much continuous, evolutionary improvement, while maintaining compatibility with existing applications.

The term *mainframe* has gradually moved from a physical description of larger IBM computers to the categorization of a style of computing. One defining characteristic of the mainframe has been a continuing compatibility that spans decades.

The roles and responsibilities in a mainframe IT organization are wide and varied. It takes skilled staff to keep a mainframe computer running smoothly and reliably. It might seem that there are far more resources needed in a mainframe environment than for small, distributed systems. But, if roles are fully identified on the distributed systems side, a number of the same roles exist there as well.

Several operating systems are currently available for mainframes. This text concentrates on one of these, z/VSE. However, mainframe students should be aware of the existence of the other operating systems and understand their positions relative to z/VSE.

Key terms in this chapter				
Architecture	Availability	Batch processing	Compatibility	E-business
Mainframe	Online transaction processing (OLTP)	Platform	Production control analyst	Scalability
Server farm	System administrator	System operator	System programmer	System/360

1.12 Questions for review

To help test your understanding of the material in this chapter, complete the following questions:

1. List ways in which the mainframe of today challenges the traditional thinking about centralized computing versus distributed computing.
2. Explain how businesses make use of mainframe processing power, and how mainframe computing differs from other types of computing.
3. List three strengths of mainframe computing, and outline the major types of workloads for which mainframes are best suited.
4. Name five jobs or responsibilities that are related to mainframe computing.
5. This chapter mentioned at least five operating systems that are used on the mainframe. Choose three of them and describe the main characteristics of each.

1.13 Topics for further discussion

1. What is a mainframe today? How did the term arise? Is it still appropriate?
2. Why is it important to maintain system compatibility for older applications? Why not simply change existing application programming interfaces whenever improved interfaces become available?
3. Describe how running a mainframe can be cost-effective, given the large number of roles needed to run a mainframe system.
4. What characteristics, good or bad, exist in a mainframe processing environment because of the roles that are present in a mainframe shop? (Efficiency? Reliability? Scalability?)
5. Most mainframe shops have implemented very rigorous systems management, security, and operational procedures. Have these same procedures been implemented in distributed system environments? Why or why not?
6. Can you find examples of mainframe use in your everyday experiences? Describe them and the extent to which mainframe processing is apparent to end users. Examples might include the following:
   – Popular Web sites that rely on mainframe technology as the back-end server to support online transactions and databases.
   – Mainframes used in your locality. These might include banks and financial centers, major retailers, transportation hubs, and the health and medical industries.

7. Can you find examples of distributed systems in everyday use? Could any of these systems be improved through the addition of a mainframe? How?

2

Mainframe hardware systems

Objective: As a new z/VSE system programmer, you will need to develop a thorough understanding of the hardware that runs the z/VSE operating system. z/VSE is designed to make full use of mainframe hardware and its many sophisticated peripheral devices.

After completing this chapter, you will be able to:

- Discuss S/360 and System z hardware design.
- Explain processing units and disk hardware.
- Explain how mainframes differ from PC systems in data encoding.
- List some typical hardware configurations.

2.1 Introduction to mainframe hardware systems

This chapter provides an overview of mainframe hardware systems, with most of the emphasis on the processor *box*.

Related reading: For detailed descriptions of the major facilities of z/Architecture, the book *z/Architecture Principles of Operation* is the standard reference. You can find this and other IBM publications at the z/VSE Internet Library Web site:

`http://www.ibm.com/servers/eserver/zseries/zvse/documentation/`

CPU
Synonymous with *processor*

Let us begin this chapter with a look at the terminology associated with mainframe hardware. Being aware of various meanings of the terms *systems*, *processors*, *CPs*, and so forth is important for your understanding of mainframe computers.

In the early S/360 days a system had a single processor, which was also known as the central processing unit (CPU). The terms *system*, *processor*, and *CPU* were used interchangeably. However, these terms became confusing when systems became available with more than one processor. This is illustrated in Figure 2-1.

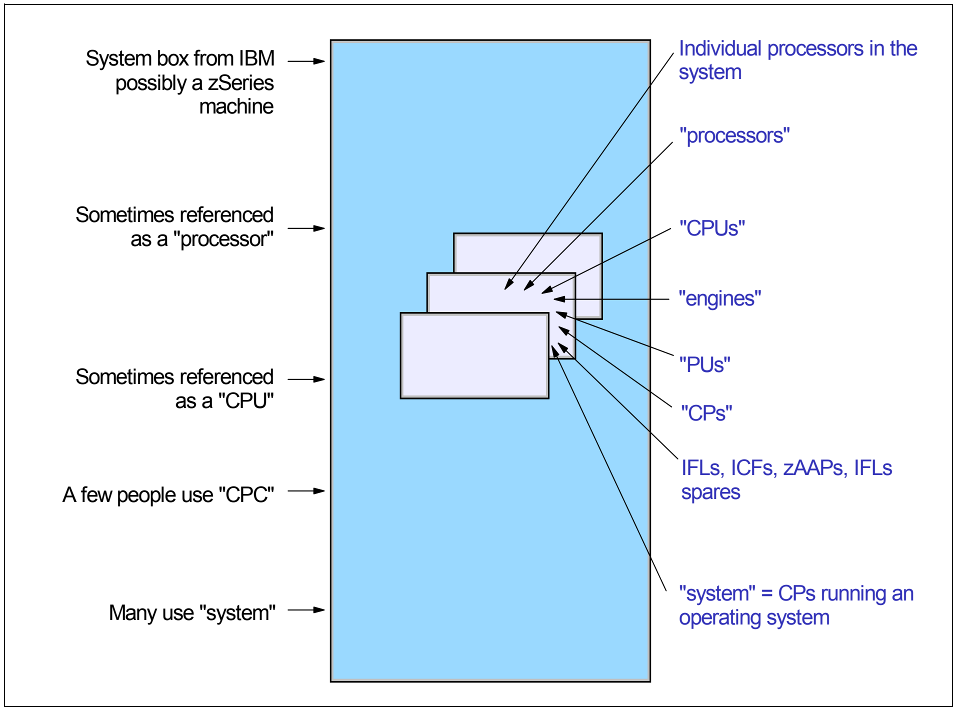

System box from IBM possibly a zSeries machine
Sometimes referenced as a "processor"
Sometimes referenced as a "CPU"
A few people use "CPC"
Many use "system"
Individual processors in the system
"processors"
"CPUs"
"engines"
"PUs"
"CPs"
IFLs, ICFs, zAAPs, IFLs spares
"system" = CPs running an operating system

Figure 2-1 Terminology overlap

CPC
The physical collection of hardware that includes main storage, one or more central processors, timers, and channels.

Processor and *CPU* can refer to either the complete system box, or to one of the processors (CPUs) within the system box. Although the meaning may be clear from the context of a discussion, even mainframe professionals must clarify which processor or CPU meaning they are using in a discussion. System programmers use the IBM term *central processor complex* (CPC) to refer to the mainframe box. In this text we use the term CPC to refer to the physical collection of hardware that includes main storage, one or more central processors, timers, and channels.

Partitioning and some of the terms in Figure 2-1 are discussed later in this chapter. Briefly, all of the S/390 or z/Architecture processors within a CPC are *processing units* (PUs). When IBM delivers the CPC, the PUs are characterized as CPs (for normal work), Integrated Facility for Linux (IFL), Integrated Coupling Facility (ICF) for Parallel Sysplex configurations, and so forth.

In this text, we hope that the meanings of *system* and *processor* are clear from the context. We normally use *system* to indicate the hardware box, a complete hardware environment (with I/O devices), or an operating environment (with software), depending on the context. We normally use *processor* to mean a single processor (CP) within the CPC.

2.2 Early system design

The central processor box contains the processors, memory,[1] control circuits, and interfaces for *channels*. A channel provides an independent data and control path between I/O devices and memory. Early systems had up to 16 channels. The largest mainframe machines at the time of writing can have over 1,000 channels.

Channels connect to *control units*. A control unit contains logic to work with a particular type of I/O device. A control unit for a printer would have much different internal circuitry and logic than a control unit for a tape drive, for example. Some control units can have multiple channel connections providing *multiple paths* to the control unit and its devices.

Control units connect to *devices*, such as disk drives, tape drives, communication interfaces, and so forth. The division of circuitry and logic between a control unit and its devices is not defined, but it is usually more economical to place most of the circuitry in the control unit.

[1] Some S/360s had separate boxes for memory. However, this is a conceptual discussion and we ignore such details.

Figure 2-2 presents a conceptual diagram of a S/360 system. Current systems are not connected, as shown in Figure 2-2. However, this figure helps explain the background terminology that permeates mainframe discussions.

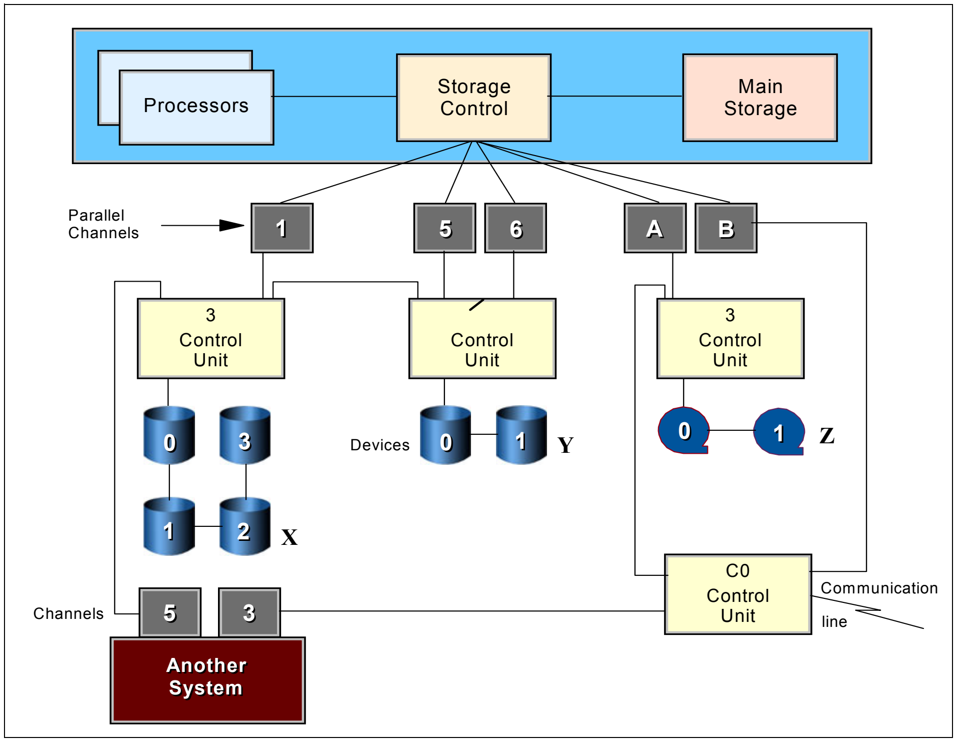


Figure 2-2 Conceptual S/360

The channels in Figure 2-2 are *parallel channels* (also known as *bus and tag* channels because of the two heavy copper cables they use). A parallel channel can be connected to a maximum of eight control units. Most control units can be connected to multiple devices. The maximum depends on the particular control unit, but 16 is a typical number.

Each channel, control unit, and device has an address, expressed as a hexadecimal number. The disk drive marked with an X in Figure 2-2 on page 39 has address 132, derived as shown in Figure 2-3.

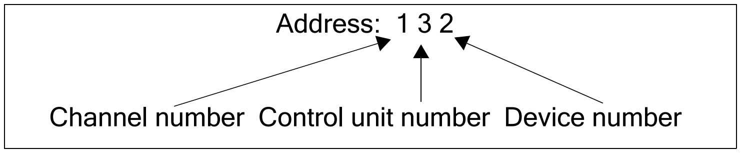


Figure 2-3 Device address

The disk drive marked with a Y in the figure can be addressed as 171, 571, or 671 because it is connected through three channels. By convention the device is known by its lowest address (171), but all three addresses could be used by the operating system to access the disk drive. Multiple paths to a device are useful for performance and for availability. When an application wants to access disk 171, the operating system will first try channel 1. If it is busy (or not available), it will try channel 5, and so forth.

Figure 2-2 on page 39 contains another S/360 system with two channels connected to control units used by the first system. This sharing of I/O devices is common in all mainframe installations. Tape drive Z is address A31 for the first system, but is address 331 for the second system. Sharing devices, especially disk drives, is not a simple topic, and there are hardware and software techniques used by the operating system to control exposures such as updating the same disk data at the same time from two independent systems.

As mentioned, current mainframes are not used exactly as shown in Figure 2-2 on page 39. Differences include:

- Parallel channels are not available on the newest mainframes and are slowly being displaced on older systems.
- Parallel channels have been replaced with ESCON (Enterprise Systems CONnection) and FICON (FIber CONnection) channels. These channels connect to only one control unit or, more likely, are connected to a *director* (switch) and are optical fibers.
- Current mainframes have more than 16 channels and use two hexadecimal digits as the channel portion of an address.
- Channels are generally known as Channel Path Identifiers (CHPIDs) or Physical Channel IDs (PCHIDs) on later systems, although the term *channel* is also correct. The channels are all integrated in the main processor box.

ESCON
Enterprise System Connection

The device address seen by software is more correctly known as a device number (although the term *address* is still widely used) and is indirectly related to the control unit and device addresses.

For more information about the development of the IBM mainframe since 1964 see Appendix A, “A brief look at IBM mainframe history” on page 581.

2.3 Current design

Current CPC designs are considerably more complex than the early S/360 design. This complexity includes many areas:

- I/O connectivity and configuration
- I/O operation
- Partitioning of the system

2.3.1 I/O connectivity

Figure 2-4 illustrates a recent configuration. A real system would have more channels and I/O devices, but this figure illustrates key concepts. Partitions, ESCON channels, and FICON channels are described later.

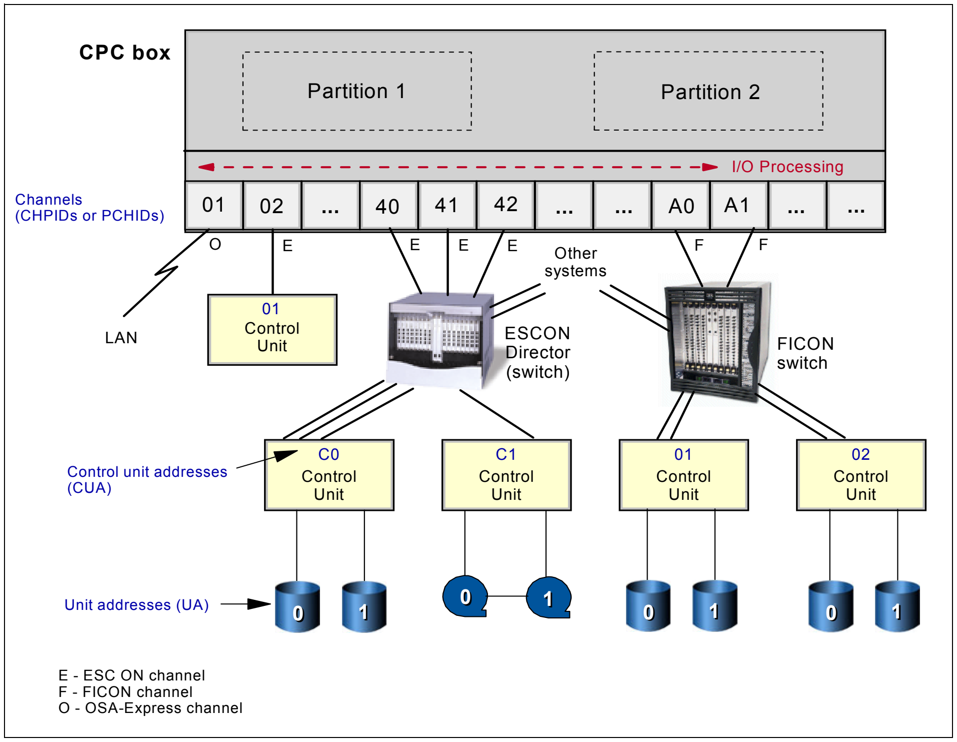


Figure 2-4 Recent system configuration

Briefly, partitions create separate logical machines in the CPC. ESCON and FICON channels are logically similar to parallel channels, but they use fiber connections and operate much faster. A modern system might have 100–200 channels or CHPIDs.[2] Key concepts partly illustrated here include the following:

- ESCON and FICON channels connect to only one device or one port on a switch.
- Most modern mainframes use switches between the channels and the control units. The switches may be connected to several systems, sharing the control units and some or all of its I/O devices across all the systems.

[2] The more recent mainframe machines can have more than 256 channels, but an additional setup is needed for this. The channels are assigned in a way that only two hexadecimal digits are needed for CHPID addresses.

CHPID
Channel path identifier

- CHPID addresses are two hexadecimal digits.
- Multiple partitions can sometimes share CHPIDs. Whether this is possible depends on the nature of the control units used through the CHPIDs. In general, CHPIDs used for disks can be shared.
- An I/O subsystem layer exists between the operating systems in partitions (or in the basic machine if partitions are not used) and the CHPIDs.

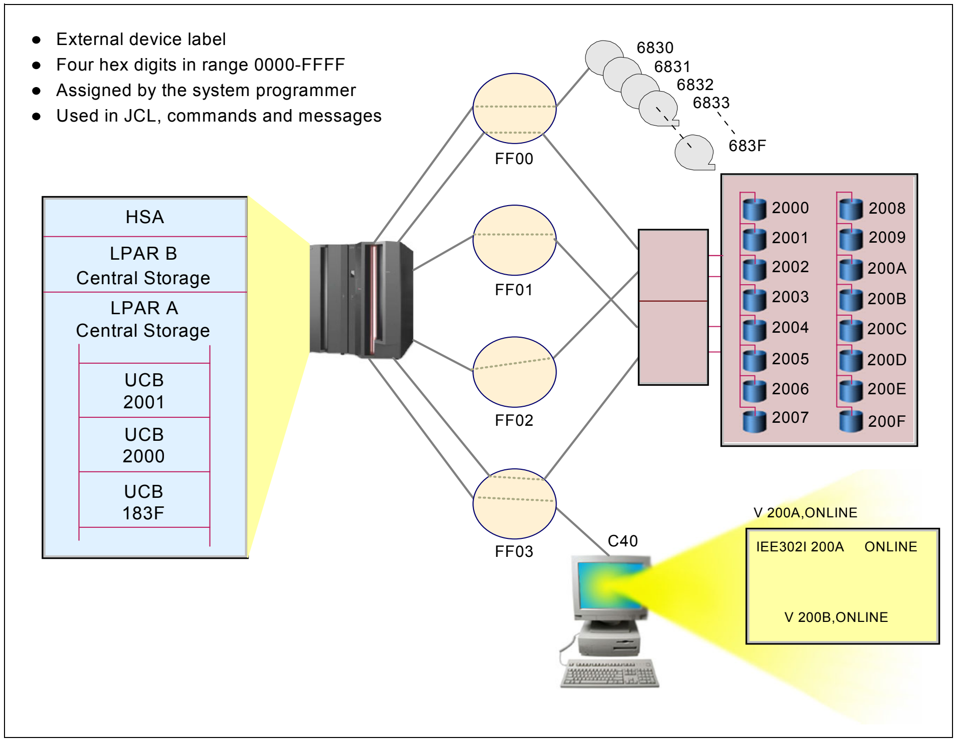
An ESCON director or FICON switch is a sophisticated device that can sustain high data rates through many connections. (A large director might have 200 connections, for example, and all of these can be passing data at the same time.) The director or switch must keep track of which CHPID (and partition) initiated which I/O operation so that data and status information is returned to the right place. Multiple I/O requests, from multiple CHPIDs attached to multiple partitions on multiple systems, can be in progress through a single control unit.

The I/O control layer uses a control file known as an I/O Control Data Set (IOCDS) that translates physical I/O addresses (composed of CHPID numbers, switch port numbers, control unit addresses, and unit addresses) into *device numbers* that are used by the operating system software to access devices. This is loaded into the Hardware Save Area (HSA) at power-on and can be modified dynamically. A device number looks like the addresses we described for early S/360 machines except that it can contain three or four hexadecimal digits.

Many users still refer to these as *addresses*, although the device numbers are arbitrary numbers between x'0000' and x'FFFF'. z/VSE uses only the range of device numbers between x'0000' and x'0FFF'. The newest mainframes, at the time of writing, have two layers of I/O address translations between the real I/O elements and the operating system software. The second layer was added to make migration to newer systems easier.

Modern control units, especially for disks, often have multiple channel (or switch) connections and multiple connections to their devices. They can handle multiple data transfers at the same time on the multiple channels. Each device will have a unit control block (UCB) in each z/VSE image.

Figure 2-5 shows the device addressing in general. However, z/VSE supports only device numbers from x'0000' to x'0FFF'.



```
V 200A,ONLINE
IEE302I 200A     ONLINE
V 200B,ONLINE
```

Figure 2-5 Device addressing in general

2.3.2 System control and partitioning

There are many ways to illustrate a mainframe's internal structure, depending on what we wish to emphasize. Figure 2-6, while highly conceptual, shows several of the functions of the internal system controls on current mainframes. The internal controllers are microprocessors but use a much simpler organization and instruction set than System z processors. They are usually known as *controllers* to avoid confusion with System z *processors*.

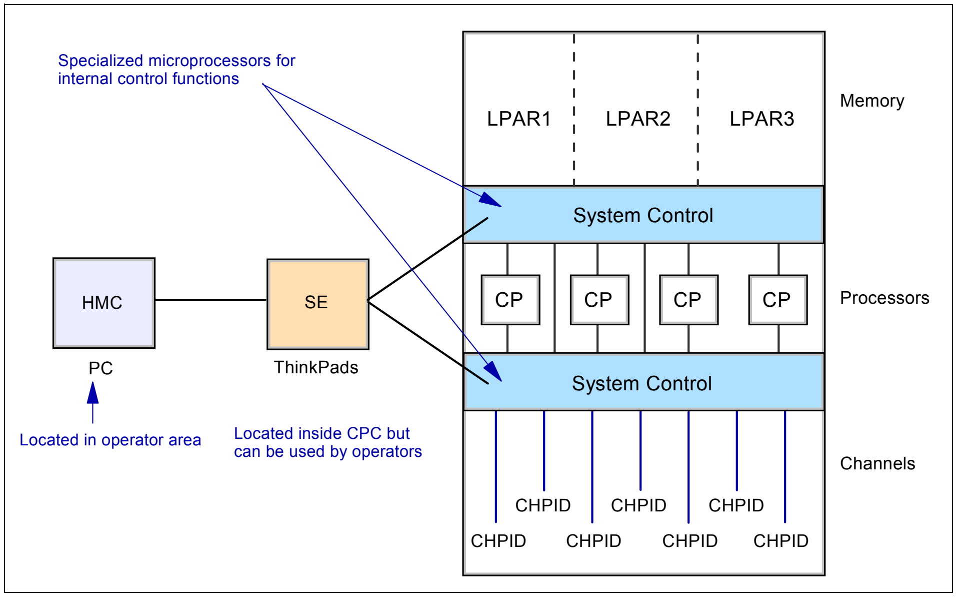


Figure 2-6 System control and partitioning

Logical partition
A subset of the processor hardware that is defined to support an operating system

Among the system control functions is the capability to partition the system into several *logical partitions* (LPARs). An LPAR is a subset of the processor hardware that is defined to support an operating system. An LPAR contains resources (processors, memory, and input/output devices) and operates as an independent system. Multiple logical partitions can exist within a mainframe hardware system.

For many years there was a limit of 15 LPARs in a mainframe. More recent machines have a limit of 30 (and potentially more). Practical limitations of memory size, I/O availability, and available processing power usually limit the number of LPARs to less than these maximums.

Note: The hardware and firmware that provide partitioning are known as Processor Resource/System Manager (PR/SM). It is the PR/SM functions that are used to create and run LPARs. This difference between PR/SM (a built-in facility) and LPARs (the result of using PR/SM) is often ignored, and the term LPAR is used collectively for the facility and its results.

System administrators assign portions of memory to each LPAR. Memory cannot be shared among LPARs. The administrators can assign processors (noted as CPs in Figure 2-6 on page 45) to specific LPARs or they can allow the system controllers to dispatch any or all the processors to all the LPARs using an internal load-balancing algorithm. Channels (CHPIDs) can be assigned to specific LPARs or can be shared by multiple LPARs, depending on the nature of the devices on each channel.

A system with a single processor (CP processor) can have multiple LPARs. PR/SM has an internal dispatcher that can allocate a portion of the processor to each LPAR, much as an operating system dispatcher allocates a portion of its processor time to each process, thread, or task.

Partitioning control specifications are partly contained in the IOCDS and are partly contained in a system *profile*. The IOCDS and profile both reside in the Support Element (SE), which is simply a notebook computer inside the system. The SE can be connected to one or more Hardware Management Consoles (HMCs), which are desktop personal computers. An HMC is more convenient to use than an SE and can control several different mainframes.

Working from an HMC (or from an SE, in unusual circumstances), an operator prepares a mainframe for use by selecting and loading a profile and an IOCDS. These create LPARs and configure the channels with device numbers, LPAR assignments, multiple path information, and so forth. This is known as a Power-on Reset (POR). By loading a different profile and IOCDS, the operator can completely change the number and nature of LPARs and the appearance of the I/O configuration. However, doing this is usually disruptive to any running operating systems and applications and is therefore seldom done without advance planning.

2.3.3 Characteristics of LPARs

LPARs are, in practice, equivalent to separate mainframes. Each LPAR runs its own operating system. This can be any mainframe operating system. There is no need to run z/VSE, for example, in each LPAR. The installation planners may elect to share I/O devices across several LPARs, but this is a local decision.

The system administrator can assign one or more system processors for the exclusive use of an LPAR. Alternately, the administrator can allow all processors to be used on some or all LPARs. Here, the system control functions (often known as microcode or firmware) provide a dispatcher to share the processors among the selected LPARs. The administrator can specify a maximum number of concurrent processors executing in each LPAR. The administrator can also provide weightings for different LPARs (for example, specifying that LPAR1 should receive twice as much processor time as LPAR2).

The operating system in each LPAR is IPLed separately, has its own copy[3] of its operating system, has its own operator console (if needed), and so forth. If the system in one LPAR crashes, there is no effect on the other LPARs.

In Figure 2-6 on page 45, for example, we might have a production z/VSE in LPAR1, a test version of z/VSE LPAR2, and Linux for S/390 in LPAR3. If our total system has 8 GB of memory, we might have assigned 4 GB to LPAR1, 1 GB to LPAR2, 1 GB to LPAR3, and have kept 2 GB in reserve for some reason. The operating system consoles for the two z/VSE LPARs might be in completely different locations[4].

For most practical purposes there is no difference between, for example, three separate mainframes running z/VSE (and sharing most of their I/O configuration) and three LPARs on the same mainframe doing the same thing. With minor exceptions z/VSE, the operators and applications cannot detect the difference.

2.3.4 Consolidation of mainframes

There are fewer mainframes in use today than there were 15 or 20 years ago. In some cases, all of the applications were moved to other types of systems. However, in most cases the reduced number is due to consolidation. That is, several smaller mainframes have been replaced with a smaller number of larger systems.

There is a compelling reason for consolidation. Mainframe software (from many vendors) can be expensive and typically costs more than the mainframe hardware. It is usually less expensive (and sometimes *much* less expensive) to replace multiple software licenses (for smaller machines) with one or two licenses (for larger machines). Software license costs are often linked to the power of the system, but the pricing curves favor a small number of large machines.

Software license costs for mainframes have become a dominant factor in the growth and direction of the mainframe industry. There are several nonlinear

[3] Most, but not all, of the z/VSE system libraries can be shared.
[4] Linux does not have an operator console in the sense of the z/VSE consoles.

factors that make software pricing very difficult. We must remember that mainframe software is not a mass market situation like PC software. The growth of mainframe processing power in recent years has been nonlinear.

The relative power needed to run a traditional mainframe application (a batch job written in COBOL, for example) does not have a linear relation to the power needed for a new application (with a GUI interface, written in C or Java). The consolidation effect has produced very powerful mainframes. These might need 1% of their power to run an application, but the application vendor often sets a price based on the total power of the machine.

This results in the odd situation where customers want the latest mainframe (to obtain new functions or to reduce maintenance costs associated with older machines) but they want the *slowest* mainframe that will run their applications (to reduce software costs based on total system processor power).

2.4 Processing units

Figure 2-1 on page 37 lists several different types of processors in a system. These are all z/Architecture processors that can be used for slightly different purposes[5]. Several of these purposes are related to software cost control, while others are more fundamental.

All these start as equivalent processor units[6] (PUs) or engines. A PU is a processor that has not been *characterized* for use. Each of the processors begins as a PU and is characterized by IBM during installation or at a later time. The potential characterizations are:

- Central Processor (CP)

  This is a processor available to normal operating system and application software.

- System Assistance Processor (SAP)

  Every modern mainframe has at least one SAP. Larger systems may have several. The SAPs execute internal code[7] to provide the I/O subsystem. A SAP, for example, translates device numbers and real addresses of CHPIDs, control unit addresses, and device numbers. It manages multiple paths to control units and performs error recovery for temporary errors. Operating

[5] Do not confuse these with the controller microprocessors. The processors discussed in this section are full, standard mainframe processors.
[6] This discussion applies to the current System z machines at the time of writing. Earlier systems had fewer processor characterizations, and even earlier systems did not use these techniques.
[7] IBM refers to this as Licensed Internal Code (LIC). It is often known as microcode (which is not technically correct) or as firmware. It is definitely not user code.

systems and applications cannot detect SAPs, and SAPs do not use any *normal* memory.

- Integrated Facility for Linux (IFL)

  This is a normal processor with one or two instructions disabled that are used only by z/VSE and z/OS. Linux does not use these instructions and can be executed by an IFL. Linux can be executed by a CP as well. The difference is that an IFL is not counted when specifying the model number[8] of the system. This can make a substantial difference in software costs.

- System z Application Assist Processor (zAAP)

  This is a processor with a number of functions disabled (interrupt handling, some instructions) such that no full operating system can be executed on the processor. However, z/OS can detect the presence of zAAP processors and will use them to execute Java code (and possibly other similar code in the future). The same Java code can be executed on a standard CP. Again, zAAP engines are not counted when specifying the model number of the system. Like IFLs, they exist only to control software costs. z/VSE does not support zAAPs.

- zIIP

  The System z9 Integrated Information Processor (zIIP) is a specialized engine for processing eligible database workloads. The zIIP is designed to help lower software costs for select workloads on the mainframe, such as business intelligence (BI), enterprise resource planning (ERP), and customer relationship management (CRM). The zIIP reinforces the mainframe’s role as the data hub of the enterprise by helping to make direct access to DB2 more cost effective and reducing the need for multiple copies of the data. z/VSE does not support zIIPs.

- Integrated Coupling Facility (ICF)

  These processors run only Licensed Internal Code. They are not visible to normal operating systems or applications. A Coupling Facility is, in effect, a large memory scratch pad used by multiple systems to coordinate work. ICFs must be assigned to LPARs that then become coupling facilities.

- Spare

  An uncharacterized PU functions as a *spare*. If the system controllers detect a failing CP or SAP, it can be replaced with a spare PU. In most cases this can be done without any system interruption, even for the application running on the failing processor.

- Various forms of Capacity on Demand and similar arrangements exist whereby a customer can enable additional CPs at certain times (for unexpected peak loads, for example).

[8] Some systems do not have different models. In this case a *capacity model number* is used.

In addition to these characterizations of processors, some mainframes have models or versions that are configured to operate slower than the potential speed of their CPs. This is widely known as *kneecapping*, although IBM prefers the term *capacity setting*, or something similar. It is done by using microcode to insert null cycles into the processor instruction stream. The purpose, again, is to control software costs by having the minimum mainframe model or version that meets the application requirements. IFLs, SAPs, zAAPs, and ICFs always function at the full speed of the processor since these processors *do not count* in software pricing calculations.[9]

2.5 Multiprocessors

All of the earlier discussions and examples assume that more than one processor (CP) is present in a system (and perhaps in an LPAR). It is possible to purchase a current mainframe with a single processor (CP), but this is not a typical system.[10] The term *multiprocessor* means several processors (CP processors) and implies that several processors are used by a copy of z/VSE.

All operating systems today, from PCs to mainframes, can work in a multiprocessor environment. However, the degree of integration of the multiple processors varies considerably. For example, pending interrupts in a system (or in an LPAR) can be accepted by any processor in the system (or working in the LPAR). Any processor can initiate and manage I/O operations to any channel or device available to the system or LPAR. Channels, I/O devices, interrupts, and memory are owned by the system (or by the LPAR) and not by any specific processor.

This multiprocessor integration appears simple on the surface, but its implementation is complex. It is also important for maximum performance. The ability of any processor to accept any interrupt sent to the system (or to the LPAR) is especially important.

Each processor in a system (or in an LPAR) has a small private area of memory (8 KB starting at real address 0 and always mapped to virtual address 0) that is unique to that processor. This is the prefix storage and is used for interrupt handling and for error handling. A processor can access another processor's prefix storage through special programming, although this is normally done only for error recovery purposes. A processor can interrupt other processors by using

[9] This is true for IBM software but may not be true for all software vendors.
[10] All current IBM mainframes also require at least one SAP, so the minimum system has two processors: one CP and one SAP. However, the use of *processor* in the text usually means a CP processor usable for applications. Whenever discussing a processor other than a CP, we always make this clear.

a special instruction (SIGP, for Signal Processor). Again, this is typically used only for error recovery.

2.6 Disk devices

IBM 3390 disk drives are commonly used on current mainframes. Conceptually, this is a simple arrangement, as shown in Figure 2-7.

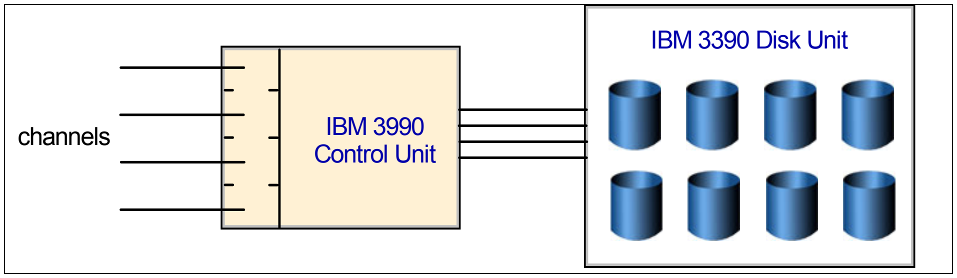


Figure 2-7   Initial IBM 3390 disk implementation

The associated control unit (3990) typically has four channels connected to one or more processors (probably with a switch), and the 3390 unit typically has eight or more disk drives. Each disk drive has the characteristics explained earlier. This illustration shows 3990 and 3390 units, and it also represents the concept or architecture of current devices.

The current equivalent device is an IBM 2105 Enterprise Storage Server, simplistically illustrated in Figure 2-8.

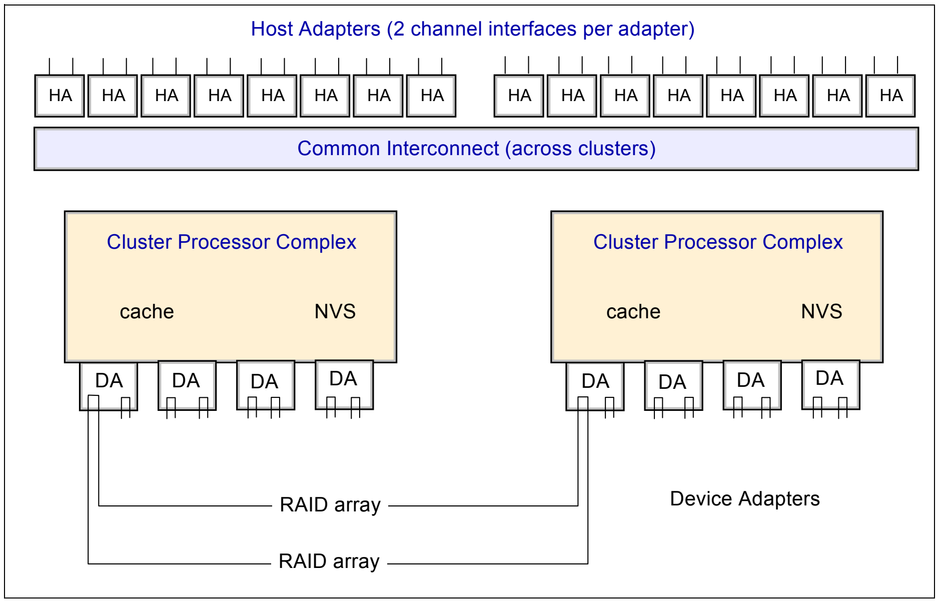

Host Adapters (2 channel interfaces per adapter)
HA
Common Interconnect (across clusters)
Cluster Processor Complex
cache
NVS
Cluster Processor Complex
cache
NVS
DA
RAID array
RAID array
Device Adapters

Figure 2-8 Current 3390 implementation

The 2105 unit is a very sophisticated device. It emulates a large number of control units and 3390 disk drives. It contains up to 11 TB of disk space, has up to 32 channel interfaces, 16 GB cache, and 284 MB of non-volatile memory (used for write queueing). The Host Adapters appear as control unit interfaces and can connect up to 32 channels (ESCON or FICON).

The physical disk drives are commodity SCSI-type units (although a serial interface, known as SSA, is used to provide faster and redundant access to the disks). A number of internal arrangements are possible, but the most common involves many RAID 5 arrays with hot spares. Practically everything in the unit has a spare or fallback unit. The internal processing (to emulate 3990 control units and 3390 disks) is provided by four high-end RISC processors in two processor complexes. Each complex can operate the total system. Internal batteries preserve transient data during short power failures. A separate console is used to configure and manage the unit.

The 2105 offers many functions not available in real 3390 units, including FlashCopy, Extended Remote Copy, Concurrent Copy, Parallel Access Volumes, Multiple Allegiance, a huge cache, and so forth.

A simple 3390 disk drive (with control unit) has different technology from the 2105 just described. However, the basic architectural appearance to software is the same. This allows applications and system software written for 3390 disk drives to use the newer technology with no revisions.[11]

There have been several stages of new technology implementing 3390 disk drives. The 2105 is one of the most recent of these. The process of implementing an architectural standard (in this case the 3390 disk drive and associated control unit) with newer and different technology while maintaining software compatibility is characteristic of mainframe development. As has been mentioned several times, maintaining application compatibility over long periods of technology change is an important characteristic of mainframes.

The architecture of a simple 3390 disk uses a data recording format employing self-defining record formats in which each record is represented by a count area that identifies the record and specifies its format, an optional key area that may be used to identify the data area contents, and a data area that contains the user data for the record. This is called Extended Count-Key-Data (ECKD) architecture.

In addition to ECKD, z/VSE supports Fixed Block Architecture (FBA) and Fiber Channel Protocol (FCP)-attached Small Computer System Interface (SCSI) disks.

The Fixed Block Architecture is based on the relative-block concept. All data is stored in blocks of fixed length. Each block is addressed by its number relative to the beginning of the disk.

The SCSI disk devices use Fixed Block (512-bytes) sectors and are defined in z/VSE as FBA devices. z/VSE SCSI disk support includes support for multipathing, a method to increase the availability of the device, and DASD[12] sharing using the z/VSE lock file on a SCSI disk[13]. SCSI disks can be emulated in 2105 and IBM TotalStorage DS/6000 and IBM TotalStorage DS/8000 disk drives.

[11] Some software enhancements are needed to use some of the new functions, but these are compatible extensions at the operating system level and do not affect application programs.
[12] Disk drives are also known as direct access storage devices (DASD). Theoretically, a DASD could be any direct access device, not only a disk. Nevertheless, the term DASD is used for disks only. For more information about DASD, see Chapter 5, “Working with files” on page 139.
[13] There are some limitations when SCSI disks are used as system disks.

2.7 Typical mainframe systems

We outline the general configurations of three different levels of configuration in this section. These are not intended to be detailed descriptions, but are simply overviews.

2.7.1 Very small systems

The first two examples, in Figure 2-9, show that *mainframe* refers more to a *style* of computing rather than to unique hardware. Two different systems are illustrated, and neither uses mainframe hardware in the generally accepted sense of the term.

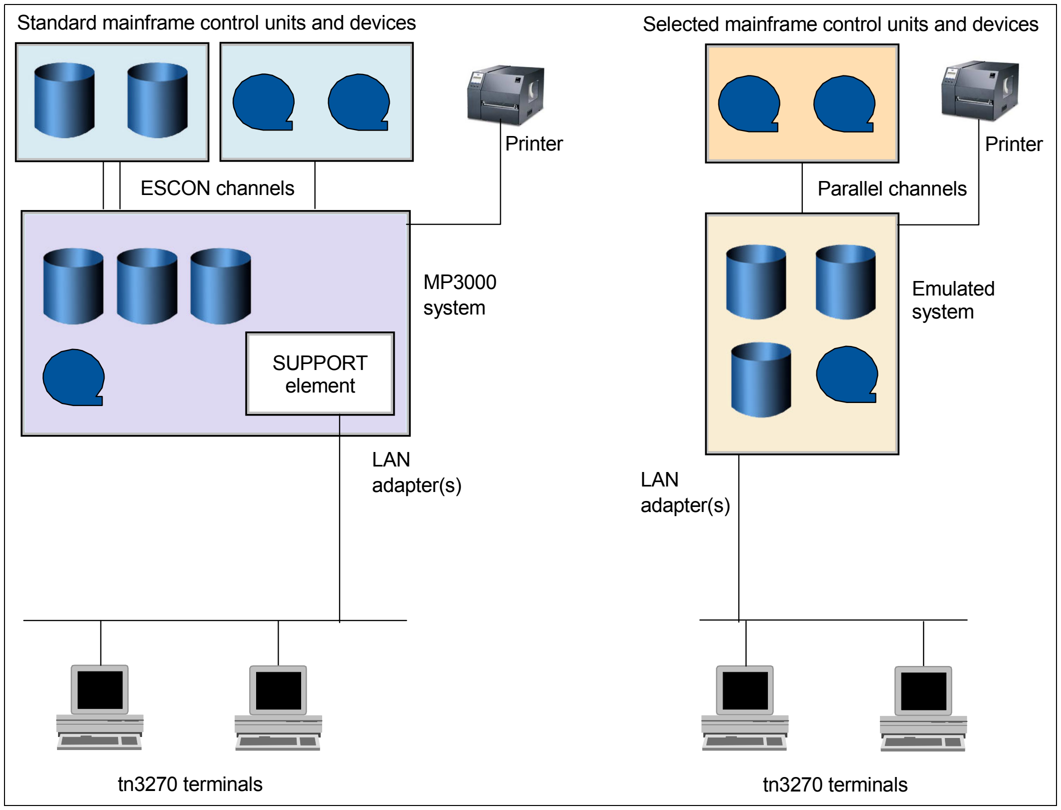


Figure 2-9 Very small mainframe configurations

The first system illustrated is an IBM Multiprise 3000 system (MP3000), which IBM withdrew from marketing as this book was written. It was the smallest S/390 system produced in recent years. The MP3000 has one or two S/390 processors plus a SAP processor. It also has internal disk drives that can be configured to operate as normal IBM 3390 disk drives. A minimal internal tape drive is normally used for software installation. The MP3000 can have a substantial number of ESCON or parallel channels for connection to traditional external I/O devices.

The MP3000 is completely compatible with S/390 mainframes, but lacks later System z features. It is typically used with VM/ESA and VSE/ESA operating systems (previous versions of z/VM and z/VSE).

The second system shown, the emulated System z system, has no mainframe hardware. It is based on a personal computer (running Linux or UNIX) and uses software to emulate a System z, which is able to run z/VSE and other System z operating systems. Special PCI channel adapters can be used to connect to selected mainframe I/O devices. The personal computer running the emulator software can have substantial internal disks (typically in a RAID array) for emulating IBM 3390 disk drives.

Both of these systems lack some features found in *real* mainframes. Nevertheless, both are capable of doing quality work. Typical application software cannot distinguish these systems from real mainframes. In fact, these are considered mainframes because their operating systems, their middleware, their applications, and their style of usage are the same as for larger mainframes. The MP3000 can be configured with LPARs and might run both test and production systems. The emulated system does not provide LPARs, but can accomplish much the same thing by running multiple copies of the emulator software.

A key attraction of these systems is that they can be a *mainframe in a box*. In many cases no external traditional I/O devices are needed. This greatly reduces the entry-level price for a mainframe system.

2.7.2 Medium single systems

Figure 2-10 shows a modest mainframe system and the typical external elements needed. The particular system shown is an IBM z890 system with two recent external disk controllers, a number of tape drives, printers, LAN attachments, and consoles.

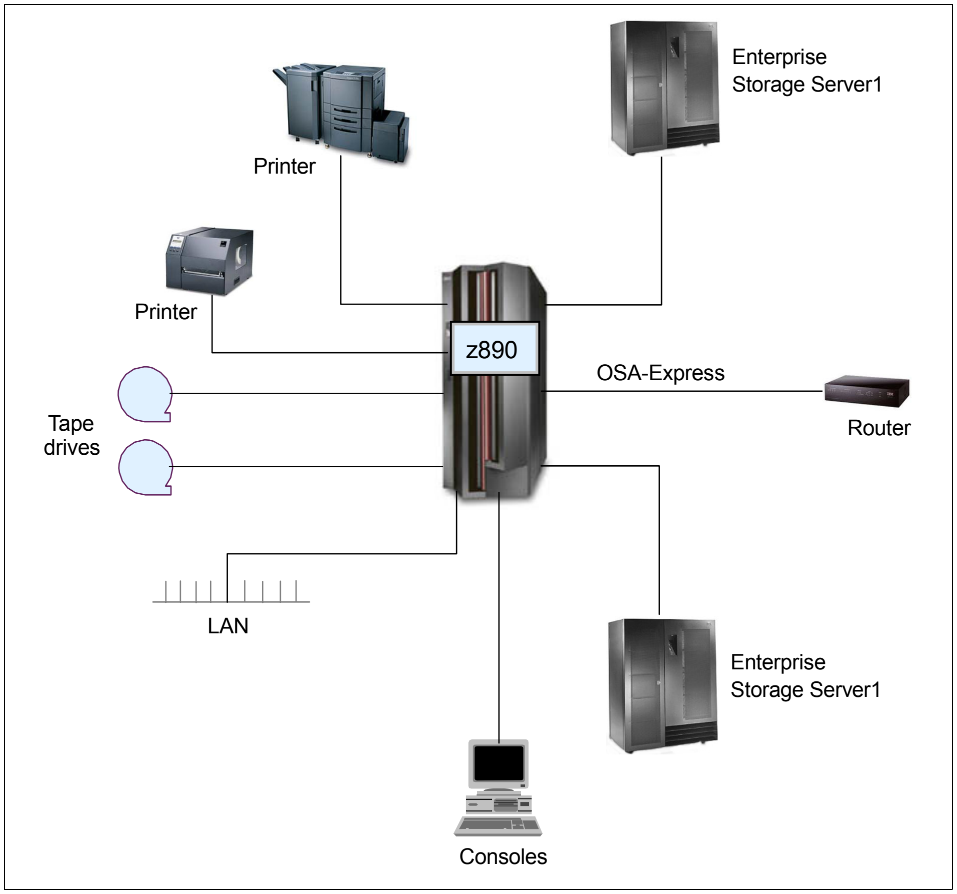


Figure 2-10 Medium mainframe configuration

This is a somewhat idealized configuration, in that no older devices are involved. The systems outlined here might have a number of LPARs active, for example:

- A production z/VSE system running interactive applications.
- A second production z/VSE devoted to major batch applications. (These could also be run in the first LPAR, but some installations prefer a separate LPAR for management purposes.)

- A test z/VSE version for testing new software releases, new applications, and so forth.
- One or several Linux partitions, perhaps running Web-related applications.

The disk controllers in Figure 2-10 on page 56 contain a large number of commodity drives running in multiple RAID configurations. The control unit transforms their interfaces to appear as standard IBM 3390 disk drives, which is the most common disk appearance for mainframes. These disk control units have multiple channel interfaces and can all operate in parallel.

2.7.3 Larger systems

Figure 2-11 shows the z/990, a larger mainframe. The z/990 is usually equipped with more processors than a z/890. In an environment with more than one mainframe, VTAM or TCP/IP of z/VSE can use a *channel-to-channel* (CTC) connection to communicate between those mainframes.

CTC connection
A connection between two CHPIDs on the same or different processors, either directly or through a switch

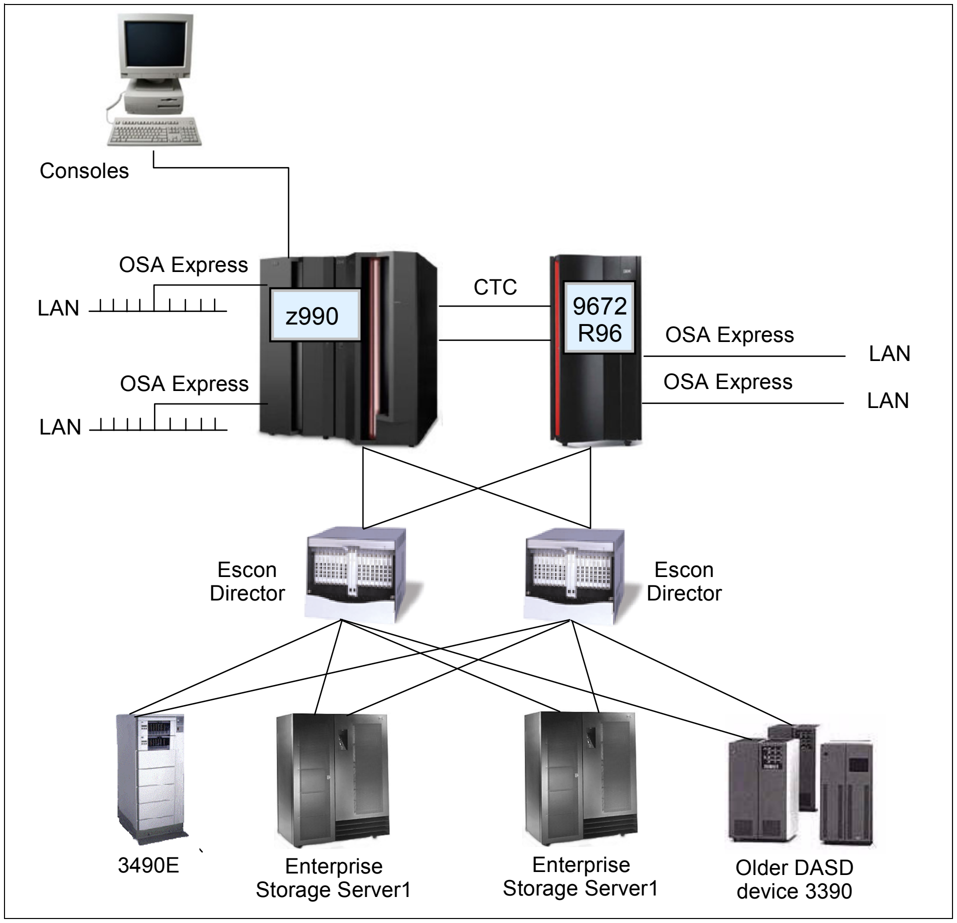


Figure 2-11 Moderately large mainframe configuration

Briefly, the devices in Figure 2-11 include:

- An IBM 3745, which is a communications controller optimized for connection to remote terminals and controllers, and LANs. A 3745 appears as a control unit to the mainframe.
- IBM 3490E tape drives, which, though somewhat outdated, handle the most widely used mainframe-compatible tape cartridges.

- A sixth-generation mainframe design (G6).
- A newer z990 mainframe.
- An Enterprise Storage Server (ESS).
- ESCON directors.
- OSA Express connections to several LANs.

2.8 Summary

Being aware of various meanings of the terms *systems*, *processors*, *CPs*, and so forth is important for your understanding of mainframe computers. The original S/360 architecture, based on CPUs, memory, channels, control units, and devices, and the way these are addressed, is fundamental to understanding mainframe hardware—even though almost every detail of the original design has been changed in various ways. The concepts and terminology of the original design still permeate mainframe descriptions and designs.

zAAP
A specialized processing assist unit configured for running Java programming on selected System z machines

The ability to partition a large system into multiple smaller systems (LPARs) is now a core requirement in practically all mainframe installations. The flexibility of the hardware design, allowing any processor (CP) to access and accept interrupts for any channel, control unit, and device connected to a given LPAR, contributes to the flexibility, reliability, and performance of the complete system. The availability of a pool of processors (PUs) that can be configured (by IBM) as central processors (CPs), I/O processors (SAPs), dedicated Linux processors (IFLs), dedicated Java-type processors (zAAPs[14]), dedicated information processors (zIIPs[14]), and spare processors is unique to mainframes and, again, provides great flexibility in meeting customer requirements. Some of these requirements are based on the cost structures of some mainframe software.

In addition to the primary processors just mentioned (the PUs, and all of their characterizations), mainframes have a network of controllers (special microprocessors) that control the system as a whole. These controllers are not visible to the operating system or application programs.

Since the early 1970s, mainframes have been designed as multiprocessor systems, even when only a single processor is installed. All operating system software is designed for multiple processors. A system with a single processor is considered a special case of a general multiprocessor design. All but the smallest mainframe installations typically use clustering techniques, although they do not normally use the terms *cluster* or *clustering*.

[14] Only supported by z/OS

As stated previously, a clustering technique can be as simple as a shared DASD configuration where manual control or planning is needed to prevent unwanted data overlap. More common today are configurations that allow sharing of locking and enqueueing controls among all systems. Among other benefits, this automatically manages access to data sets so that unwanted concurrent usage does not occur.

Key terms in this chapter		
Central processing complex (CPC)	Central processing unit (CPU)	Channel path identifier (CHPID)
Channel-to-channel (CTC) connection	ESCON channel	FICON channel
Hardware management console (HMC)	I/O connectivity	Integrated facility for Linux (IFL)
Logical partition (LPAR)	Multiprocessor	Power-on reset (POR)
Support element (SE)	Z/Architecture	System z Application Assist Processor (zAAP)

2.9 Questions for review

To help test your understanding of the material in this chapter, complete the following questions:

1. Why does software pricing for mainframes seem so complex?
2. Why does IBM have so many models (or *capacity settings*) in recent mainframe machines?
3. *Multiprocessor* means several processors (and that these processors are used by the operating system and applications). What does *multiprogramming* mean?
4. What z/VSE application changes are needed to work in an LPAR?

2.10 Topics for further discussion

Visit a mainframe installation, if it can be arranged. The range of new, older, and much older systems and devices found in a typical installation is usually interesting and helps to illustrate the sense of continuity that is so important to mainframe customers.

2.11 Exercises

1. To display the CPU configuration:
   a. Access the system console or the Interactive Interface console via fast path 31 in the menu of an administrative user.
   b. On the command line, enter `QUERY TD` and press Enter.
   c. Then enter `SYSDEF TD,START=ALL` and again `QUERY TD` and look on the output. What has changed?
2. To display the DASD and tape devices defined to the system:
   a. On the command line, enter `VOLUME` and press Enter.
   b. Choose one of the devices, which are flagged as USED, and enter `STATUS <device number>`. On the right side of the output you will see the CHPIDs through which the device can be accessed.
3. To display more system information:
   a. Enter `SIR` on the command line and see which device number is the current IPL device.
   b. Enter `SIR CHPID` to display all CHPIDs accessible from this LPAR.

3

z/VSE overview

Objective: As the newest member of your company's mainframe IT group, you will need to know the basic functional characteristics of the mainframe operating system. The operating system taught in this course is z/VSE, an entry mainframe operating system. z/VSE is known for its ability to serve thousands of users concurrently and for processing large workloads in a secure and reliable manner.

After completing this chapter, you will be able to:

- Give examples of how z/VSE differs from a single-user operating system.
- List the major types of storage used by z/VSE.
- Explain the concept of virtual storage and its use in z/VSE.
- State the relationship between pages, frames, and page data set.
- List several defining characteristics of the z/VSE operating system.
- List several software products used with z/VSE to provide a complete system.
- Describe several differences and similarities between the z/VSE and UNIX operating systems.

3.1 What is an operating system?

In simplest terms, an *operating system* is a collection of programs that manage the internal workings of a computer system. Operating systems are designed to make the best use of the computer's various resources and ensure that the maximum amount of work is processed as efficiently as possible. Although an operating system cannot increase the speed of a computer, it can maximize its use, thereby making the computer seem faster by allowing it to do more work in a given period of time.

A computer's *architecture* consists of the functions the computer system provides. The architecture is distinct from the physical design, and, in fact, different machine designs might conform to the same computer architecture. In a sense, the architecture is the computer as seen by the user, such as a system programmer. For example, part of the architecture is the set of machine instructions that the computer can recognize and execute. In the mainframe environment, the system software and hardware comprise a highly advanced computer architecture, the result of decades of technological innovation.

3.2 What is z/VSE?

The operating system we discuss in this course is z/VSE[1], a widely used mainframe operating system. z/VSE is designed to offer a stable, secure, and continuously available environment for applications running on the mainframe.

z/VSE today is the result of four decades of technological advancement. z/VSE evolved from a simple operating system that could process a single program at a time to a sophisticated operating system that can handle many programs and interactive users concurrently. To understand how and why z/VSE functions as it does, it is important to understand some basic concepts about z/VSE and the environment in which it functions. This chapter introduces some of the concepts that you will need to understand the z/VSE operating system.

In most early operating systems, requests for work entered the system one at a time. The operating system processed each request or *job* as a unit, and did not start the next job until the one ahead of it had completed. This arrangement worked well when a job could execute continuously from start to completion. But often a job had to wait for information to be read in from, or written out to a device such as a tape drive or a printer. Input and output (I/O) takes a long time compared to the electronic speed of the processor. When a job waited for I/O, the processor was idle.

[1] z/VSE is designed to take advantage of the IBM ESA/390 architecture or z/Architecture.

Finding a way to keep the processor working while a job waited would increase the total amount of work the processor could do without requiring additional hardware. z/VSE gets work done by dividing it into pieces and giving portions of the job to various system components and subsystems that function interdependently. At any point in time, one component or another gets control of the processor, makes its contribution, and then passes control along to a user program or another component.

The next section describes these steps of evolution in more details.

3.2.1 History of z/VSE

The first *VSE* was Disk Operating System/360 (DOS/360). It was developed in 1964 as an alternative operating system to the predecessor of z/OS designed for smaller members of the S/360 hardware family. DOS/360 was first shipped in 1965. It began with a single partition[2], but quickly grew to three for basic multiprogramming. Later, BTAM (basic telecommunications access method) added primitive telecommunications.

In 1965, an imaginary DOS/360 customer might have an S/360-30 system with 32K bytes of main memory.

In 1970, System/370 followed. Virtual storage was added in 1972. Virtual storage expanded system capacity and made programming easier and more productive. Real memory options for the S/370-135 ranged from 96K to 256K bytes.

After 27 releases, DOS/360 became DOS/VS. DOS/VS offered five partitions (later 7) and a relocating loader for effective multiprogramming. Priority Output Writers, Execution Processors, and Input Readers (POWER) was added for I/O spooling[3]. A new virtual storage access method (VSAM[4]) file system for balanced random and sequential processing became part of DOS/VS. Database/Data Communication (DBDC) became a fundamental part of VSE as the use of CICS grew. A hierarchical database known as DL/1 (data language 1) was available as well. At this time, DOS/VS became something we would clearly recognize today as a VSE system.

In 1979, IBM introduced the IBM 4300 system. Real memory options ranged from 512K to 4,096K (4 MB) bytes. Along with the 4300, IBM introduced new disk systems based on Fixed Block Architecture (FBA).

[2] A partition is a division of the virtual address area available for running programs. See also 3.4.2, “What is an address space?” on page 76.
[3] For details see Chapter 7, “Batch processing and VSE/POWER” on page 177.
[4] For more information see 5.5, “What is VSAM?” on page 145.

DOS/VS became DOS/VSE. E stood for *extended*. DOS/VSE offered up to 12 partitions. MSHP[5] (maintain system history program) was added to enhance service and control of the system. ICCF[6] (interactive computing and control facility) became the interactive component. DOS/VSE also offered improvements such as ASI[7] procedures, DASD sharing (locking across z/VSE systems), and so on. ACF/VTAM[8] (advanced communications function for virtual telecommunications access method) became a component of VSE. A major extension was support for FBA disk devices.

In 1984, DOS/VSE transformed into VSE/Systems Package (VSE/SP). VSE/SP consisted of an integrated, pre-packaged VSE system. VSE/SP V1 and V2 refined the concept and made it more generally acceptable. VSE/SP was a major step forward in terms of usability. The Interactive Interface became an integral part of VSE. It provides dialogs for installation, administration, and operation, as shown in Chapter 4, "The Interactive Interface of z/VSE" on page 111. The Fast Service Upgrade (FSU) process made release-to-release migration simpler.

Around this time the main memory ranged from 4 to 16 MB.

By 1987, VSE/SP V3 implemented a packaging concept that remains visible to this day in z/VSE. The structure consists of base and optional products. The *base* is an integrated package containing key, commonly used products. It is designed, developed, tested, delivered, and serviced as an integrated whole. *Optional* products are coordinated so that the list contains the correct level of each product. Installation procedures and dialogs are provided. Following an order, base and optional products are stacked together on the delivery media. Service is coordinated for both base and optional products, and appropriate prerequisite and co-requisite levels and service are identified.

VSE/SP V3 included a new library structure (the VSE librarian[9]), conditional job control language (JCL), and Virtual Address Extensions (VAE). VAE was an attempt to extend the capacity of VSE. It provided up to three virtual address spaces[10], each with no more that 16 MB. Each address space contained a shared space, common to all address spaces, and the private space, unique to the address space[11].

During the 1980s, some customer applications began to bump up against the limitations of the S/370 24-bit (16 MB) architecture. S/370-XA (eXtended

[5] For more information see 15.2, "Customizing the system" on page 373.
[6] For details see Chapter 4, "The Interactive Interface of z/VSE" on page 111.
[7] Automated System Initialization (ASI). See also 15.3.3, "IPL and system startup" on page 380.
[8] For more information about VTAM see Chapter 17, "Network communications on z/VSE" on page 403.
[9] See Chapter 5, "Working with files" on page 139.
[10] This is described in 3.4, "Virtual storage and other mainframe concepts" on page 75
[11] Figure 3-9 on page 89 shows the storage map for z/VSE.

Architecture) extensions added 31-bit (2 GB) real and virtual addressing in response to customer needs. Because VSE/SP did not implement 370-XA architecture, it began to look as if VSE might be left behind.

In September 1990 the IBM Enterprise System/9000 (ES/9000) was announced. ES/9000 processors implemented Enterprise Systems Architecture (ESA), an extension of 370-XA. There main memory ranged from 256 MB to 9 GB.

Also in 1990, VSE/SP became VSE/ESA Version 1. VSE/ESA V1 kept the best parts of VSE/SP and focused on quality, capacity, and MVS *affinity*.

For increased capacity, VSE/ESA V1 first implemented 31-bit for real memory, then added 31-bit virtual addressing. VSE/ESA V1 offered dynamic partitions. Dynamic partitions are created just for a job in execution. The former partitions (as known from prior releases) are called static partitions. These partitions are usually created after IPL. In later releases, VSE/ESA V1 added support for ESA data spaces and virtual disk in storage that can be used for temporary data. VSE/ESA V1 exploited ESA access registers. New versions of CICS/VSE, ACF/VTAM, and VS COBOL II were added for greater z/OS affinity[12].

In 1994 new processors were introduced, known as 9672 G1. VSE/ESA V2 replaced V1 and introduced the turbo dispatcher to support those processors. For the first time VSE supported n-way servers. VSE/ESA V2 also introduced Language Environment technology[13] and newer levels of COBOL, PL/I, and C for increased z/OS affinity.

VSE/ESA V2.2 shipped in late 1996. It was the first VSE to be *Year 2000* ready.

The Year 2000 problem had its origins in the early days of S/360 when memory was a precious resource and 2000 seemed far in the future. Widespread practice was to store and process only the last two digits of the year. For example, 1972 was stored as 72, 1996 was 96, and 2000 was 00. As 2000 approached, systems and business applications were exposed. For example, subtracting 72 from 96, the correct answer is 24. However, subtracting 72 from 00, the answer is negative 72 (mathematically OK, but incorrect for its intended use). This simple, seemingly trivial mistake had the potential to seriously disrupt both the financial integrity and the operational effectiveness of many companies.

In 1997, VSE/ESA V2.3 introduced a native TCP/IP (licensed from Connectivity Systems Incorporated) implementation.

In 1999, VSE/ESA V2.4 launched CICS Transaction Server for VSE/ESA (CICS TS) (though CICS/VSE continued to ship along with CICS TS VSE/ESA for compatibility reasons). Because CICS TS was ported from OS/390, it

[12] z/OS is used here as a synonym for MVS, OS/390, or z/OS.
[13] For details see Chapter 9, “Using programming languages on z/VSE” on page 217.

represented a major expansion of z/OS affinity. As a preparation for the CICS TS port, more than 60 MVS (OS/390) interfaces were ported (for example, GETMAIN, FREEMAIN, ATTACHX, WAIT, and so on). In addition, cross memory services were implemented. Those services allow execution of programs in different address spaces. CICS TS exploits that capability.

In 2000, VSE/ESA 2.5 introduced VSE Connectors to allow access from Java applications to VSE resources, such as VSAM, DL/1, VSE/POWER, and so on.

Since 2000 the connectivity to VSE and non-VSE systems has improved. The VSE team introduced new connectors, added SOAP support, and exploited System z specific hardware features such as Hipersockets, hardware crypto capabilities, and so on.

In 2005, VSE/ESA V2 became z/VSE V3. Like z/VM and Linux on System z, z/VSE V3.1 supports FCP-SCSI disks. z/VSE can execute in 31-bit mode only. It does not implement z/Architecture, and specifically does not implement 64-bit mode capabilities.

In January 2007, IBM announced z/VSE V4.1. z/VSE 4.1 executes in z/Architecture mode only (requires System z processors) and uses 64 bit real addressing to support up to 8 GB of processor storage. This will be transparent to application programs. z/VSE V4.1 will not support 64 bit virtual addressing.

3.2.2 Hardware resources used by z/VSE

The z/VSE operating system executes in a processor and resides in processor storage during execution. z/VSE is commonly referred to as the *system* software.

Mainframe hardware consists of processors and a multitude of peripheral devices such as disk drives (also called direct access storage devices, or DASD), magnetic tape drives, and various types of user consoles. See Figure 3-1. Tape and DASD are used for system functions and by user programs executed by z/VSE.

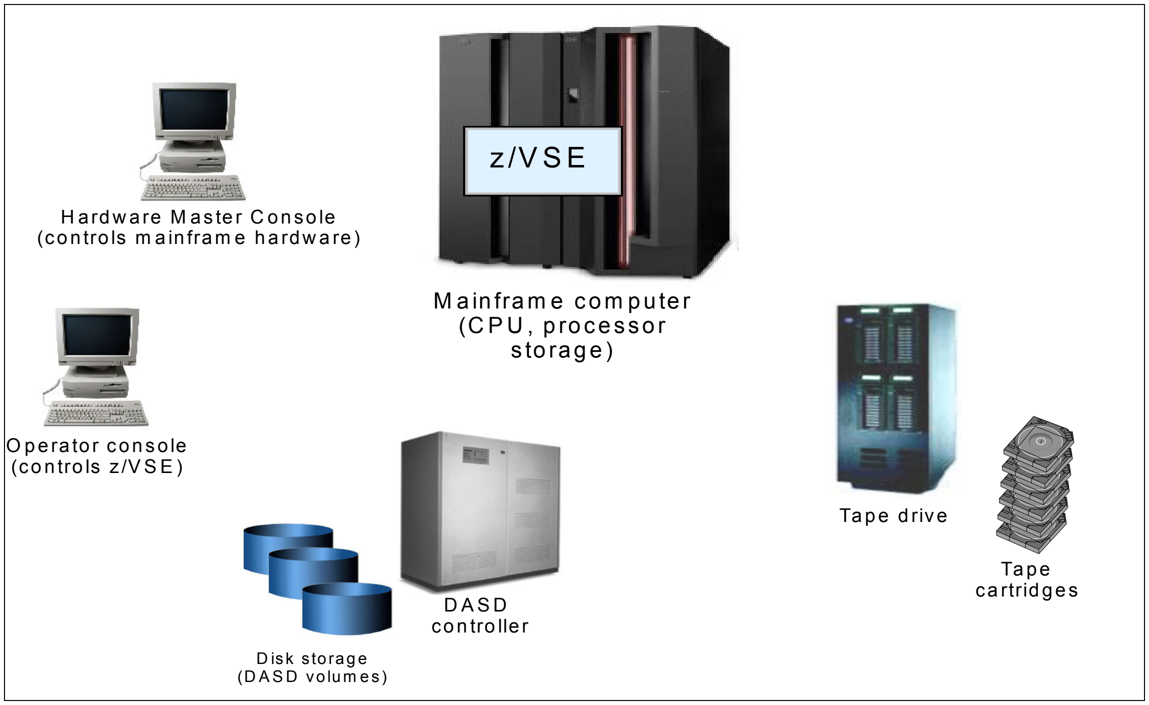


Figure 3-1   Hardware resources used by z/VSE

To fulfill a new order for a z/VSE system, IBM ships the system code to the customer through the Internet or (depending on customer preference) on physical tape cartridges or CD-ROM. At the customer site, a person such as the z/VSE system programmer receives the order and starts the VSE installation process, which copies the new system to disk volumes. During the installation process, customization, and operation, system consoles are required to interact with the z/VSE system.

The z/VSE releases address the latest IBM mainframe hardware and its many sophisticated peripheral devices. Figure 3-1 presents a simplified view of mainframe concepts that students will build upon throughout this course:

- Software - The z/VSE operating system consists of phases, which are *executable code.* During the install process, the installation process copies these phases to *VSE libraries* residing on disk volumes.

- Hardware - The system hardware consists of all the devices, controllers, and processors that constitute a mainframe environment:
  - Peripheral devices - These include tape drives, disks, and consoles. There are many other types of devices, some of which are discussed in Chapter 2, “Mainframe hardware systems” on page 35.
  - Processor storage - Often called real or central storage (or memory), this is where the z/VSE operating system executes. Also, all user programs share the use of processor storage with the operating system.

As a *big picture* of a typical mainframe hardware configuration, Figure 3-1 on page 69 is far from complete. Not shown, for example, are the hardware control units that connect the mainframe to the other tape drives, DASD, and consoles. Also not shown are connections to the network.

Related reading: The standard reference for descriptions of the major facilities of z/Architecture is the IBM publication *z/Architecture Principles of Operation*. You can find this and other related publications at the z/VSE Internet Library Web site:

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

3.2.3 Multiprogramming and multiprocessing

Multi-programming Executing many programs concurrently.

The earliest operating systems were used to control single-user computer systems. In those days, the operating system would read in one job, find the data and devices the job needed, let the job run to completion, and then read in another job. In contrast, the computer systems that z/VSE manages today are capable of *multiprogramming,* or executing many programs concurrently. With multiprogramming, when a job cannot use the processor (for example, when it is waiting for the completion of an input/output operation), the system can suspend, or *interrupt*, the job, freeing the processor to work on another job.

z/VSE makes multiprogramming possible by capturing and saving all of the relevant information about the interrupted program before allowing another program to execute. When the interrupted program is ready to begin executing again, it can resume execution just where it left off. Multiprogramming allows z/VSE to run lots of programs simultaneously for users who might be working on different projects at different physical locations around the world.

Multi-processing The simultaneous operation of two or more processors that share the various hardware resources.

z/VSE can also perform *multiprocessing,* which is the simultaneous operation of two or more processors that share the various hardware resources, such as memory and external disk storage devices. The techniques of multiprogramming and multiprocessing make z/VSE ideally suited for processing workloads that require many input/output (I/O) operations. Mainframe workloads often include long-running applications that write updates to millions of records in a database, and online applications for thousands of interactive users at any given time. By way of contrast, consider the operating system that might be used for a single-user computer system. Such an operating system would need to execute programs on behalf of one user only. In the case of a personal computer (PC), for example, the entire resources of the machine are often at the disposal of one user.

Many users running many separate programs means that z/VSE users need large amounts of memory to ensure suitable system performance. Even small and medium sized companies run sophisticated business applications that access large databases and industry-strength middleware products. Such applications require the operating system to protect privacy among users, as well as enable the sharing of databases and software services.

Thus, multiprogramming, multiprocessing, and the need for a large amount of memory means that z/VSE must provide function beyond simple, single-user applications. The sections that follow explain, in a general way, the attributes that enable z/VSE to manage complex computer configurations. Subsequent portions of this text explore these features in more detail.

3.2.4 Phases, modules, and macros

z/VSE is made up of programming instructions that control the operation of the computer system. These instructions ensure that the computer hardware is used efficiently and allows application programs to run. z/VSE includes sets of instructions that, for example, accept work, convert work to a form that the computer can recognize, keep track of work, allocate resources for work, execute work, monitor work, and handle output. A group of related instructions is called a *routine* or *module*. One or more modules can build a *phase*. Only a phase can be loaded into the storage for execution. A set of related phases that make a particular system function possible is called a *system component*. The VSE/POWER component of z/VSE, for instance, starts jobs and manages printed output.

Sequences of instructions that perform frequently used system functions can be invoked with executable macro instructions, or *system macros*. z/VSE system macros exist for functions such as opening and closing data files, loading and deleting programs, and sending messages to the computer operator.

3.2.5 Control blocks

As system programs execute the work of a z/VSE system, they may keep track of this work in storage areas known as *control blocks*. In general, there are five types of z/VSE control blocks:

- System-related control blocks
- Resource-related control blocks
- Address space-related control blocks
- Partition-related control blocks
- Task-related control blocks

The system-related control blocks represent one z/VSE system and contain system-wide information, such as how many processors are in use. Each resource-related control block represents one resource, such as a processor or storage device. Each address space-related control block represents one address space. Each partition-related control block represents one partition on the system where tasks can run. Each task-related control block represents one task.

Control block A data structure that serves as a vehicle for communication in z/VSE

Control blocks serve as vehicles for communication throughout z/VSE. Such communication is possible because the structure of a control block is known to the system programs that use it, and thus these system programs can find needed information about the task or resource. Control blocks representing many units of the same type may be chained together on queues, with each control block pointing to the next one in the chain. The operating system can search the queue to find information about a particular unit of work or resource, which might be:

- An address of a control block or a required routine
- Actual data, such as a value, a quantity, a parameter, or a name
- Status flags (usually single bits in a byte, where each bit has a specific meaning)

z/VSE uses a variety of control blocks, many with very specialized purposes. This chapter mentions five of the most important control blocks:

- Task control block (TCB) together with the task information block (TIB), which represent a task.
- Partition control block (PCB) together with the partition communication region (COMREG), which represent a partition.
- System communication region (SYSCOM), which represents the whole system.

3.2.6 Physical storage used by z/VSE

Central storage
Physical storage on the processor

Conceptually, mainframes and all other computers have two types of physical storage[14]:

- The physical storage located on the mainframe processor itself. This is called processor storage, real storage, or *central storage*. Think of it as *memory* for the mainframe.
- The physical storage external to the mainframe, including storage on direct access devices, such as disk drives. This storage is called paging storage or *auxiliary storage.*

The primary difference between the two kinds of storage relates to the way in which it is accessed, as follows:

- Central storage is accessed synchronously with the processor. That is, the processor must wait while data is retrieved from central storage[15].
- Auxiliary storage is accessed asynchronously. The processor accesses auxiliary storage through an input/output (I/O) request, which is scheduled to run amid other work requests in the system. During an I/O request, the processor is free to execute other, unrelated work.

Auxiliary storage
Physical storage external to the mainframe, including storage on direct access devices, such as disk drives

As with memory for a personal computer, mainframe central storage is tightly coupled with the processor itself, whereas mainframe auxiliary storage is located on (comparatively) slower, external disk drives. Because central storage is more closely integrated with the processor, it takes the processor much less time to access data from central storage than from auxiliary storage. Auxiliary storage, however, is less expensive than central storage.

3.3 Overview of z/VSE facilities

An extensive set of system facilities and unique attributes makes z/VSE well suited for processing large, complex workloads, such as those that require many I/O operations, access to large amounts of data, or comprehensive security. Typical mainframe workloads include long-running applications that update millions of records in a database, and online applications that can serve many thousands of users concurrently.

[14] Many computers also have a fast memory, local to the processor, called the processor cache. The cache is not visible to the programmer or application programs, or even to the operating system directly. This is beyond the scope of this book.
[15] Some processor implementations use techniques such as instruction or data pre-fetching or *pipelining* to enhance performance. These techniques are not visible to the application program or even the operating system, but a sophisticated compiler can organize the code it produces to allow the best use of these techniques.

Figure 3-2 provides a snapshot view of the z/VSE operating environment.

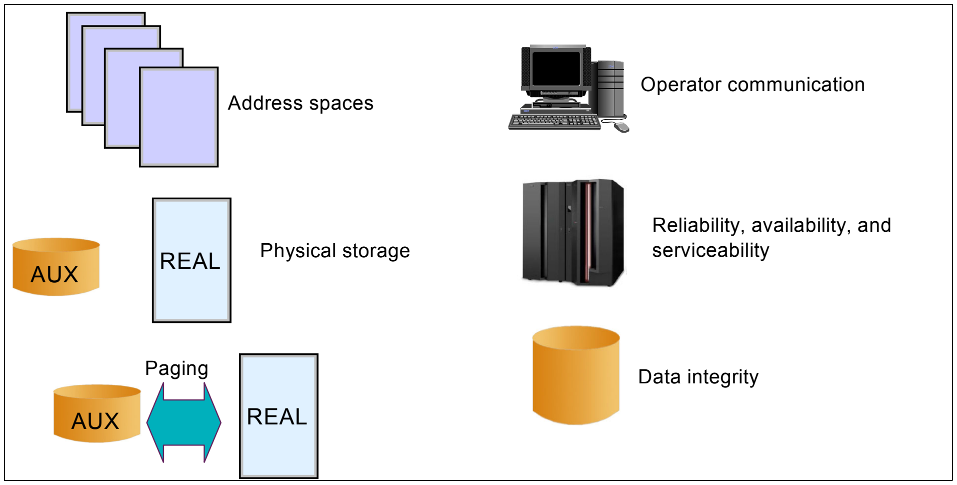


Figure 3-2 z/VSE operating environment

These facilities are explored in greater depth in the remaining portions of this text, but are summarized here:

- An address space describes the virtual storage addressing range available to an online user or a running program.
- Two types of physical storage are available: central storage and auxiliary storage (AUX). Central storage is also referred to as *real storage* or *real memory*.
- z/VSE moves programs and data between central storage and auxiliary storage through processes called paging.
- z/VSE dispatches work for execution (not shown in the figure). That is, it selects programs to be run based on priority and ability to execute and then restores the program's status. All program instructions and data must be in central storage when executing.
- An extensive set of facilities manages files stored on direct access storage devices (disks) or tape cartridges.
- System operators use consoles to start and stop z/VSE, enter commands, and manage the operating system.

z/VSE is further defined by many other operational characteristics, such as security, recovery, and data integrity.

3.4 Virtual storage and other mainframe concepts

z/VSE uses both types of physical storage (real and auxiliary) to enable another kind of storage called *virtual storage*. In z/VSE, each user has access to virtual storage, rather than physical storage. This use of virtual storage is central to the ability of z/VSE to interact with large numbers of users concurrently, while processing large workloads.

3.4.1 What is virtual storage?

Virtual storage means that each running program can assume that it has access to all of the storage defined by the architecture's addressing scheme. The only limit is the number of bits in a storage address. This ability to use a large number of storage locations is important because a program may be long and complex, and both the program's code and the data it requires must be in central storage for the processor to access them.

The z/Architecture supports 64-bit long addresses, which allows you to address up to 18,446,744,073,709,600,000 bytes (16 exabytes) of storage locations. z/VSE uses 64-bit addresses only to address real storage. For user applications, z/VSE supports 31-bit long addresses, which allows a program to address up to 2,147,483,648 bytes (2 gigabytes) of storage locations. In reality, the mainframe might have a few gigabytes of central storage installed. Much depends on the model of computer and the system configuration.

To allow each user to act as though this much storage really exists in the computer system, z/VSE keeps only the active portions of each program in central storage. z/VSE keeps the rest of the code and data in files called *page data sets* on auxiliary storage, which consists of high-speed direct access storage devices (disks).

Virtual storage, then, is this combination of real and auxiliary storage. z/VSE uses a system of tables and indexes to relate locations on auxiliary storage to locations in central storage. It uses special settings (bit settings) to keep track of the identity and authority of each user or program. z/VSE uses a variety of storage manager components to manage virtual storage. This chapter briefly covers the key points in the process.

This process is shown in more detail in 3.4.4, "Virtual storage overview" on page 78.

Terms: Mainframe workers use the terms *central storage*, *real memory*, *real storage*, and *main storage* interchangeably. Likewise, they use the terms *virtual memory* and *virtual storage* synonymously.

3.4.2 What is an address space?

The range of virtual addresses that the operating system assigns to a user or separately running program is called an *address space*. This is the area of contiguous virtual addresses available for executing instructions and storing data. The range of virtual addresses in an address space starts at zero and can extend to the highest address permitted by the operating system architecture.

Address space
The range of virtual addresses that the operating system assigns to a user or program

However, the use of multiple virtual address spaces in z/VSE holds some special advantages. Virtual addressing permits an addressing range that may be greater than the central storage capabilities of the system. The use of multiple virtual address spaces provides this virtual addressing capability to each job in the system by assigning each job its own separate virtual address space. The potentially large number of address spaces provides the system with a large virtual addressing capacity.

With multiple virtual address spaces, errors are confined to one address space, except for errors in commonly addressable storage, thus improving system reliability and making error recovery easier. Programs in separate address spaces are protected from each other. Isolating data in its own address space also protects the data.

Independent of the importance of address spaces in z/VSE, in the daily business partitions are more visible than the address spaces. A *partition* is a division of an address space available for running programs. Within each partition the user can start multiple tasks. For each task, the system uses *task control blocks* (TCBs) that allow multiprogramming.

Partition
A division of the virtual address area available for running programs

z/VSE knows static partitions and dynamic partitions:

- Static partitions

  There are 12 static partitions: BG, F1 to F9, FA and FB. The static partitions and their static address spaces are allocated during system startup. They exist until the shutdown of the system. A static address space can contain more than one static partition. Normally there is only one per address space.

- Dynamic partitions

  Dynamic partitions and their address spaces are allocated by VSE/POWER for the lifetime of a power job. There can be only one dynamic partition in an address space.

An active z/VSE system uses many address spaces and partitions. There is usually one address space for each job in progress. Online users are using the address space of the corresponding subsystem like CICS, not their own address spaces. There may also be address spaces for operating system functions, such as security, and so on.

Address space isolation

The use of address spaces allows z/VSE to maintain the distinction between the programs and data belonging to each address space. The private areas within one user's address space are isolated from the private areas within other address spaces, and this provides much of the operating system's security.

Yet, each address space also contains a common area that is accessible to every other address space. Because it maps all of the available addresses, an address space includes system code and data as well as user code and data. Thus, not all of the mapped addresses are available for user code and data.

The ability of many users to share the same resources implies the need to protect users from one another and to protect the operating system itself. Along with such methods as *keys* for protecting central storage and code words for protecting data files and programs, separate address spaces ensure that users' programs and data do not overlap.

Address space communication

In a multiple virtual address space environment, applications need ways to communicate between address spaces. z/VSE provides methods of inter-address space communication using cross-memory application programming interfaces (APIs) and access registers.

A program uses cross-memory APIs to access another user's address spaces directly. You might compare z/VSE cross-memory APIs to the UNIX Shared Memory functions. z/VSE cross-memory APIs allow efficient and secure ways to execute programs in other address spaces to access data owned by others, data owned by the user but stored in another address space for convenience, and for rapid and secure communication with services like transaction managers and database managers.

3.4.3 What is dynamic address translation?

Dynamic address translation (DAT) is the process of translating a virtual address during a storage reference into the corresponding real address. If the virtual address is already in central storage, the DAT process may be accelerated through the use of a translation lookaside buffer. If the virtual address is not in central storage, a page fault interrupt occurs, z/VSE supervisor is notified, and z/VSE brings the page into central storage either for the first time (there is no copy on auxiliary storage) or from auxiliary storage.

DAT is implemented by both hardware and software through the use of page tables, segment tables, and translation lookaside buffers.

3.4.4 Virtual storage overview

Recall that for the processor to execute a program instruction—both the instruction and the data it references—must be in central storage. The convention of early operating systems was to have the entire program reside in central storage when its instructions were executing. However, the entire program does not really need to be in central storage when an instruction executes. Instead, by bringing pieces of the program into central storage only when the processor is ready to execute them—moving them out to auxiliary storage when it does not need them, an operating system can execute more and larger programs concurrently.

How does the operating system keep track of each program piece? How does it know whether it is in central storage or auxiliary storage, and where? It is important for z/VSE professionals to understand how the operating system makes this happen.

Physical storage is divided into areas, each the same size and accessible by a unique address. In central storage, these areas are called *frames*. In auxiliary storage, they are called *slots.* Similarly, the operating system can divide a program into pieces the size of frames or slots and assign each piece a unique address. This arrangement allows the operating system to keep track of these pieces. In z/VSE, the program pieces are called *pages*. These areas are discussed further in "Frames, pages, and slots" on page 80.

Pages are referenced by their virtual addresses and not by their real addresses. From the time a program enters the system until it completes, the virtual address of the page remains the same, regardless of whether the page is in central storage or auxiliary storage. Each page consists of individual locations called bytes, each of which has a unique virtual address.

Format of a virtual address

As mentioned, virtual storage is an illusion created by the architecture, in that the system seems to have more memory than it really has. Each program gets an address space, and each address space may contain the same range of storage addresses. Only those portions of the address space that are needed at any one point in time are actually loaded into central storage. z/VSE keeps the inactive pieces of address spaces in auxiliary storage. z/VSE manages address spaces in units of various sizes according to the following architecture:

Page	Address spaces are divided into 4-kilobyte units of virtual storage called pages.
Segment	Address spaces are divided into 1-megabyte units called segments. A segment is a block of sequential virtual addresses spanning megabytes, beginning at a

1-megabyte boundary. A 2-gigabyte address space, for example, consists of 2048 segments.

A virtual address, accordingly, is divided into three fields. Bits 1–11 are called the segment index (SX), bits 12–19 are called the page index (PX), and bits 20–31 are called the byte index (BX).

The virtual address has the format shown in Figure 3-3.

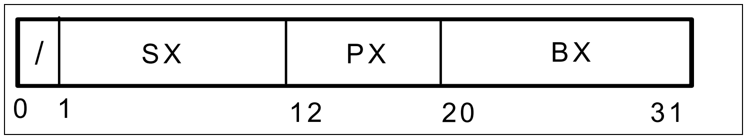


Figure 3-3 Format of virtual address

How virtual storage addressing works in z/VSE

As stated previously, the use of virtual storage in z/VSE means that only the pieces of a program that are currently active need to be in central storage at processing time. The inactive pieces are held in auxiliary storage. Figure 3-4 shows the virtual storage concept at work in z/VSE.

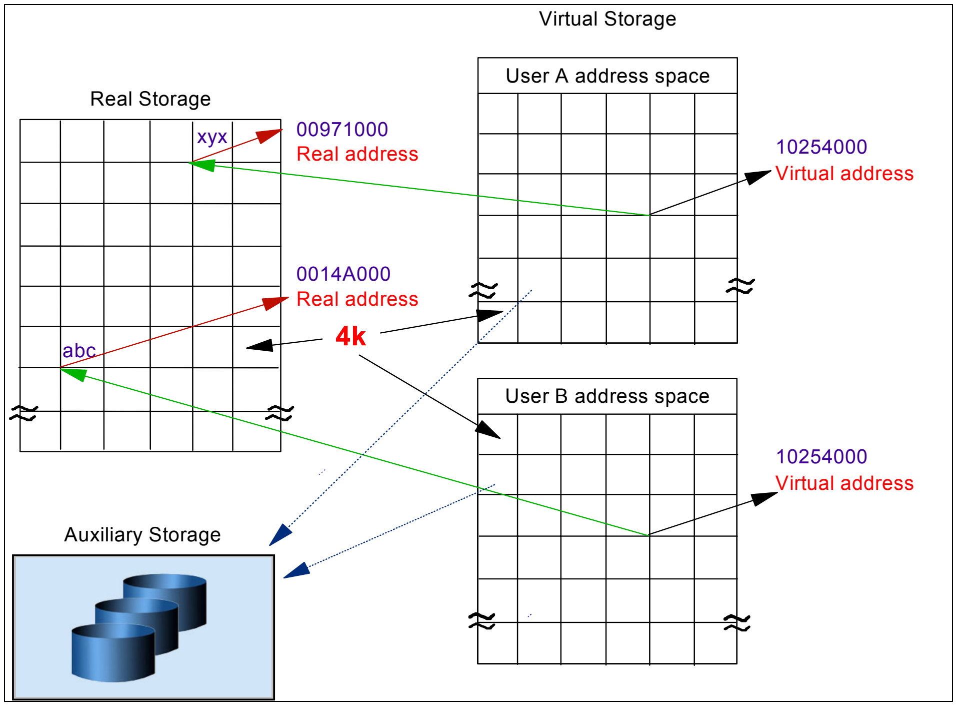


Figure 3-4 Real and auxiliary storage combine to create the illusion of virtual storage

In Figure 3-4 on page 79, observe the following:

- An address is an identifier of a required piece of information, but not a description of where in central storage that piece of information is. This allows the size of an address space (that is, all addresses available to a program) to exceed the amount of central storage available.
- For most user programs, all central storage references are made in terms of virtual storage addresses.[16]
- Dynamic address translation (DAT) is used to translate a virtual address during a storage reference into a physical location in central storage. As shown in Figure 3-4 on page 79, the virtual address 10254000 can exist more than once, because each virtual address maps to a different address in central storage.
- When a requested address is not in central storage, a hardware interruption is signaled to z/VSE, and the operating system pages in the required instructions and data to central storage.

Frames, pages, and slots

When a program is selected for execution, the system brings it into virtual storage, divides it into pages of four kilobytes, and transfers the pages into central storage for execution. To the programmer, the entire program appears to occupy contiguous space in storage at all times. Actually, not all pages of a program are necessarily in central storage, and the pages that *are* in central storage do not necessarily occupy contiguous space.

The pieces of a program executing in virtual storage must be moved between real and auxiliary storage. To allow this, z/VSE manages storage in units, or *blocks*, of four kilobytes. The following blocks are defined:

Frame
In central storage, areas of equal size and accessible by a unique address

- A block of central storage is a *frame*.
- A block of virtual storage is a *page*.
- A block of auxiliary storage is a *slot*.

[16] Some instructions, primarily those used by operating system programs, require real addresses.

A page, a frame, and a slot are all the same size: four kilobytes. An active virtual storage page resides in a central storage frame. A virtual storage page that becomes inactive may reside in an auxiliary storage slot (in a paging data set). Figure 3-5 shows the relationship of pages, frames, and slots.

Slot
In auxiliary storage, areas of equal size and accessible by a unique address

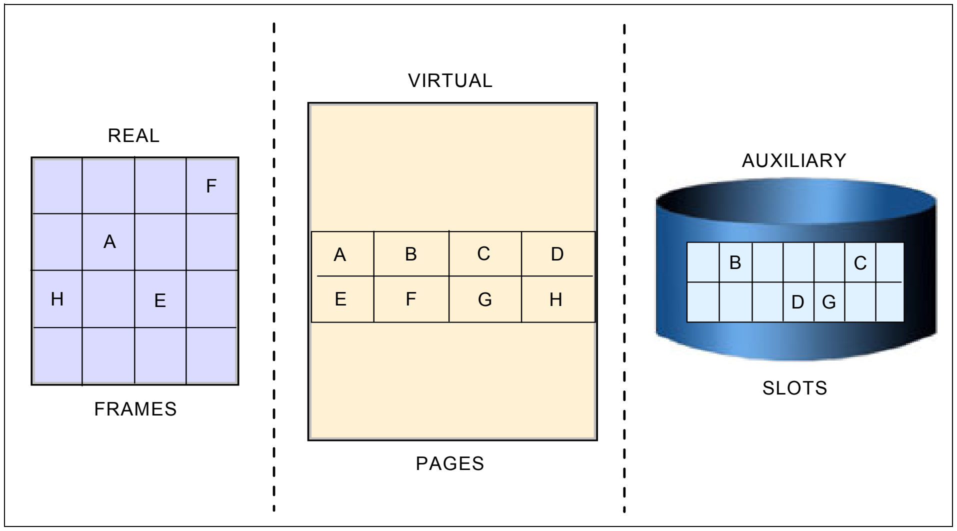


Figure 3-5 Frames, pages, and slots

In Figure 3-5, z/VSE is performing paging for a program running in virtual storage. The lettered boxes represent parts of the program. In this simplified view, program parts A, E, F, and H are active and running in central storage frames, while parts B, C, D, and G are inactive and have been moved to auxiliary storage slots. All of the program parts, however, reside in virtual storage and have virtual storage addresses.

3.4.5 What is paging?

As stated previously, z/VSE uses a series of tables to determine whether a page is in real or auxiliary storage, and where. To find a page of a program, z/VSE checks the table for the virtual address of the page, rather than searching through all physical storage for it. z/VSE then transfers the page into central storage or out to auxiliary storage as needed. This movement of pages between auxiliary storage slots and central storage frames is called paging. Paging is key to understanding the use of virtual storage in z/VSE.

z/VSE paging is transparent to the user. During job execution, only those pieces of the application that are required are brought in, or paged in, to central storage. The pages remain in central storage until no longer needed, or until another page is required by the same application or a higher-priority application and no empty central storage is available. To select pages for paging out to auxiliary storage, z/VSE follows a *least used* algorithm. That is, z/VSE assumes that a page that has not been used for some time will probably not be used in the near future.

How paging works in z/VSE

In addition to the DAT hardware and the segment and page tables required for address translation, paging activity involves a number of system components to handle the movement of pages, and several additional tables to keep track of the most current version of each page.

To understand how paging works, assume that DAT encounters an invalid page table entry during address translation, indicating that a page is required that is not in a central storage frame. To resolve this page fault, the system must bring the page in from auxiliary storage. First, however, it must locate an available central storage frame. If none are available, the request must be saved and an assigned frame freed. To free a frame, the system copies its contents to auxiliary storage and marks its corresponding page table entry as invalid. This operation is called a page-out.

After a frame is located for the required page, the contents of the page are copied from auxiliary storage to central storage and the page table invalid bit is set off. This operation is called a page-in.

Paging can also take place when z/VSE loads an entire program into virtual storage. z/VSE obtains virtual storage for the user program and allocates a central storage frame to each page. Each page is then active and subject to the normal paging activity (that is, the most active pages are retained in central storage while the pages not currently active might be paged out to auxiliary storage).

Page stealing

z/VSE tries to keep an adequate supply of available central storage frames on hand. When a program refers to a page that is not in central storage, z/VSE uses a central storage page frame from a supply of available frames.

When this supply becomes low, z/VSE reuses page frames (that is, it takes a frame assigned to an active user and makes it available for other work). The decision to steal a particular page is based on the activity history of each page currently residing in a central storage frame. Pages that have not been active for a relatively long time are good candidates for reuse.

Page selection

z/VSE uses a queuing algorithm to manage virtual storage based on which pages were most recently used.

All page frames assigned to virtual pages are queued in the page selection queue (PSQ). New pages are queued at the end. The position in the queue indicates how long relatively it has been since a program referenced the page.

When a page frame is needed from this queue, the system scans the PSQ from the beginning and checks the reference bit of a certain number of page frames. If the reference bit is on, it is reset and the page frame is requeued at the end of the PSQ.

The first page frame found with change bit and reference bit off is the one used for replacement. It is the *oldest* inactive page frame, because it has moved from the end of the PSQ to the bottom without having been referenced.

3.4.6 What is storage protection?

Up to now, we have discussed virtual storage mostly in the context of a single user or program. In reality, of course, many programs and users would be competing for the use of the system. z/VSE uses the following techniques to preserve the integrity of each user's work:

- A private address space for each user
- Page protection
- Multiple storage protect keys, as described in this section.

How storage protect keys are used

Under z/VSE, the information in central storage is protected from unauthorized use by means of multiple storage protect keys. A control field in storage called a key is associated with each 4K frame of central storage.

When a request is made to modify the contents of a central storage location, the key associated with the request is compared to the storage protect key. If the keys match or the program is executing in key 0, the request is satisfied. If the key associated with the request does not match the storage key, the system rejects the request and issues a program exception interruption.

When a request is made to read (or fetch) the contents of a central storage location, the request is automatically satisfied unless the fetch protect bit is on, indicating that the frame is fetch-protected. When a request is made to access the contents of a fetch-protected central storage location, the key in storage is compared to the key associated with the request. If the keys match, or the requestor is in key 0, the request is satisfied. If the keys do not match, and the requestor is not in key 0, the system rejects the request and issues a program exception interruption.

How storage protect keys are assigned

z/VSE uses 16 storage protect keys. A specific key is assigned according to the type of work being performed. As Figure 3-5 shows, the key is stored in bits 8 through 11 of the program status word (PSW). A PSW is assigned to each job in the system whenever it starts executing. The PSW shows the current program status and the address of the next instruction to be executed. It also holds the interrupt information and the storage protection key.

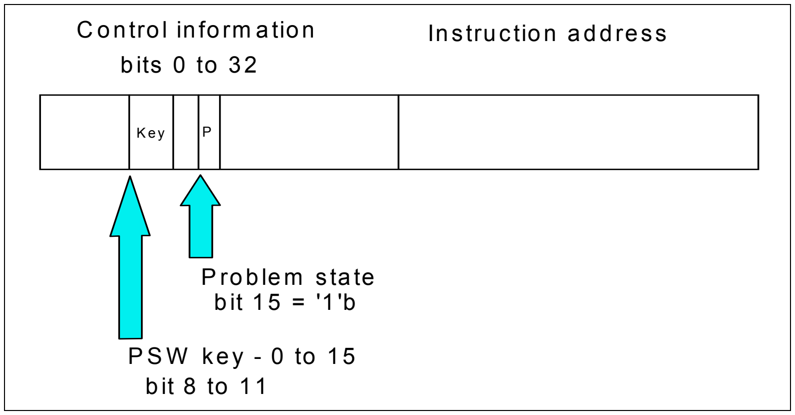


Figure 3-6 Location of storage protect key

Storage protect keys 0 is used by the z/VSE control program (supervisor) and various subsystems and middleware products. Storage protect key 0 is the master key. In almost any situation, a storage protect key of 0 associated with a request to access or modify the contents of a central storage location means that the request is satisfied.

Storage protect keys 1 through 15 are assigned to users. z/VSE assigns keys 1 through 12 to static partitions, and key 13 to dynamic partitions. Users can have the same storage protect key because they are isolated in private address spaces.

3.4.7 Storage management in z/VSE

Central storage frames and auxiliary storage slots, and the virtual storage pages that they support, are managed by separate components of z/VSE. These components are known as the page manager and the virtual storage management services. Here we describe the role of each briefly.

Page manager

The page manager keeps track of the contents of central storage. It manages the paging activities described earlier, such as page-in and page-out. The page manager also performs *page fixing* (marking pages as unavailable for paging).

In addition, the page manager keeps track of the contents of special use files called *page data sets.* Page data sets contain slots representing virtual storage pages that are not currently occupying a central storage frame. Page data sets also contain slots representing pages that do not occupy a frame, but, because the frame's contents have not been changed, the slots are still valid.

When a page-in or page-out is required, the page manager locates the proper central storage frames and auxiliary storage slots.

Virtual storage management services

The virtual storage management services respond to requests to obtain and free virtual storage. Storage is allocated to code and data when they are loaded in virtual storage. As they run, programs can request more storage by means of a system service, such as the *GETVIS macro*. Programs can release storage with the *FREEVIS macro.*

The virtual storage management services keep track of the map of virtual storage for each address space and partition. It knows two GETVIS areas:

- Partition GETVIS area

  Each partition has its own partition GETVIS area, which may cross the 16 MB line depending on the allocation value. The minimum is 48 KB and must be below 16 MB.

- System GETVIS area

  As the name implies, this area is reserved for system use. It is permanently assigned and belongs to the shared areas of virtual storage. z/VSE supports a 24-bit and a 31-bit system GETVIS area located in the corresponding shared areas. The 31-bit area may reside partly or completely below 16 MB.

Figure 3-10 on page 91 shows these GETVIS areas.

For dynamic partitions, z/VSE supports a 24-bit dynamic space GETVIS area. It can be considered as an extension of the system GETVIS area. The area exists from dynamic partition initialization until partition deactivation. The size of the dynamic space GETVIS area (the minimum is 128 KB) also influences the maximum partition size of a dynamic partition. The maximum size of a dynamic partition is: Private Area - Dynamic Space GETVIS Area.

z/VSE also supports z/OS macros GETMAIN and FREEMAIN for z/OS affinity.

In addition, virtual storage management services for the z/OS macros keep track of the map of virtual storage for each address space. In so doing, it sees an address space as a collection of 256 *subpools,* which are logically related areas of virtual storage identified by the numbers 0 to 255. Being logically related means that the storage areas within a subpool share characteristics such as:

- Storage protect key
- Whether they are fetch protected or pageable
- Where they must reside in virtual storage (above or below 16 megabytes)
- Whether they can be shared by more than one task

Some subpools (numbers 128 to 255) are predefined by use for system programs. Subpool 252, for example, is for authorized programs from authorized program libraries. Others (numbered 0 to 127) are defined by user programs.

3.4.8 A brief history of virtual storage and 64-bit addressability

In 1970, IBM introduced System/370, the first of its architectures to use virtual storage and address spaces. Since that time, the operating system has changed in many ways. One key area of growth and change is addressability.

A program running in an address space can reference all of the storage associated with that address space. In this text, a program's ability to reference all of the storage associated with an address space is called *addressability*.

Addressability
A program's ability to reference all of the storage associated with an address space

System/370 defined storage addresses as 24 bits in length, which meant that the highest accessible address was 16,777,215 bytes (or 2^{24}-1 bytes)[17]. The use of 24-bit addressability allowed DOS/VS (the *VSE* operating system at that time) to allot up to a total of 16 megabytes. Over the years, as VSE gained more functions and was asked to handle more complex applications. Even access to 16 megabytes of virtual storage fell short of user needs.

[17] Addressing starts with 0, so the last address is always one less than the total number of addressable bytes.

With the release of the System/370-XA architecture in 1983, IBM extended the addressability of the architecture to 31 bits. DOS/VSE (the VSE operating system at that time) did not implement S/370-XA architecture. Later, S/370-XA was followed by the ESA/370 architecture—also a 31-bit architecture. In 1993, VSE/ESA Version 1 Release 3 increased the addressability of virtual storage from 16 MB to 2 gigabytes (2 GB). The 16 MB address became the dividing point between the two architectures and is commonly called in z/VSE the *16 MB line*[18] (see Figure 3-7).

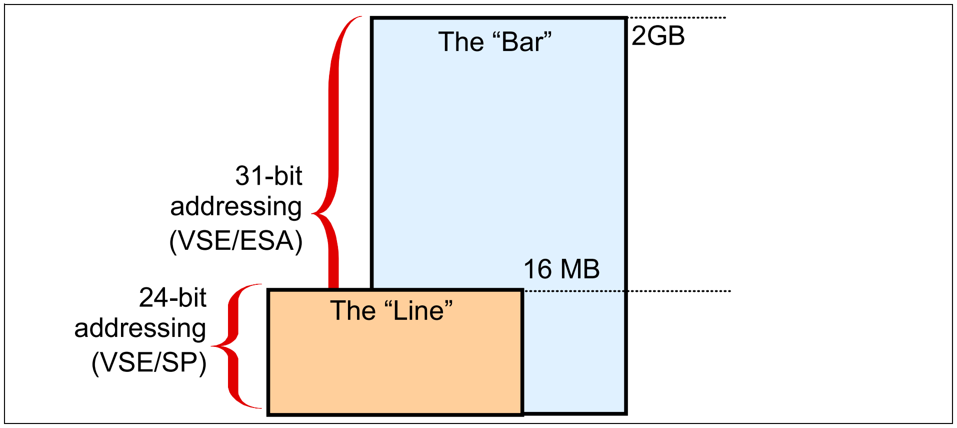


Figure 3-7 31-bit addressability allows for 2-gigabyte address spaces in VSE/ESA

The new architecture did not require customers to change existing application programs. To maintain compatibility for existing programs, VSE/ESA remained compatible for programs originally designed to run with 24-bit addressing, while allowing application developers to write new programs to exploit the 31-bit technology.

To preserve compatibility between the different addressing schemes, the ESA architecture did not use the *high-order bit* of the address (Bit 0) for addressing. Instead, it reserved this bit to indicate how many bits would be used to resolve an address: 31-bit addressing (Bit 0 on) or 24-bit addressing (Bit 0 off).

[18] Other mainframe operating systems like z/OS may only use the term *line* for it.

With the release of zSeries mainframes in 2000, IBM further extended the addressability of the architecture to 64 bits. z/VSE Version 4 introduces z/Architecture to VSE. z/VSE V4 is capable of addressing up to 8 GB of real processor storage (sometimes called real, or central storage). However, each virtual address space is limited to 2 GB (Figure 3-8).

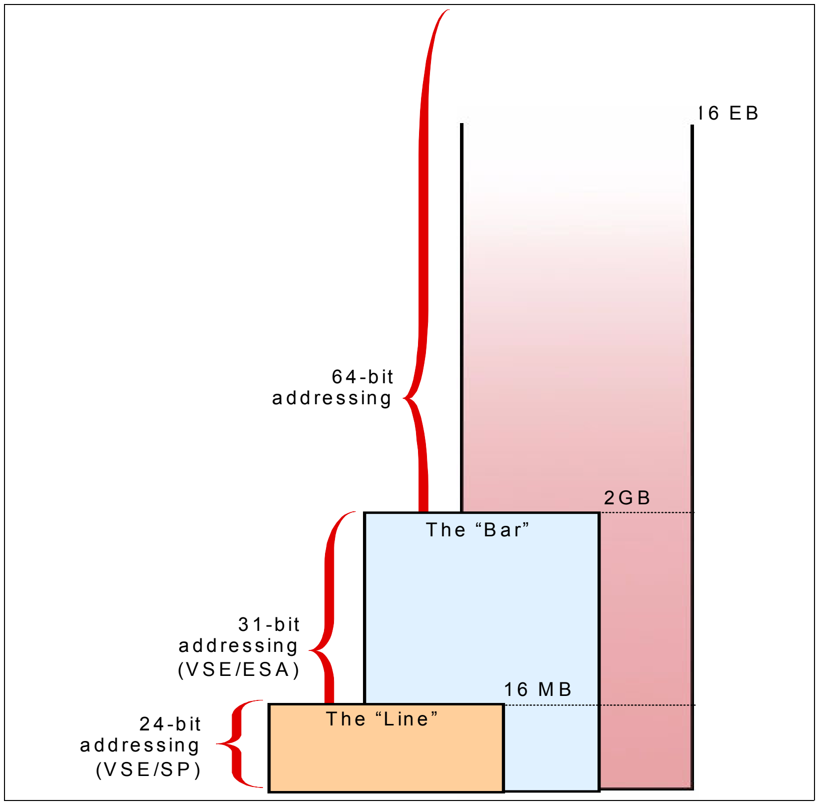


Figure 3-8 Using z/Architecture and 64-bit addressability

With z/VSE Version 4, there is a distinction between the size of real processor storage and the maximum size of an address space. z/VSE V4 implements 64-bit addressing for real processor storage. However, while the theoretical size of processor storage is 16 exabytes, z/VSE Version 4 Release 1 supports up to 8 GB of real processor storage. The maximum virtual address space remains 2 GB, as shown in Figure 3-9.

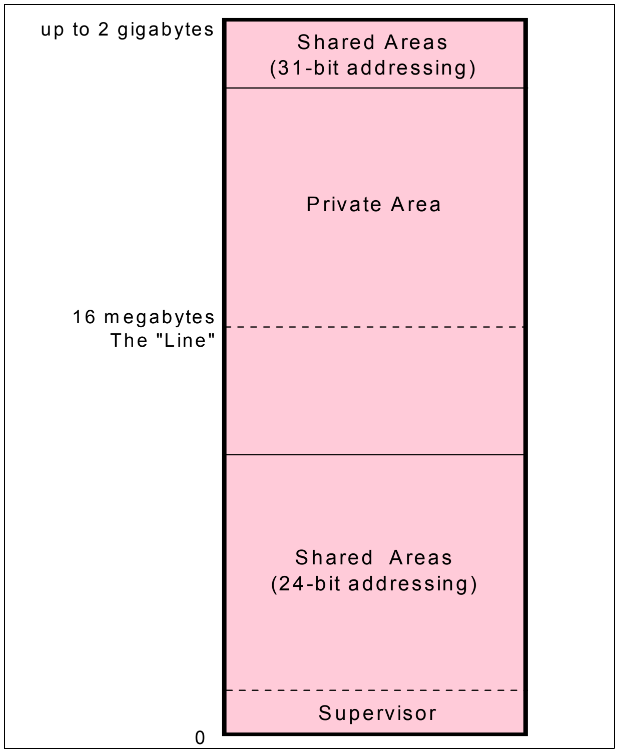

up to 2 gigabytes
Shared Areas (31-bit addressing)
Private Area
16 megabytes The "Line"
Shared Areas (24-bit addressing)
Supervisor
0

Figure 3-9 Storage map of z/VSE virtual address space

A program running on z/VSE V4 can run with 24-bit or 31-bit addressing (and can switch among these if needed).

3.4.9 What is meant by below-the-16-MB-line storage?

z/VSE programs and data reside in virtual storage that, when necessary, is backed by central storage. Most programs and data do not depend on their real addresses. Some z/VSE programs, however, do depend on real addresses, and some require these real addresses to be less than 16 megabytes. z/VSE programmers refer to this storage as being *below the 16-megabyte line*.

In z/VSE, a program's attributes include one called *residence mode* or RMODE, which specifies whether the program must reside (be loaded) in storage below 16 megabytes. A program with RMODE(24) must reside below 16 megabytes, while a program with RMODE(31) can reside anywhere in virtual storage.

Examples of programs that require below-the-16-MB-line storage include any program that uses application programming interfaces (APIs) that require 24-bit addressing, like some I/O APIs. Those programs, however, can be 31-bit addressing mode or AMODE(31), but they have to run below the 16 MB line.

Many programs in use today are AMODE(24) and therefore RMODE(24). Every program written before VSE/ESA was available, and not subsequently changed, has that characteristic. There are relatively few reasons these days why a new program might need to be AMODE(24), so a new application likely has next to nothing that is RMODE(24).

3.4.10 What is in an address space?

Another way of thinking of an address space is as a programmer's map of the virtual storage available for code and data. An address space provides each programmer with access to all of the addresses available through the computer architecture (earlier, we defined this as addressability).

z/VSE provides each job with a unique address space and maintains the distinction between the programs and data belonging to each address space. Because it maps all of the available addresses, however, an address space includes system code and data as well as user code and data. Thus, not all of the mapped addresses are available for user code and data.

Understanding the division of storage areas in an address space is made easier with a diagram. The diagram shown in Figure 3-10 is more detailed than needed for this part of the course, but is included here to show that an address space maintains the distinction between programs and data belonging to the user, and those belonging to the operating system.

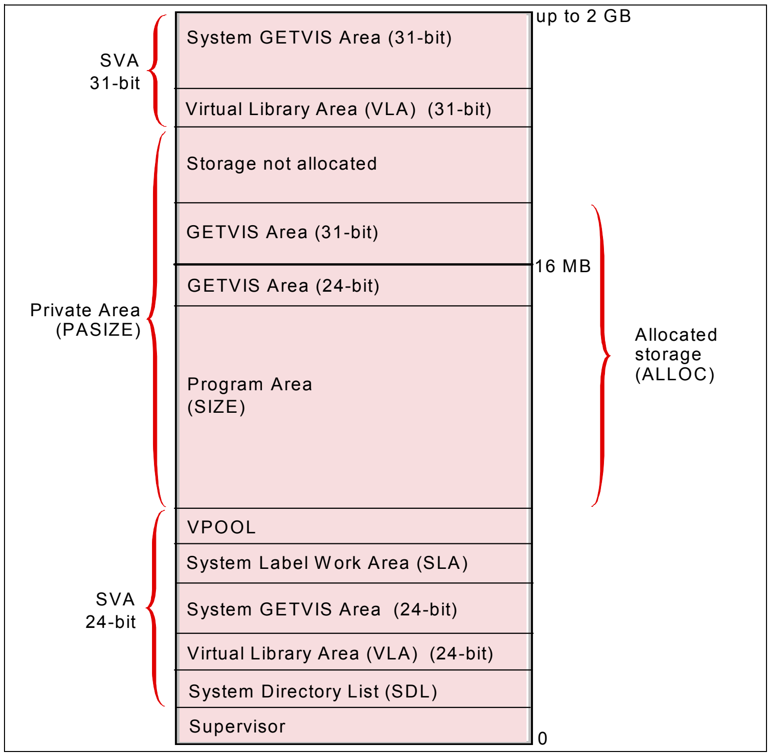

up to 2 GB
SVA 31-bit
System GETVIS Area (31-bit)
Virtual Library Area (VLA) (31-bit)
Storage not allocated
GETVIS Area (31-bit)
16 MB
GETVIS Area (24-bit)
Private Area (PASIZE)
Program Area (SIZE)
Allocated storage (ALLOC)
VPOOL
System Label Work Area (SLA)
SVA 24-bit
System GETVIS Area (24-bit)
Virtual Library Area (VLA) (24-bit)
System Directory List (SDL)
Supervisor
0

Figure 3-10 Storage areas in an address space

Figure 3-10 shows the major storage areas in each address space. These are described briefly:

- SVA 31-bit

  The Shared Virtual Area (SVA) above the 16 MB line contains:

  – System GETVIS area

    This area is used by the system to dynamically allocate virtual storage using the GETVIS macro. This storage can be freed with the FREEVIS macro. If, for z/OS affinity, the macro GETMAIN is used instead of GETVIS, the z/OS subpools 227, 228, 231, 239, 241, and 245 will also use this area.

  - Virtual Library Area (VLA)

    Phases that are often used are stored in this area during system startup.

- Private area

  The private area is this part of the virtual storage where the user program or application is loaded and where the this program takes its dynamically allocated storage. The size of the private area is defined by the PASIZE value during IPL.

  There are several applications that are started during system startup and stopped at system shutdown. They run in parallel in different partitions and use a different amount of storage in the private area. Each partition can have its own amount of allocated storage, defined for static partitions with the ALLOC command.

  The total maximum size of allocated storage of all partitions that can run concurrently is taken from the maximum total virtual storage size defined at IPL in the VSIZE parameter. The VSIZE can be up to 90 GB. It also defines the size of the page data sets.

  The total maximum virtual storage size does not only include the sum allocated storage of all partitions, it also includes the total size of all shared areas (supervisor, SVA) and the size reserved for virtual disks and data spaces. If the VSIZE is too small or the sum of allocated partition storage is too large, some of the partition will not run in parallel due to the lack of virtual storage and space for paging.

- SVA 24-bit

  The Shared Virtual Area (SVA) below the 16 MB line contains:

  - VPOOL

    The VPOOL area is needed to exchange data with the virtual I/O area (VIO).

  - System Label Work Area (SLA)

    This area is used to store and maintain system and user label information.

  - System GETVIS area

    This area is used by the system to dynamically allocate virtual storage using the GETVIS macro. This storage can be freed with the FREEVIS macro. If, for z/OS affinity, the macro GETMAIN is used instead of GETVIS, the z/OS subpools 226, 227, 228, 231, 239, 241, and 245 will also use this area.

  - Virtual Library Area (VLA)

    Phases that are often used are stored in this area during system startup.

- System Directory List (SDL)

  The SDL is the directory of the phases to be loaded into the SVA (VLAs) during the system startup.

- Supervisor

  In this area is the *low core* and a part of a control program that coordinates the use of resources and maintains the flow of processor operations.

  The low core is a common area of virtual storage starting from address zero in every address space. There is one unique Low Core for every processor installed in a system. The Low Core is 4k prior to z/VSE 4.1 and 8K in z/VSE 4.1. It maps architecturally fixed hardware and software storage locations for the processor. Because there is a unique Low Core for each processor, from the view of a program running on z/VSE, the contents of the Low Core can change any time the program is dispatched on a different processor. This feature is unique to the Low Core area and is accomplished through a unique DAT manipulation technique called *prefixing*.

3.4.11 System functions and the partition BG

The z/VSE system functions in SVA are available to each address space and partition. The static partition background partition (BG), for example, uses system functions to bring up the system.

When you start z/VSE after IPLing, a procedure ($0JCL) is used to initialize partition BG first. From this partition the other partitions, such as the security server partition (FB) and the VSE/POWER partition (F1), are started. VSE/POWER is the primary job entry subsystem. It starts other subsystems such as CICS or DB2.

3.5 I/O and data management

Nearly all work in the system involves data input or data output. In a mainframe, the channel subsystem manages the use of I/O devices, such as disks, tapes, and printers. The operating system must associate the data for a given task with a device, and provides services for file allocation, placement, monitoring, migration, backup, recall, recovery, and deletion.

3.6 Supervising the execution of work in the system

To enable multiprogramming, z/VSE requires the use of a number of supervisor controls, as follows:

- Interrupt processing

  Multiprogramming requires that there be some technique for switching control from one routine to another so that, for example, when routine A must wait for an I/O request to be satisfied, routine B can execute. In z/VSE, this switch is achieved by interrupts, which are events that alter the sequence in which the processor executes instructions. When an interrupt occurs, the system saves the execution status of the interrupted routine and analyzes and processes the interrupt.

- Creating dispatchable units of work

  To identify and keep track of its work, the z/VSE operating system represents each unit of work with a control block. Two types of control blocks represent dispatchable units of work: partition control blocks (PCBs) represent a partition and task control blocks or TCBs represent tasks.

- Dispatching work

  After interrupts are processed, the operating system determines which unit of work (of all the units of work in the system) is ready to run and has the highest priority, and passes control to that unit of work.

- Serializing the use of resources

  In a multiprogramming system, almost any sequence of instructions can be interrupted, to be resumed later. If that set of instructions manipulates or modifies a resource (for example, a data file), a program may use locking services to prevent other programs from using the resource until the interrupted program has completed its processing of the resource.

3.6.1 What is interrupt processing?

An interrupt is an event that alters the sequence in which the processor executes instructions. An interrupt might be planned (specifically requested by the currently running program) or unplanned (caused by an event that might or might not be related to the currently running program). z/VSE uses six types of interrupts, as follows:

- Supervisor call or SVC interrupts

  These occur when the program issues an SVC to request a particular system service. An SVC interrupts the program being executed and passes control to the supervisor so that it can perform the service. Programs request these services through macros such as GETVIS (obtain storage), ATTACH (create a new task = subtask), and EOJ (terminate a task).

- I/O interrupts

  These occur when the channel subsystem signals a change of status, such as an I/O operation completing, an error occurring, or an I/O device such as a printer has become ready for work.

- External interrupts

  These can indicate any of several events, such as a time interval expiring, the operator pressing the interrupt key on the console, or the processor receiving a signal from another processor.

- Restart interrupts

  These occur when the operator selects the restart function at the console or when a restart SIGP (signal processor) instruction is received from another processor.

- Program interrupts

  These are caused by program errors (for example, the program attempts to perform an invalid operation), page faults (the program references a page that is not in central storage), or requests to monitor an event.

- Machine check interrupts

  These are caused by machine malfunctions.

When an interrupt occurs, the hardware saves pertinent information about the program that was interrupted and, if possible, disables the processor for further interrupts of the same type. The hardware then routes control to the appropriate interrupt handler routine. The program status word (PSW) is a key resource in this process.

How is the program status word used?

Modern System z processors provide two types of architectures: ESA/390 and z/Architecture. In systems running ESA/390 mode, for example, z/VSE 3.1, the program status word, is a 64-bit data area in the processor that, along with a variety of other types of registers (control registers, general registers, and prefix registers) provides details crucial to both the hardware and the software. In systems running z/Architecture mode (for example, z/VSE 4.1), the PSW is a 128-bit data area. The current PSW includes the address of the next program instruction and control information about the program that is running. Each processor has only one current PSW. Thus, only one task can execute on a processor at a time. To ensure compatibility to system as well as user programs, z/VSE 4.1 maintains z/Architecture as well as ESA/390 data areas.

The PSW controls the order in which instructions are fed to the processor, and indicates the status of the system in relation to the currently running program. Although each processor has only one PSW, it is useful to think of three types of PSWs in order to understand interrupt processing:

- Current PSW
- New PSW
- Old PSW

The current PSW indicates the next instruction to be executed. It also indicates whether the processor is enabled or disabled for I/O interrupts, external interrupts, machine check interrupts, and certain program interrupts. When the processor is enabled, these interrupts can occur. When the processor is disabled, these interrupts are ignored or remain pending.

There is a new PSW and an old PSW associated with each of the six types of interrupts. The new PSW contains the address of the routine that can process its associated interrupt. If the processor is enabled for interrupts when an interrupt occurs, PSWs are switched through the following technique:

1. Storing the current PSW in the old PSW associated with the type of interrupt that occurred
2. Loading the contents of the new PSW for the type of interrupt that occurred into the current PSW

The current PSW, which indicates the next instruction to be executed, now contains the address of the appropriate routine to handle the interrupt. This switch has the effect of transferring control to the appropriate interrupt handling routine.

Registers and the PSW

The mainframe architecture provides registers to keep track of things. See Figure 3-11.The PSW, for example, is a register used to contain information that is required for the execution of the currently active program. Mainframes provide other registers, as follows:

- *Access registers* are used to specify the address space in which data is found.
- *General registers* are used to address data in storage, and also for holding user data.
- *Floating point registers* are used to hold numeric data in floating point form.
- *Control registers* are used by the operating system itself (for example, as references to translation tables).

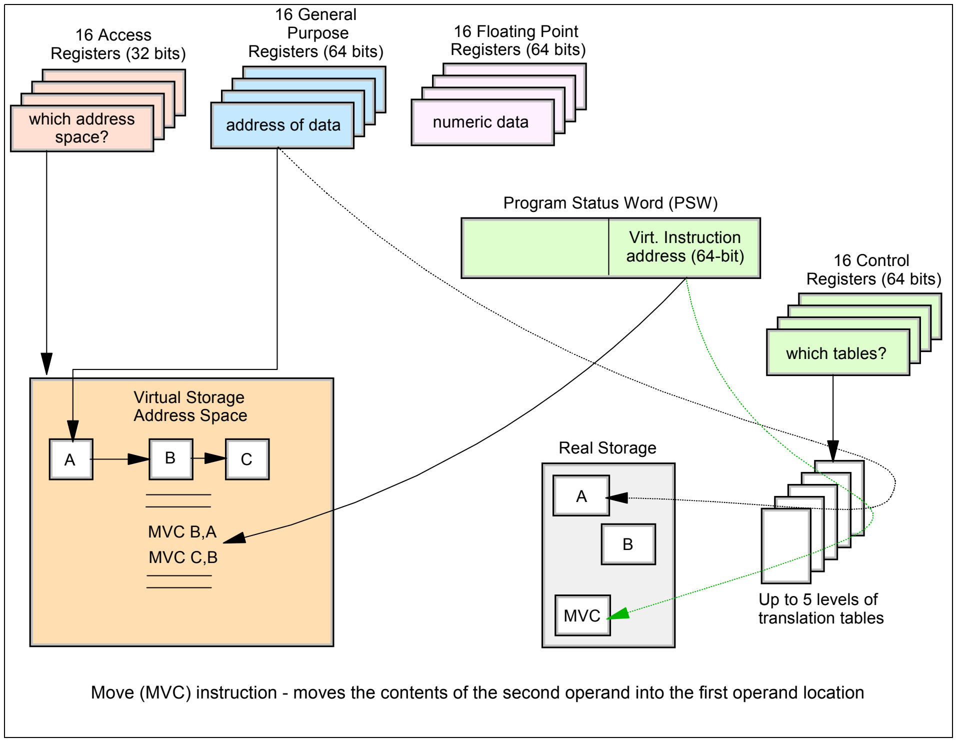

16 Access Registers (32 bits)
which address space?
16 General Purpose Registers (64 bits)
address of data
16 Floating Point Registers (64 bits)
numeric data
Program Status Word (PSW)
Virt. Instruction address (64-bit)
16 Control Registers (64 bits)
which tables?
Virtual Storage Address Space
A
B
C
MVC B,A
MVC C,B
Real Storage
A
B
MVC
Up to 5 levels of translation tables
Move (MVC) instruction - moves the contents of the second operand into the first operand location

Figure 3-11 Registers and the PSW

Related reading: The IBM publications *ESA/390 Principles of Operation* and *z/Architecture Principles of Operation* describe the hardware facilities for the switching of system status, including CPU states, control modes, the PSW, and control registers. You can find this and other related publications starting from the z/VSE Web site:

`http://www.ibm.com/servers/eserver/zseries/zvse/documentation/`

There you will find a link to the IBM Publication Center. Select your language and search for *Principles of Operation*.

3.6.2 Creating dispatchable units of work

In z/VSE, dispatchable units of work are represented by two types of control blocks:

- Partition control blocks (PCBs)

  These represent partitions[19] allocated within one address space. Job Control, system, or user programs may execute within a partition. Partitions may hold up to 32 tasks (a maintask, which is always present, and subtasks) attached by a user program (a maximum of 31 subtasks can be attached).

- Task control blocks (TCBs)

  These represent tasks (maintask or subtasks) executing within a partition, such as user programs and system programs that support user programs.

What is a PCB?

A partition control block (PCB) represents a partition, such as your program, as it runs in an address space. It contains control information, the partition layout, and tasks that belong to it.

Do not confuse the z/VSE term *PCB* with the UNIX data structure called a *process control block* or *PCB*.

What is a TCB?

A TCB is a control block that represents a task, such as your program or routine of your program, as it runs in a partition. A TCB contains information about the running task, such as control information, the address of the save area, that holds the program's status (PSW and general registers of last interrupt).

TCBs can be created in response to an ATTACH macro. By issuing the ATTACH macro, a user program or system routine begins the execution of the program specified on the ATTACH macro, as a subtask of the partition. As a subtask, the

[19] In z/OS an address space does not contain partitions. Therefore an address space with one partition can be seen as a region in z/OS.

specified program can compete for processor time and can use certain resources already allocated to the partition.

z/VSE allocates up to 32 system tasks (system TCBs), which execute system function, like services for I/O, page manager, fetch/load, or the dispatcher.

3.6.3 Preemptable versus non-preemptable

Which routine receives control after an interrupt is processed depends on whether the interrupted unit of work was preemptable. If so, the operating system determines which unit of work should be performed next. That is, the system determines which unit or work, of all the work in the system, has the highest priority, and passes control to that unit of work.

A non-preemptable unit of work can be interrupted, but must receive control after the interrupt is processed. For example, system routines are often non-preemptable. Thus, if a routine represented by a non-preemptable system routine is interrupted, it will receive control after the interrupt has been processed. In contrast, a routine represented by a user task, such as a user program, is usually preemptable[20]. If it is interrupted, control returns to the operating system when the interrupt handling completes. z/VSE then determines which task, of all the ready tasks, will execute next.

3.6.4 What does the dispatcher do?

New work is selected, for example, when a task is interrupted or becomes non-dispatchable, or after an interrupt is processed.

In z/VSE, the dispatcher component (called the turbo dispatcher) is responsible for routing control to the highest priority unit of work that is ready to execute.

The turbo dispatcher processes work in the following order:

1. Special exits

   These are exits that have a high priority because of specific conditions in the system (for example, if an I/O system function needs to get control before the dispatcher can select a partition/task).

2. System tasks have the highest priority.
3. Ready partitions and tasks in order of priority.

   A partition is ready when at least one of its tasks is ready to execute (that is, if it is not waiting for some event to complete). A partition's priority is determined by the dispatching priority specified by the user (via PRTY

[20] A user task is non-preemptable when it is executing an SVC.

command) or the installation. After selecting the highest priority partition, z/VSE (through the dispatcher) selects the highest priority task within the partition. First it executes system exits for the selected task, so-called dispatcher exits (for example, subsystem services), and then the dispatcher will restore the task's status and load the PSW to give control to the program.

If there is no ready work in the system, z/VSE assumes a state called allbound state (enabled wait) until fresh work enters the system.

Different models of the System z hardware can have one or multiple central processors (CPs). Each and every CP can be executing instructions at the same time. Dispatching priorities determine when ready-to-run partitions get dispatched.

An partition can be on the ready-to-run queue or in wait queues and may have the following states:

- Ready - waiting to be dispatched
- Wait - waiting on some event to complete

Characteristics of the z/VSE dispatcher

The z/VSE dispatcher (called turbo dispatcher) can utilize uniprocessors as well as multiprocessors by distributing the workload across several central processors (CPs), enabling them to work in parallel and thus increase the overall throughput of a z/VSE system.

Further characteristics of the turbo dispatcher are:

- z/VSE can run:
  - Native
  - In a LPAR (logical partition)
  - Under z/VM as a guest system
- The turbo dispatcher uses one CP during Initial Program Load (IPL). Additional CPs can be started through the startup procedure of the BG partition or by the operator through an attention routine command.

Key 0 and non-key 0 programs

It is important to understand the dispatching of key 0 and non-key 0 programs, because the amount of key 0 units of work determine the exploitation of CPs. Application programs are usually non-key 0 programs. They access data in the same partition in which they run and can be processed in parallel with other programs. Key 0 programs are usually system programs and require non-parallel processing. They have read and write access to any storage area.

IPL-CP and additional CPs

Although for the turbo dispatcher all CPs of a multiprocessor have equal rights, the CP from which IPL is performed has certain functional *privileges*. After IPL completes, this is the only active CP. The other CPs (here called additional CPs) must be started separately. The IPL-CP cannot be stopped via a command, as is possible for additional CPs. This ensures that z/VSE runs at least in *uniprocessor mode* (with only the IPL-CP active). The z/VSE system can best exploit up to three CPs.

How does the turbo dispatcher work?

The turbo dispatcher distributes work dynamically on the partition level (that is, it dispatches an entire partition to a CP waiting for work). The program or application running in the partition consists, as seen by the turbo dispatcher, of many units of work. One unit of work is defined as a set of instructions processed from the point of selection by the turbo dispatcher until the next interrupt.

Processing rules

From a dispatching point of view, all CPs of a multiprocessor have equal rights. That is, all CPs receive I/O (input/output) interrupts and external interrupts, and the turbo dispatcher dedicates work to a specific CP in exceptional cases only. Only one unit of work of a partition (program) can be processed at a time. That is, no other unit of work of the same partition can run on another CPU concurrently.

The turbo dispatcher distinguishes two types of units of work:

- Parallel units of work (PA)

  Customer applications in a batch or online (CICS) environment can be processed mostly as parallel units of work. To be processed in parallel, units of work must be part of different programs that reside in different partitions. The parallel processing of programs is frequently interrupted for the processing of non-parallel units of work (for example, when system services are required).

- Non-parallel units of work (NP)

  Typical examples of non-parallel units of work are most z/VSE system services and key 0 programs (such as supervisor services, CICS or VTAM services). Non-parallel units of work cannot be processed in parallel. Only one CP of a multiprocessor system can process a non-parallel unit of work at any point in time. But at the same time, the other CPs can process parallel units of work of other partitions. VSE/POWER is an exception. Although it is a system program and as such a key 0 program, parallel processing of VSE/POWER is possible and can be requested when running on a multiprocessor. The code of programs that must be processed non-parallel is also referred to as *non-parallel code*.

The turbo dispatcher identifies units of work and their status and initiates their processing according to the following rules:

- If a CP processes a unit of work of a particular partition (program), no other CP can execute any other unit of work of that partition. This means that for programs using multitasking (programs with attached z/VSE subtasks), no other task of the same program can be processed on another CP when one task of that program is already active.
- Only one CP can process a non-parallel unit of work at a time. This means that during this time no other CP can process a non-parallel unit of work even if it belongs to another partition (program).

3.6.5 Serializing the use of resources

In a multitasking, multiprocessing environment, resource serialization is the technique used to coordinate access to resources that are used by more than one application. Programs that change data need exclusive access to the data. Otherwise, if several programs were to update the same data at the same time, the data could be corrupted (also referred to as a loss of data integrity). On the other hand, programs that need only to read data can safely share access to the same data at the same time.

The most common techniques for serializing the use of resources is locking. This technique allows for orderly access to resources needed by more than one user in a multiprogramming environment. In z/VSE, locking is managed by the lock manager of the z/VSE supervisor component.

Locking is the means by which a program running on z/VSE requests control of a serially reusable resource. Locking is accomplished by means of the LOCK and UNLOCK macros, which are available to all programs running on the system.

3.7 Defining characteristics of z/VSE

The defining characteristics of z/VSE are summarized as follows:

- The use of address spaces in z/VSE holds many advantages: isolation of private areas in different address spaces provides for system security, yet each address space also provides a common area that is accessible to every address space.
- The system is designed to preserve *data integrity*, regardless of how large the user population might be. z/VSE prevents users from accessing or changing any objects on the system, including user data, except by the system-provided interfaces that enforce authority rules.

- The system is designed to manage a large number of concurrent batch jobs, with no need for the customer to externally manage workload balancing or integrity problems that might otherwise occur due to simultaneous and conflicting use of a given set of data.
- The security design extends to system functions as well as simple files. Security can be incorporated into applications, resources, and user profiles.
- The system allows multiple communications subsystems at the same time, permitting unusual flexibility in running disparate communications-oriented applications (with mixtures of test, production, and fall-back versions of each) at the same time. For example, multiple TCP/IP stacks can be operational at the same time, each with different IP addresses and serving different applications.
- The system provides extensive software recovery levels, making unplanned system restarts rare in a production environment. System interfaces allow application programs to provide their own layers of recovery. These interfaces are seldom used by simple applications—they are normally used by more sophisticated applications.
- The system is designed to manage disparate workloads, with automatic balancing of resources to meet production requirements established by the system administrator.
- The system is designed to routinely manage I/O configurations with up to 1024 devices. This might be disk drives, multiple automated tape libraries, many printers, networks of terminals, and so forth.
- The system is controlled from one or more operator terminals.
- The operator interface is a critical function of z/VSE. It provides status information, messages for exception situations, control of job flow, hardware device control, and allows the operator to manage unusual recovery situations.

3.8 Additional software products for z/VSE

A z/VSE customer usually purchases additional products (such as IBM middleware) that are needed to create a practical working system. For example, a production z/VSE system often includes a transaction server product and a database manager product. When talking about a z/VSE *platform* or *environment*, people often assume the inclusion of these additional products.

Many of the most commonly used IBM software products are packaged as part of the z/VSE *base*. Examples are VSE Central Functions (base operating system functions), High Level Assembler, CICS Transaction Server for VSE/ESA, TCP/IP for VSE/ESA, DB2 Server for VSE and VM, and so on. The z/VSE base is designed, implemented, tested, packaged, and serviced as if it were a single, integrated product.

Licensed program
An additional, priced software product, not part of the z/VSE base

Other commonly used VSE-related IBM software products are available as *IBM licensed programs*, also called Optional Products. Examples are languages (C, COBOL, and PL/I), RPG II, SDF II, MQSeries, Print Service Facilities (PSF/VSE), and DL/1. Optional products do not have the same level of integration as base products. Nevertheless, they are tested with the z/VSE base. Service of optional products and VSE base products is coordinated as well.

A large number of VSE-related products with different functionality are available from various independent software vendors (ISVs), as they are commonly called in the industry. Examples are security managers and database managers. The ability to choose from a variety of licensed programs to accomplish a task considerably increases the flexibility of the z/VSE operating system and allows the mainframe IT group to tailor the products it runs to meet their company's specific needs.

We do not attempt to list all of the IBM licensed z/VSE programs in this text (dozens exist). Some common choices include:

- Security system

  z/VSE provides a basic framework for security. The VSE Basic Security Manager (BSM) is included in z/VSE at no additional charge. If more functionality is required, external security manager products are available from ISVs to be used.

- Compilers

  The z/VSE base includes the High Level Assembler (HLASM). Compiler for other programming languages, such as COBOL, PL/I, and C are offered as optional products.

- Relational database

  One example is DB2. Other types of database products such as DL/1, a hierarchical database, are also available as an optional product or from ISVs.

- Transaction processing facilities

  IBM-offered CICS Transaction Server for VSE/ESA is widely used by VSE customers.

- Sort program

  Fast, efficient sorting of large amounts of data is highly desirable in batch processing. IBM offers DFSORT/VSE. Other sorting products are available from ISVs as well.

- A large variety of utility programs

  For example, the DITTO/ESA program, which is used extensively to copy tapes, and view and edit data, is a licensed product.

A large number of complementary VSE-related products are available from ISVs. For example, ISVs offer products that provide common systems management functions, such as job scheduling, console management, tape/DASD management, performance monitors, and application development tools.

3.9 Middleware for z/VSE

Middleware is typically software between the operating system and an end user (or end-user application). It supplies major functions not provided by the operating system. The term usually applies to major software products such as database managers, transaction monitors, and so forth. *Subsystem* is another term often used for this type of software. These are usually licensed programs, although there are exceptions.

Middleware
Software that supplies major functions not provided by the operating system

z/VSE is a base for using 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, database queries and updates, and so on) is characteristic of z/VSE.

Typical z/VSE middleware includes:

- Transaction managers
- Database systems
- Message queueing and routing functions

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. An example of mainframe middleware APIs includes the DB2 database management product, which provides an API (expressed in the SQL language) that is used with many different languages and applications.

3.10 A brief comparison of z/VSE and UNIX

What do we find if we compare z/VSE and UNIX? In many cases, we find that quite a few concepts are mutually understandable to users of either operating system, despite the differences in terminology.

For experienced UNIX users, Table 3-1 provides a small sampling of familiar computing terms and concepts. As a new user of z/VSE, many of the z/VSE terms will sound unfamiliar to you. As you work through this course, however, the z/VSE meanings will be explained and you will find that many elements of UNIX have analogs in z/VSE.

Table 3-1 Mapping UNIX to z/VSE terms and concepts

Term or concept	UNIX	z/VSE
Start the operating system	Boot the system.	Initial program load (IPL) the system.
Virtual storage given to each user of the system	Users get whatever virtual storage they need to reference, within the limits of the hardware and operating system.	Jobs get an address space. A range of addresses extends to 2 GB of virtual storage, though some of this storage contains system code that is common for all users.
Data storage	Files.	Data sets (sometimes called files).
Data format	Byte orientation. Organization of the data is provided by the application.	Record orientation. Often an 80-byte record reflecting the traditional punched card image.
System configuration data	The /etc file system controls characteristics.	Parameters control how the system IPLs and how address spaces behave.
Scripting languages	Shell scripts, Perl, awk, and other languages.	REXX execs.
Smallest element that performs work	A thread. The kernel supports multiple threads.	A task. The z/VSE system functions support multiple tasks.
A long-running unit of work	A daemon.	A long-running job. Often this is a subsystem of z/VSE.

Order in which the system searches for programs to run	Programs are loaded from the file system according to the user's PATH environmental variable (a list of directories to be searched).	The system searches the VSE/LIBRARIAN for the program to be loaded.
Using the system interactively	Users *log in* to systems and execute shell sessions in the shell environment. They can issue the `rlogin` or `telnet` commands to connect to the system. Each user can have many login sessions open at once.	Authorized users *log on* to the system through one or more VSE consoles, the VSE Interactive User Interface (IUI)[a], or VSE/ICCF.
Editing data or code	Many editors exist, such as vi, ed, sed, and emacs.	ICCF editor.
Source and destination for input and output data	stdin and stdout.	SYSRDR, SYSIPT for input. SYSLST, SYSPCH for output.
Managing programs	The `ps` shell command allows users to view processes and threads, and kill jobs with the `kill` command.	The Interactive Interface allows users to view and terminate their jobs.

a. Interactive User Interface (IUI) is the old name for the Interactive Interface. Nevertheless, the abbreviation IUI is still widely used in z/VSE.

A major difference for UNIX users moving to z/VSE is the idea that the user is just one of *many* other users. In moving from a UNIX system to the z/VSE environment, users typically ask questions such as "Can I have the root password because I need to do....." or "Would you change this or that and restart the system?" It is important for new z/VSE users to understand that potentially thousands of other users are active on the same system, and so the scope of user actions and system restarts in z/VSE are carefully controlled to avoid negatively affecting other users and applications. Also, under z/VSE, there does not exist a single root password or root user.

Both z/VSE and UNIX provide APIs to allow in-memory data to be shared between processes. In z/VSE, a user can access another user's address spaces directly through cross-memory APIs. Similarly, UNIX has the concept of shared memory functions, and these can be used on UNIX without special authority.

z/VSE cross-memory APIs, however, require the issuing program to have special authority. These methods allows efficient and secure access to data owned by others, data owned by the user but stored in another address space for convenience, and for rapid and secure communication with services like transaction managers and database managers.

3.11 Summary

An operating system is a collection of programs that manage the internal workings of a computer system. The operating system taught in this course is z/VSE, a widely used mainframe operating system. The z/VSE operating system's use of multiprogramming and multiprocessing, and its ability to access and manage enormous amounts of storage and I/O operations, makes it ideally suited for running mainframe workloads.

The concept of virtual storage is central to z/VSE. Virtual storage is an illusion created by the architecture, in that the system seems to have more storage than it really has. Virtual storage is created through the use of tables to map virtual storage pages to frames in central storage or slots in auxiliary storage. Only those portions of a program that are needed are actually loaded into central storage. z/VSE keeps the inactive pieces of address spaces in auxiliary storage.

z/VSE is designed around address spaces, which are ranges of addresses in virtual storage. Each user of z/VSE gets an address space containing the same range of storage addresses. The use of address spaces in z/VSE allows for isolation of private areas in different address spaces for system security, yet also allows for inter-address space sharing of programs and data through a common area accessible to every address space.

The terms processor storage, central storage, real memory, main storage, and main memory are synonyms and are used interchangeably. Likewise, virtual memory and virtual storage are used interchangeably.

The amount of processor storage needed to support the virtual storage in an address space depends on the working set of the application being used, and this varies over time. A user does not automatically have access to all of the virtual storage in the address space. Requests to use a range of virtual storage are checked for size limitations, and then necessary paging table entries are constructed to create the requested virtual storage.

Programs running on z/VSE and System z mainframes can run with 24-bit or 31-bit addressing (and can switch between these modes if needed). Programs can use a mixture of instructions with 16-bit or 32-bit operands, and can switch between these if needed.

Mainframe operating systems seldom provide complete operational environments. They depend on licensed programs for middleware and other functions. In addition to IBM, many ISVs provide middleware and various utility products.

Middleware is a relatively recent term that can embody several concepts at the same time. A common characteristic of middleware is that it provides a programming interface, and applications are written (or partially written) to this interface.

Key terms in this chapter			
Address space	Addressability	Auxiliary storage	Central storage
Control block	Dynamic address translation (DAT)	Frame	Input/output (I/O)
Licensed program	Middleware	Multiprogramming	Multiprocessing
Page/paging	Page selection	Service level agreement (SLA)	Slot
Partition	Virtual storage	ISVs	z/VSE

3.12 Questions for review

To help test your understanding of the material in this chapter, complete the following questions:

1. How does z/VSE differ from a single-user operating system? Give two examples.
2. z/VSE is designed to take advantage of what mainframe architecture? In what year was it introduced?
3. List the three major types of storage used by z/VSE.
4. What is *virtual* about virtual storage?

5. Match the following terms:

a. Page	__ Auxiliary storage
b. Frame	__ Virtual storage
c. Slot	__ Central storage

6. List several defining characteristics of the z/VSE operating system.
7. List three types of software products that might be added to z/VSE to provide a complete system.
8. List several differences and similarities between z/VSE and UNIX operating systems.
9. Which of the following is/are not considered to be middleware in a z/VSE system?
   a. Transaction managers
   b. Database managers
   c. Auxiliary storage manager

3.13 Topics for further discussion

Further exploration of z/VSE concepts could include the following areas of discussion:

1. Why might moving programs and data blocks from below the 16 MB line to above the 16 MB line be complicated for application owners? How might this be done without breaking compatibility with existing applications?
2. An application program can be written to run in 24-bit or 31-bit addressing mode. How does the programmer select this? In a high-level language? In Assembler language?
3. Will more central storage allow a system to run faster? What measurements indicate that more central storage is needed? When is no more central storage needed? What might change this situation?
4. Why are licensed programs needed? Why not simply include all of the software with the operating system?

4

The Interactive Interface of z/VSE

Objective: In working with the z/VSE operating system, you will need to know its end-user interfaces. Chief among these is the Interactive Interface. These dialogs allow you to log on to the system, run programs, and manipulate data files.

After completing this chapter, you will be able to:

- Log on to z/VSE.
- Navigate through the menu options of the Interactive Interface.
- Get help in various problem situations.
- Use the ICCF fulist to display your primary ICCF library.
- Use the ICCF editor to make changes to a data set.
- Perform main functions for an administrator, an operator, and a programmer.

4.1 How do we interact with z/VSE?

We have mentioned that z/VSE is ideal for processing batch jobs (workloads that run in the background with little or no human interaction). However, z/VSE is just as much an interactive operating system as it is a batch processing system. By *interactive* we mean that end users (sometimes tens of thousands of them concurrently in the case of z/VSE) using the system through direct interaction, such as commands and menu style user interfaces.

z/VSE provides a number of facilities to allow users to interact directly with the operating system. This chapter provides an overview of each facility, as follows:

- 4.3, "Interactive Interface overview" on page 114, shows how to log on to z/VSE, describes the user IDs that are defined in z/VSE, and shows main selection menus.
- 4.3.1, "Navigating through Interactive Interface menus" on page 116, lists the functions that are most frequently needed by online users.
- 4.3.2, "Using PF1-HELP and the z/VSE tutorial" on page 118, explains the use of the z/VSE tutorial and the help function provided by the PF1 - help key.
- 4.4.1, "Administrator tasks" on page 120, describes important tasks for a system administrator.
- 4.4.2, "Operator tasks" on page 124, lists some tasks that are typical for an operator.
- 4.4.3, "Important tasks for a programmer" on page 127, explains tasks for the daily use of a programmer.

Hands-on exercises are provided at the end of the chapter to help students develop their understanding of these important facilities.

In addition to these terminal-based interactions, z/VSE provides a graphical user interface via the VSE Navigator application. For more details refer to 13.11, "Using the VSE Navigator application" on page 354.

4.2 Logon to z/VSE

z/VSE Interactive Interface allows users to create an interactive session with the z/VSE system. It provides a single-user logon capability and a basic command prompt interface to z/VSE.

Logon
The procedure by which a user begins a terminal session

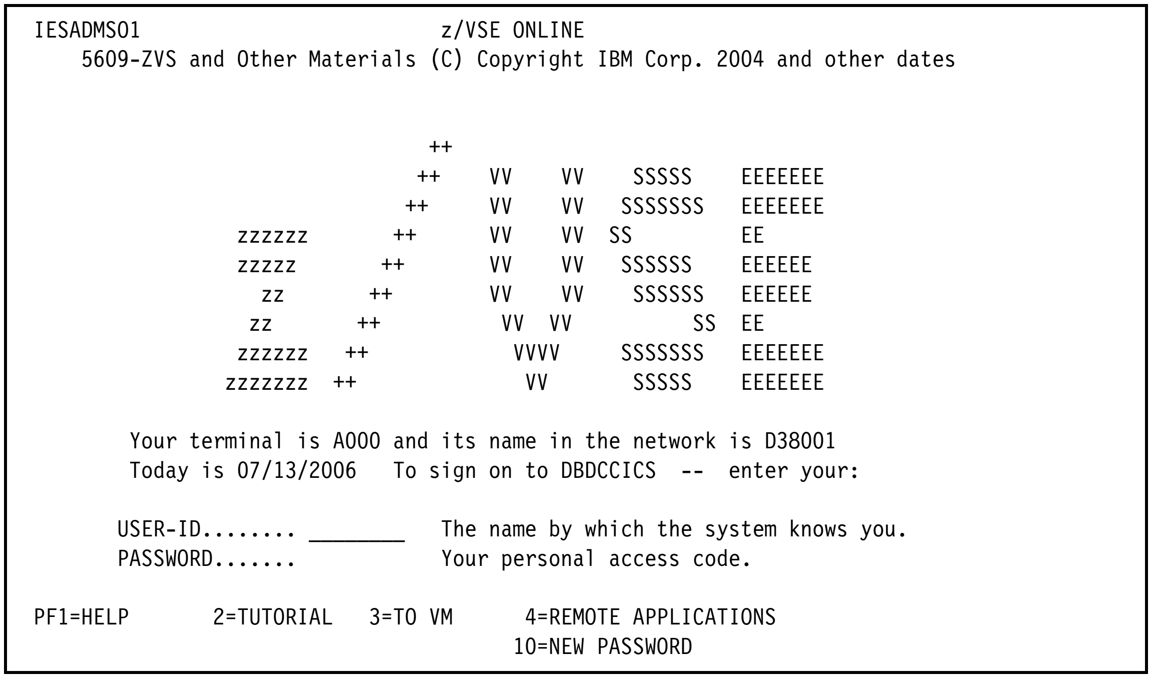
Most users work with z/VSE through its menu-driven interface, the *Interactive Interface.* This collection of menus and panels offers a wide range of functions to assist users in working with data files on the system. The users are system administrators, system programmers, operators, application programmers, and general users. In general, the Interactive Interface makes it easier for people with varying levels of experience to interact with the z/VSE system.

In a z/VSE system, each user is granted a user ID and a password authorized for logon. Logging on to z/VSE requires a 3270 display device or, more commonly, a TN3270 emulator running on a PC.

During logon, the system displays the z/VSE logon screen on the user's 3270 display device or TN3270 emulator. The logon screen serves the same purpose as a Windows logon panel.

z/VSE system programmers often modify the particular text layout and information of the z/VSE logon panel to better suit the needs of the system's users. Therefore, the screen captures shown in this book will likely differ from what you might see on an actual production system.

Figure 4-1 shows a typical example of a z/VSE logon screen.



```
IESADMSO1                         z/VSE ONLINE
    5609-ZVS and Other Materials (C) Copyright IBM Corp. 2004 and other dates



                                  ++
                                 ++    VV    VV    SSSSS    EEEEEEE
                                ++     VV    VV   SSSSSSS   EEEEEEE
                 zzzzzz        ++      VV    VV  SS         EE
                 zzzzz        ++       VV    VV   SSSSSS    EEEEEE
                   zz        ++        VV    VV    SSSSSS   EEEEEE
                  zz        ++          VV  VV         SS  EE
                 zzzzzz   ++             VVVV     SSSSSSS   EEEEEEE
                zzzzzzz  ++               VV       SSSSS    EEEEEEE

        Your terminal is A000 and its name in the network is D38001
        Today is 07/13/2006   To sign on to DBDCCICS  --  enter your:

       USER-ID........ ________   The name by which the system knows you.
       PASSWORD.......            Your personal access code.

PF1=HELP       2=TUTORIAL   3=TO VM      4=REMOTE APPLICATIONS
                                        10=NEW PASSWORD
```

Figure 4-1 Typical z/VSE logon screen

4.3 Interactive Interface overview

Every user of the Interactive Interface is defined by a user profile. The profile includes a unique user ID and password that are used to sign on to the Interactive Interface. It also determines what is displayed after the user signs on. The system can display a selection panel or access an application directly. The user needs a certain authorization to do his tasks. The authorization of a user is defined in the user profile as *user type*. There are three user types:

- Type 1 or type administrator

  This is used for the system administrators and system programmers.

- Type 2 or type programmer or operator

  This is used for application programmers and operators.

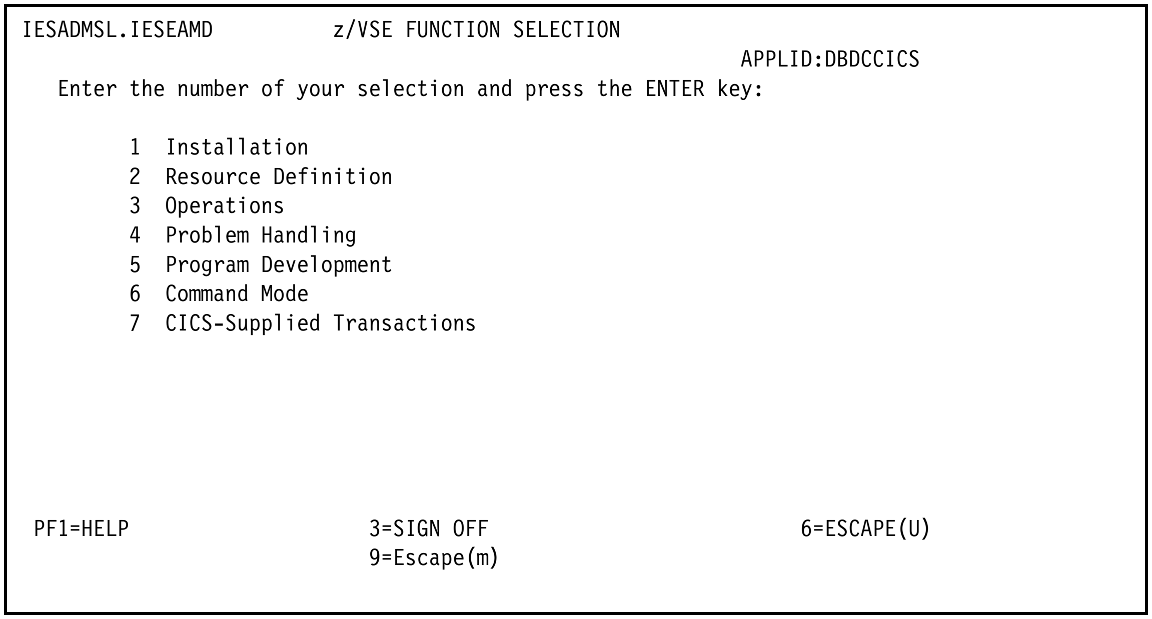
- Type 3 or type general user

  This is used for all other users.

z/VSE provides the user IDs *SYSA*, *PROG*, and *OPER*, which can be used as models to define your own profiles for an administrator, operator, or programmer.

After logging on to z/VSE, users typically access the primary selection menu. The Interactive Interface is a full panel application navigated by keyboard. It includes a text editor and browser, and functions for locating and listing files and performing other utility functions. Interactive Interface menus list the functions that are most frequently needed by online users.

Figure 4-2 shows the primary selection menu for an administrator.



```
IESADMSL.IESEAMD          z/VSE FUNCTION SELECTION
                                                            APPLID:DBDCCICS
   Enter the number of your selection and press the ENTER key:

         1  Installation
         2  Resource Definition
         3  Operations
         4  Problem Handling
         5  Program Development
         6  Command Mode
         7  CICS-Supplied Transactions






 PF1=HELP                    3=SIGN OFF                            6=ESCAPE(U)
                             9=Escape(m)
```

Figure 4-2 Primary selection panel for administrators

z/VSE provides a menu tree, oriented on the task that should be performed.

Figure 4-3 shows the first part of the menu structure for an administrator.

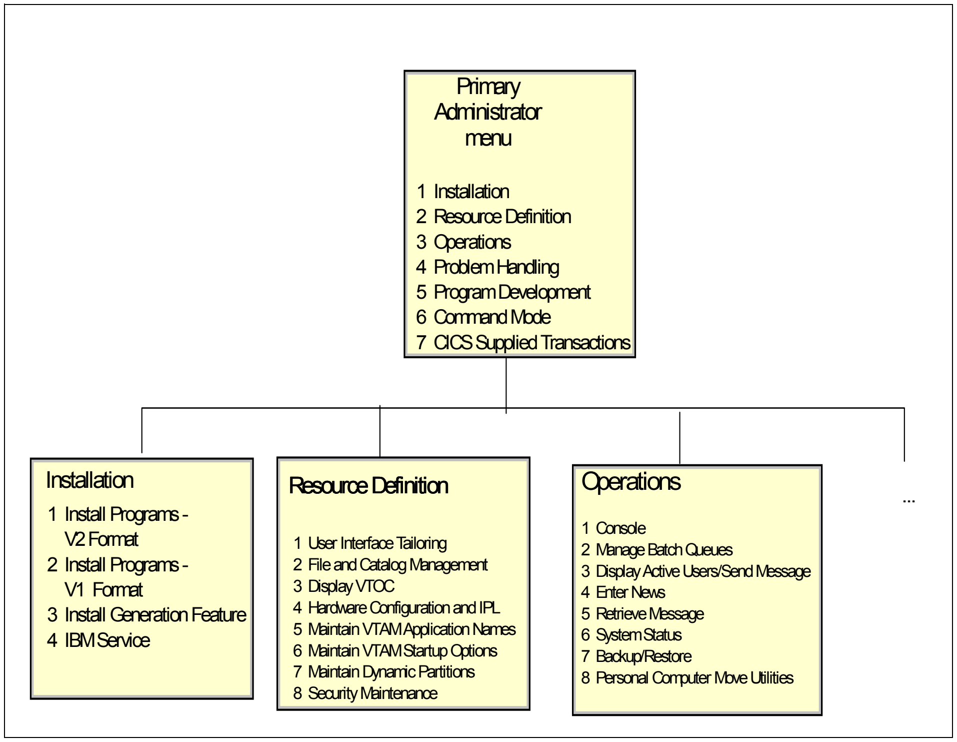

Primary Administrator menu
1 Installation
2 Resource Definition
3 Operations
4 Problem Handling
5 Program Development
6 Command Mode
7 CICS Supplied Transactions
Installation
1 Install Programs - V2 Format
2 Install Programs - V1 Format
3 Install Generation Feature
4 IBM Service
Resource Definition
1 User Interface Tailoring
2 File and Catalog Management
3 Display VTOC
4 Hardware Configuration and IPL
5 Maintain VTAM Application Names
6 Maintain VTAM Startup Options
7 Maintain Dynamic Partitions
8 Security Maintenance
Operations
1 Console
2 Manage Batch Queues
3 Display Active Users/Send Message
4 Enter News
5 Retrieve Message
6 System Status
7 Backup/Restore
8 Personal Computer Move Utilities
...

Figure 4-3 Entry menu structure for an administrator

You can navigate through the tree to do the selected task, entering the number of the next selection, or, if you know the complete path numbers, you can enter the fastpath in one.

Example: To enter the system console follow path 3 Operations, then 1 Console, or enter the fastpath 31.

4.3.1 Navigating through Interactive Interface menus

The z/VSE Interactive Interface makes it easier for you to interactively use the facilities of z/VSE and its components. Select the task that you want to perform from selection panels. A dialog requests input from you to complete the specific task.

On a selection panel you can enter your selection number one by one, or, if you know it, you may enter the fastpath that represents all selections starting from the current panel. Or you may enter the synonym for the dialog you want to select. This is a character string describing the dialog.

For example, the administrator fastpath for User Profile Maintenance is 211. The synonym is UPM.

Some PF keys used by the Interactive Interface have the same function from every panel that uses them. Other PF key functions differ for different dialogs. Each panel shows the PF keys you can use and the functions to which they correspond. When you use a PF key, review the panel you are working with to know which function the PF key represents.

Table 4-1 shows the PF keys that have the same meaning on all panels of the Interactive Interface.

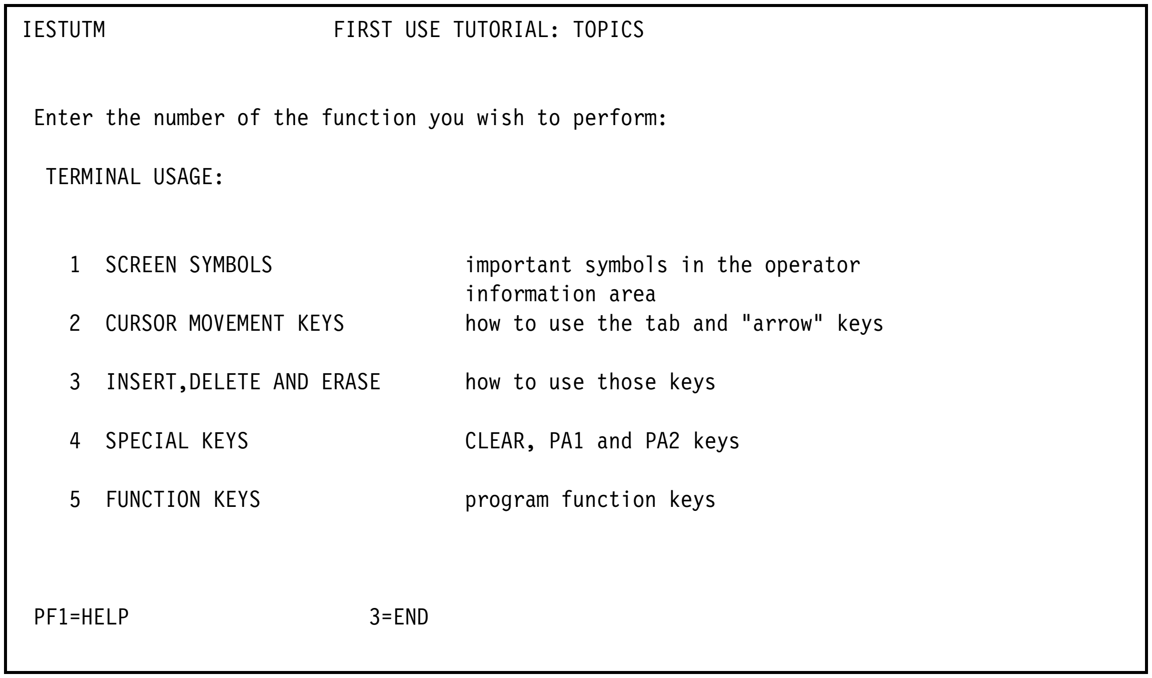
Table 4-1 Standard PF key usage

Key	Name	Function
PF1	HELP	Display help information.
PF3	END	Quit and go back on level.
PF4	RETURN	Quit and return to top panel.
PF7	BACKWARD	Scroll (move) to previous page.
PF8	FORWARD	Scroll (move) to next page.

To process the data you entered, on some dialogs PF5=PROCESS is offered. In this case you have to press the process key. If PF5=PROCESS is not offered, ENTER will process your entries and continue with the dialog.

4.3.2 Using PF1-HELP and the z/VSE tutorial

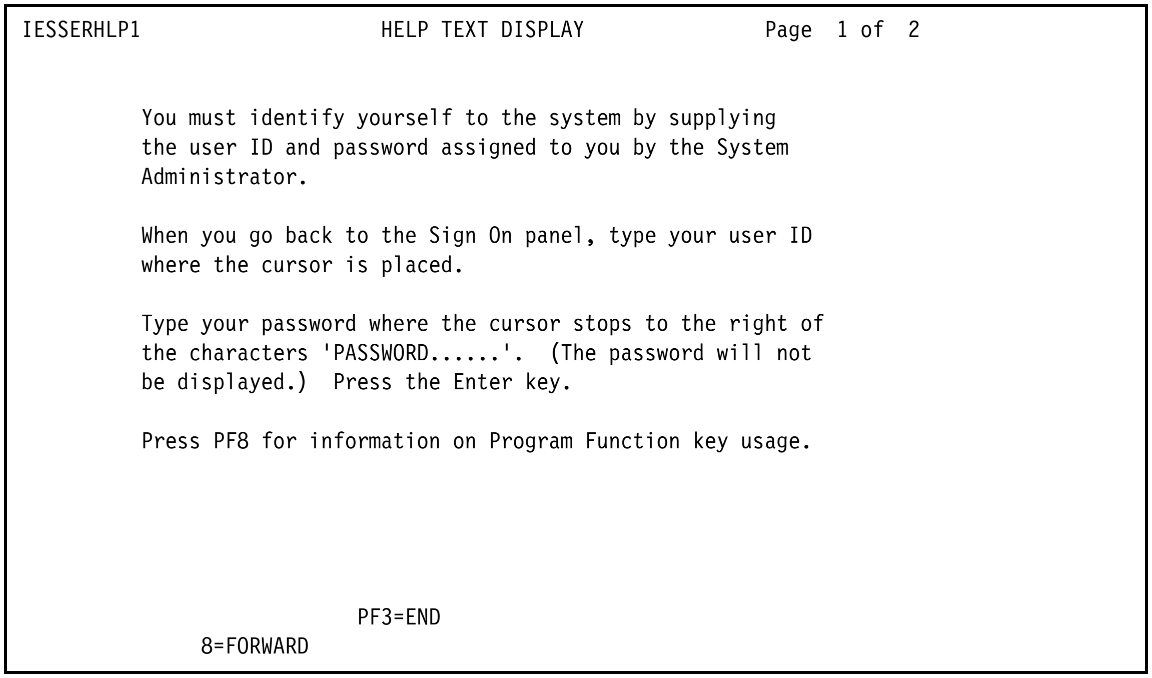
From the LOGON panel, press the PF2 TUTORIAL key to display the z/VSE tutorial. New users of z/VSE should acquaint themselves with the tutorial (see Figure 4-4) and with the extensive online help facilities of z/VSE.



```
IESTUTM                   FIRST USE TUTORIAL: TOPICS


 Enter the number of the function you wish to perform:

  TERMINAL USAGE:


    1  SCREEN SYMBOLS                important symbols in the operator
                                     information area
    2  CURSOR MOVEMENT KEYS          how to use the tab and "arrow" keys

    3  INSERT,DELETE AND ERASE       how to use those keys

    4  SPECIAL KEYS                  CLEAR, PA1 and PA2 keys

    5  FUNCTION KEYS                 program function keys



 PF1=HELP                     3=END
```

Figure 4-4 z/VSE tutorial main menu

Besides the tutorial, you can access online help from any of the z/VSE panels. Press the PF1 HELP key for explanation of the function you want to perform, or to get help in error situations. When you invoke help, you get a panel like Figure 4-5, where you can scroll through information.

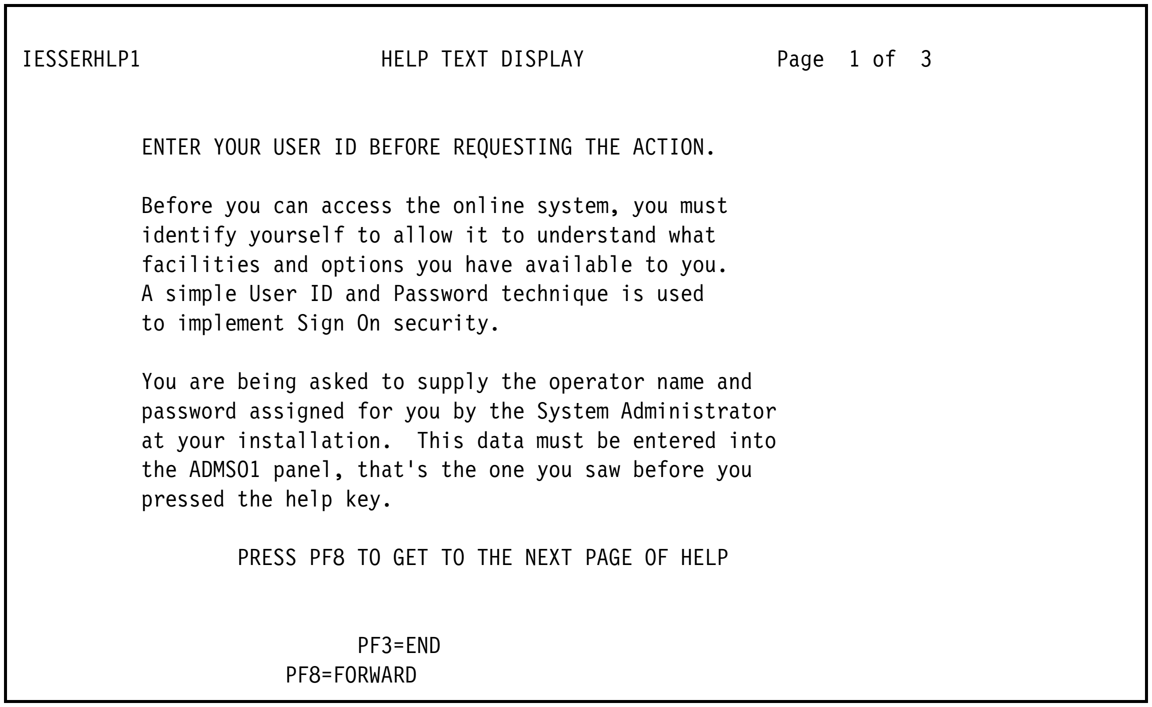


```
IESSERHLP1                   HELP TEXT DISPLAY              Page  1 of  2



          You must identify yourself to the system by supplying
          the user ID and password assigned to you by the System
          Administrator.

          When you go back to the Sign On panel, type your user ID
          where the cursor is placed.

          Type your password where the cursor stops to the right of
          the characters 'PASSWORD......'.  (The password will not
          be displayed.)  Press the Enter key.

          Press PF8 for information on Program Function key usage.






                            PF3=END
               8=FORWARD
```

Figure 4-5 Sample of a help panel

When getting an error message on a panel, pressing PF1 help may provide you with additional information about this error situation.

Example: When message `ENTER YOUR USER ID BEFORE REQUESTING THE ACTION` comes up on the LOGON screen, pressing PF1 provides the information shown in Figure 4-6.



```
IESSERHLP1                    HELP TEXT DISPLAY                Page  1 of  3


          ENTER YOUR USER ID BEFORE REQUESTING THE ACTION.

          Before you can access the online system, you must
          identify yourself to allow it to understand what
          facilities and options you have available to you.
          A simple User ID and Password technique is used
          to implement Sign On security.

          You are being asked to supply the operator name and
          password assigned for you by the System Administrator
          at your installation.  This data must be entered into
          the ADMSO1 panel, that's the one you saw before you
          pressed the help key.

                  PRESS PF8 TO GET TO THE NEXT PAGE OF HELP


                            PF3=END
                      PF8=FORWARD
```

Figure 4-6 Sample help for a dialog message

4.4 Task-oriented dialogs

The users of z/VSE have different tasks. Users with the same tasks builds a group that uses its own task-specific dialogs. Below you will get an overview of the standard tasks in z/VSE.

4.4.1 Administrator tasks

The Interactive Interface offers dialogs or skeletons for many functions the system administrator or system programmer has to perform. Some of these tasks are:

- Tailoring IPL and system startup
- Configuring hardware devices
- Tailoring the Interactive Interface

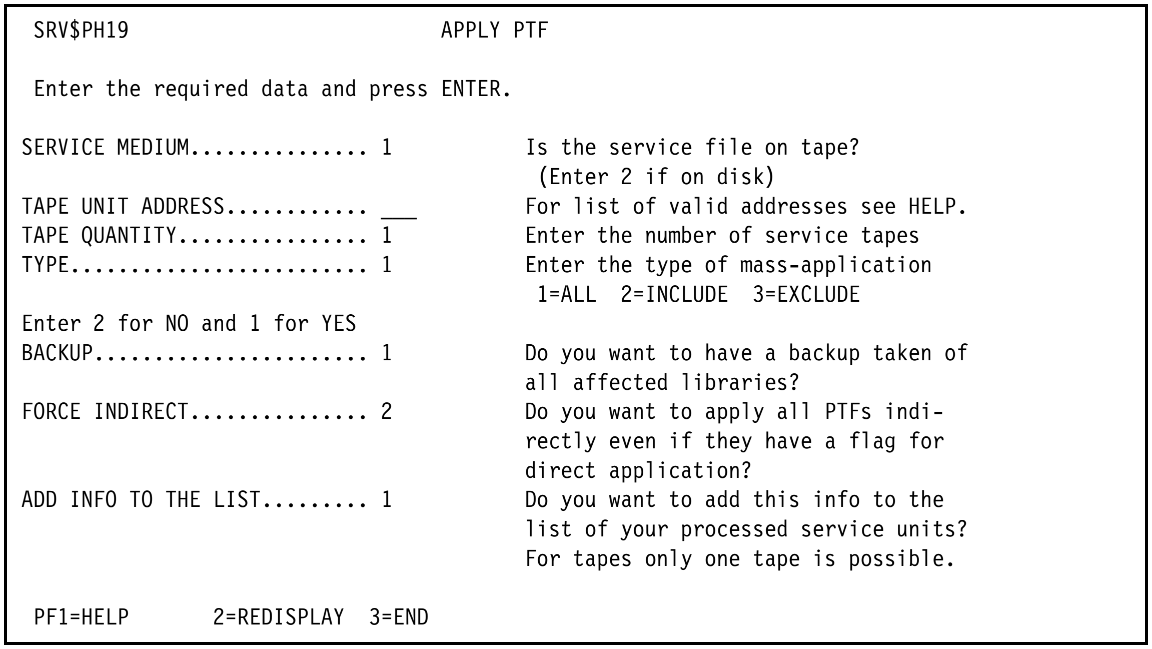
- Maintaining user IDs
- Managing VSE/VSAM files
- Applying service

As an example for administrator dialogs, the applying service task is shown below.

Applying service

In z/VSE a fix for problems is called *Program Temporary Fix* (PTF). Such a PTF can replace or add one or more parts of z/VSE.

After you have used fastpath 1423 to enter the Apply PTF dialog, the panel shown in Figure 4-7 is displayed.

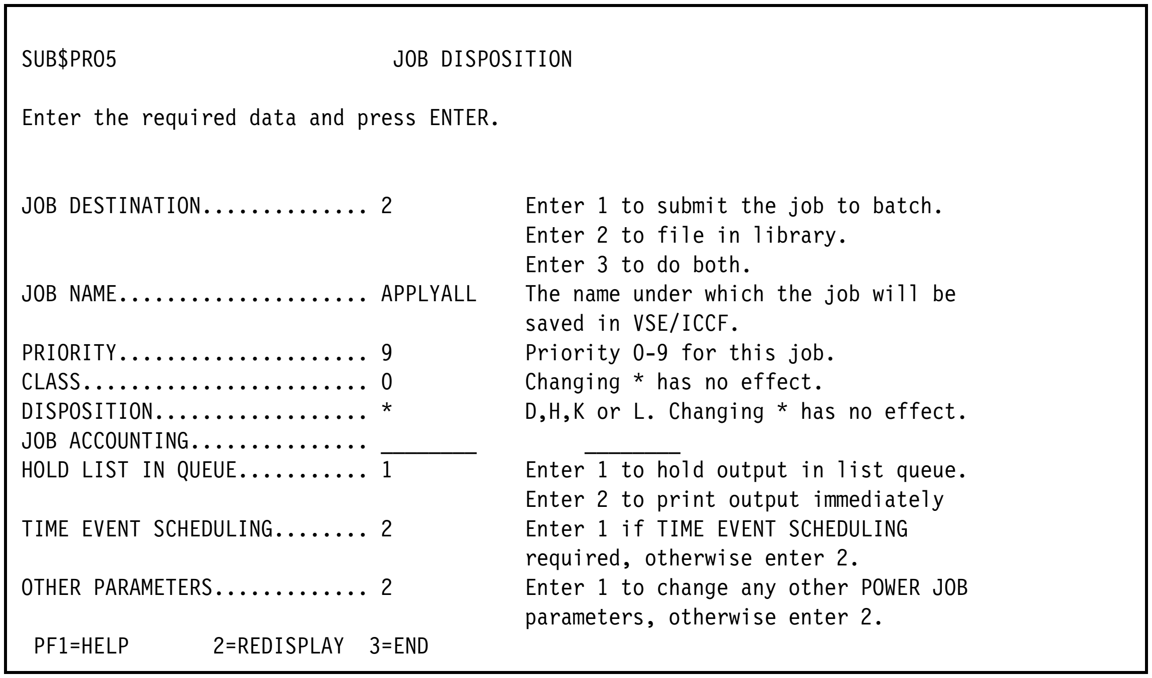


```
 SRV$PH19                          APPLY PTF

 Enter the required data and press ENTER.

SERVICE MEDIUM.............. 1          Is the service file on tape?
                                         (Enter 2 if on disk)
TAPE UNIT ADDRESS........... ___        For list of valid addresses see HELP.
TAPE QUANTITY............... 1          Enter the number of service tapes
TYPE........................ 1          Enter the type of mass-application
                                         1=ALL  2=INCLUDE  3=EXCLUDE
Enter 2 for NO and 1 for YES
BACKUP...................... 1          Do you want to have a backup taken of
                                        all affected libraries?
FORCE INDIRECT.............. 2          Do you want to apply all PTFs indi-
                                        rectly even if they have a flag for
                                        direct application?
ADD INFO TO THE LIST........ 1          Do you want to add this info to the
                                        list of your processed service units?
                                        For tapes only one tape is possible.

 PF1=HELP      2=REDISPLAY  3=END
```

Figure 4-7 APPLY PTF panel

Here you can see how useful the PF1 help key is. If you do not know the address of your tape unit, press PF1 and all valid addresses are shown.

The service will be applied in a batch job. After having supplied all needed information in the APPLY PTF panel (see Figure 4-7 on page 121), you will get the JOB DISPOSITION panel (see Figure 4-8).



```
SUB$PRO5                       JOB DISPOSITION

Enter the required data and press ENTER.


JOB DESTINATION............. 2          Enter 1 to submit the job to batch.
                                        Enter 2 to file in library.
                                        Enter 3 to do both.
JOB NAME.................... APPLYALL   The name under which the job will be
                                        saved in VSE/ICCF.
PRIORITY.................... 9          Priority 0-9 for this job.
CLASS....................... 0          Changing * has no effect.
DISPOSITION................. *          D,H,K or L. Changing * has no effect.
JOB ACCOUNTING.............. ________       ________
HOLD LIST IN QUEUE.......... 1          Enter 1 to hold output in list queue.
                                        Enter 2 to print output immediately
TIME EVENT SCHEDULING....... 2          Enter 1 if TIME EVENT SCHEDULING
                                        required, otherwise enter 2.
OTHER PARAMETERS............ 2          Enter 1 to change any other POWER JOB
                                        parameters, otherwise enter 2.
 PF1=HELP      2=REDISPLAY  3=END
```

Figure 4-8 JOB DISPOSITION panel

The source of a batch job can be modified and stored in an Interactive Computing and Control Facility (ICCF) library (see “Using the VSE/ICCF editor” on page 130).

You may enter an own job name. Then the job is filed under this name in your primary ICCF library, from where you can submit it. Or you may submit it immediately by changing the job destination field.

Use fastpath 511 to access your primary library.

Tailoring a compile skeleton

You may use the *Program Development Library* dialog to access and work with VSE/ICCF libraries. Various options allow you to create, maintain, and process library members.

One of the options allows you to compile library members. Before a programmer can use the *Compile a Member* option, the related compile skeletons must be tailored as needed for your installation.

The compile skeletons are available in VSE/ICCF library 2. They are available for various languages and for different types of programs. As an example we take C$QCVBAT, which is the skeleton for a COBOL batch program with DB2 access.

You may edit the skeleton in ICCF library 2 like this:

- Change the LIB.SUBLIB to the library, where the created phase should be cataloged.
- Check the LIBDEF search chain, so that the installed products DB2 and the language COBOL are found.

After this preparation a programmer can use this compile option.

4.4.2 Operator tasks

The primary selection panel shown in Figure 4-9 shows the tasks available for an operator.

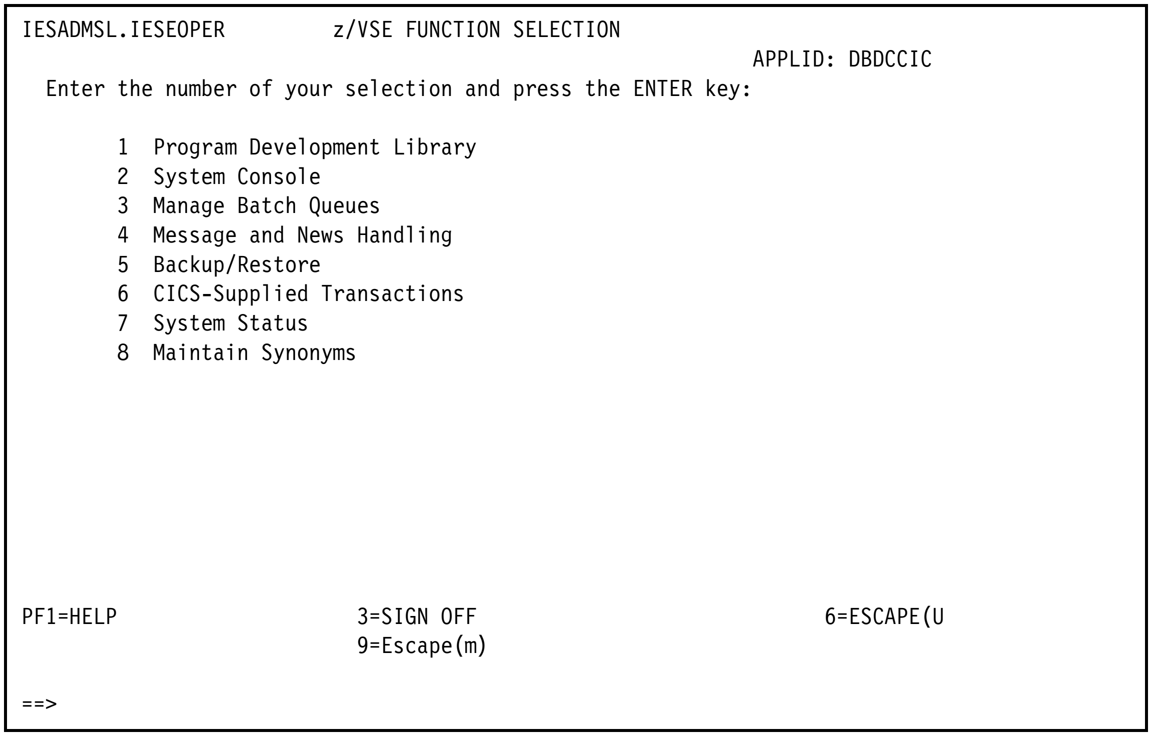

```
IESADMSL.IESEOPER         z/VSE FUNCTION SELECTION
                                                            APPLID: DBDCCIC
  Enter the number of your selection and press the ENTER key:

        1  Program Development Library
        2  System Console
        3  Manage Batch Queues
        4  Message and News Handling
        5  Backup/Restore
        6  CICS-Supplied Transactions
        7  System Status
        8  Maintain Synonyms








PF1=HELP                    3=SIGN OFF                             6=ESCAPE(U
                            9=Escape(m)

==>
```

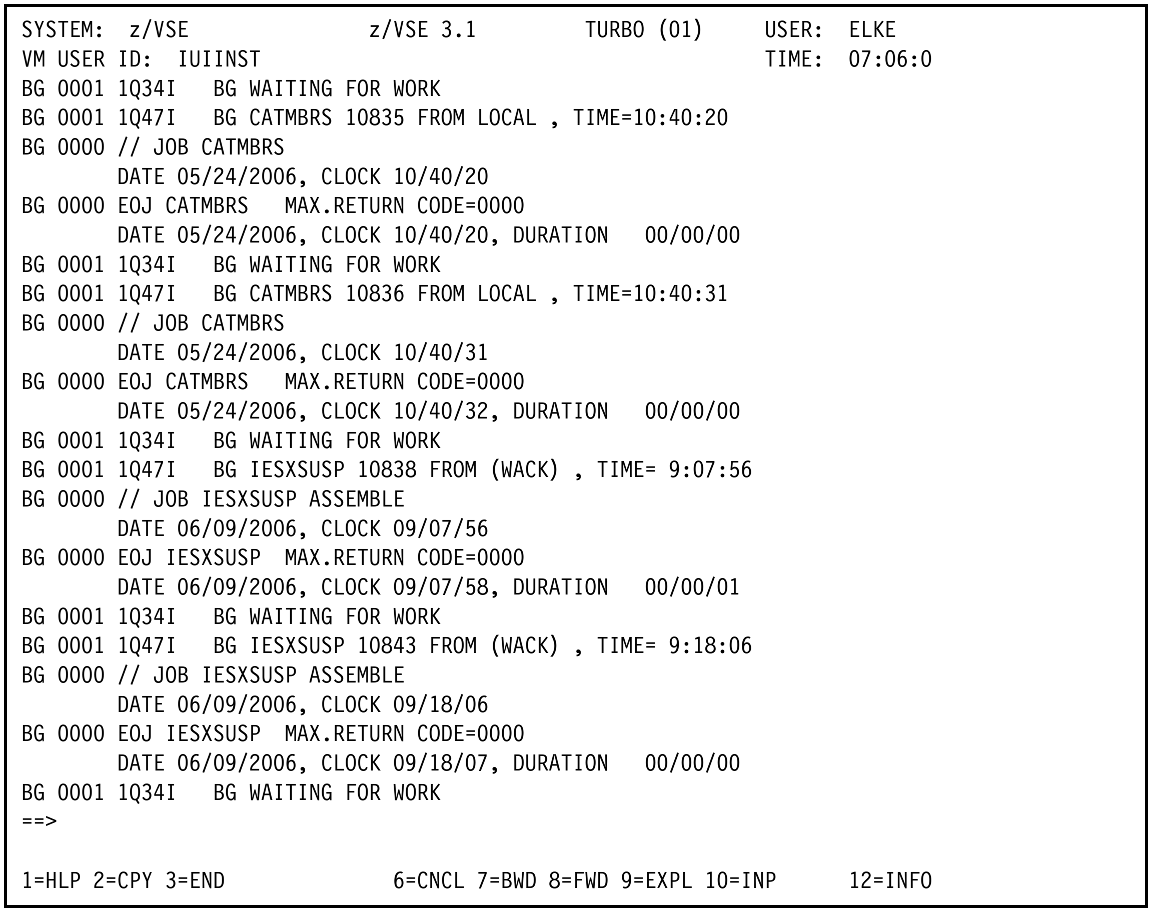
Figure 4-9 Primary selection menu for an operator

The operator mainly uses the following functions:

- Console function
- Backup and restore data function

Using console function and online message explanation

An important function of the Interactive Interface is the console function. After you entered the Interactive Interface Console using fastpath 2, you get the console screen like Figure 4-10.

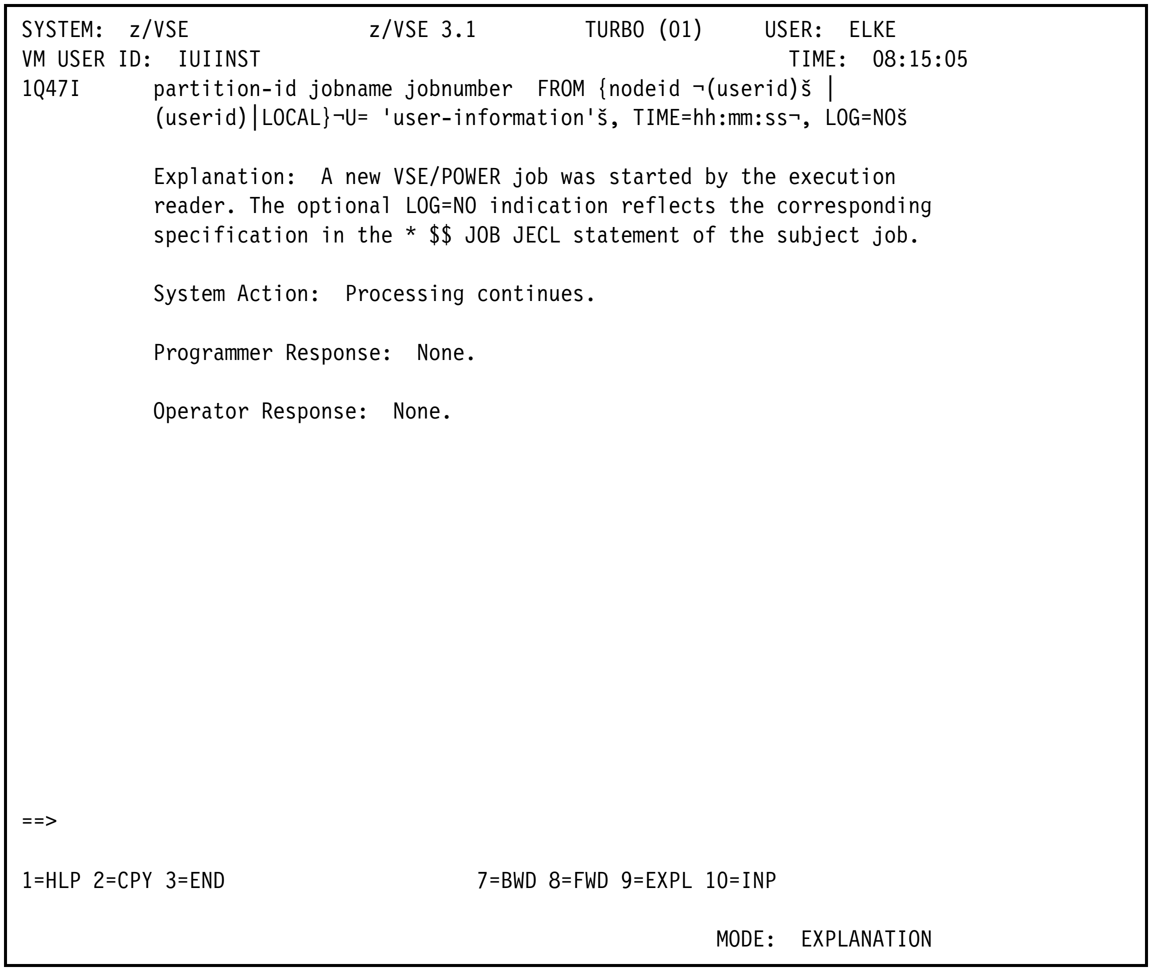


```
SYSTEM:  z/VSE              z/VSE 3.1        TURBO (01)     USER:  ELKE
VM USER ID:  IUIINST                                        TIME:  07:06:0
BG 0001 1Q34I   BG WAITING FOR WORK
BG 0001 1Q47I   BG CATMBRS 10835 FROM LOCAL , TIME=10:40:20
BG 0000 // JOB CATMBRS
        DATE 05/24/2006, CLOCK 10/40/20
BG 0000 EOJ CATMBRS   MAX.RETURN CODE=0000
        DATE 05/24/2006, CLOCK 10/40/20, DURATION   00/00/00
BG 0001 1Q34I   BG WAITING FOR WORK
BG 0001 1Q47I   BG CATMBRS 10836 FROM LOCAL , TIME=10:40:31
BG 0000 // JOB CATMBRS
        DATE 05/24/2006, CLOCK 10/40/31
BG 0000 EOJ CATMBRS   MAX.RETURN CODE=0000
        DATE 05/24/2006, CLOCK 10/40/32, DURATION   00/00/00
BG 0001 1Q34I   BG WAITING FOR WORK
BG 0001 1Q47I   BG IESXSUSP 10838 FROM (WACK) , TIME= 9:07:56
BG 0000 // JOB IESXSUSP ASSEMBLE
        DATE 06/09/2006, CLOCK 09/07/56
BG 0000 EOJ IESXSUSP  MAX.RETURN CODE=0000
        DATE 06/09/2006, CLOCK 09/07/58, DURATION   00/00/01
BG 0001 1Q34I   BG WAITING FOR WORK
BG 0001 1Q47I   BG IESXSUSP 10843 FROM (WACK) , TIME= 9:18:06
BG 0000 // JOB IESXSUSP ASSEMBLE
        DATE 06/09/2006, CLOCK 09/18/06
BG 0000 EOJ IESXSUSP  MAX.RETURN CODE=0000
        DATE 06/09/2006, CLOCK 09/18/07, DURATION   00/00/00
BG 0001 1Q34I   BG WAITING FOR WORK
==>

1=HLP 2=CPY 3=END             6=CNCL 7=BWD 8=FWD 9=EXPL 10=INP      12=INFO
```

Figure 4-10 Console screen

The PF1 help key provides you with important information about the console functions.

A main function is the PF9 Explain key. Position the cursor under a message you received, then press PF9. The online message explanation is then displayed, as shown in Figure 4-11.



```
SYSTEM:  z/VSE               z/VSE 3.1        TURBO (01)     USER:  ELKE
VM USER ID:  IUIINST                                           TIME:  08:15:05
1Q47I      partition-id jobname jobnumber  FROM {nodeid ¬(userid)š |
           (userid)|LOCAL}¬U= 'user-information'š, TIME=hh:mm:ss¬, LOG=NOš

           Explanation:  A new VSE/POWER job was started by the execution
           reader. The optional LOG=NO indication reflects the corresponding
           specification in the * $$ JOB JECL statement of the subject job.

           System Action:  Processing continues.

           Programmer Response:  None.

           Operator Response:  None.













==>

1=HLP 2=CPY 3=END                     7=BWD 8=FWD 9=EXPL 10=INP

                                                               MODE:  EXPLANATION
```

Figure 4-11 Online message explanation

Backup and restore VSE libraries

At various times the operator should make backups of the system.

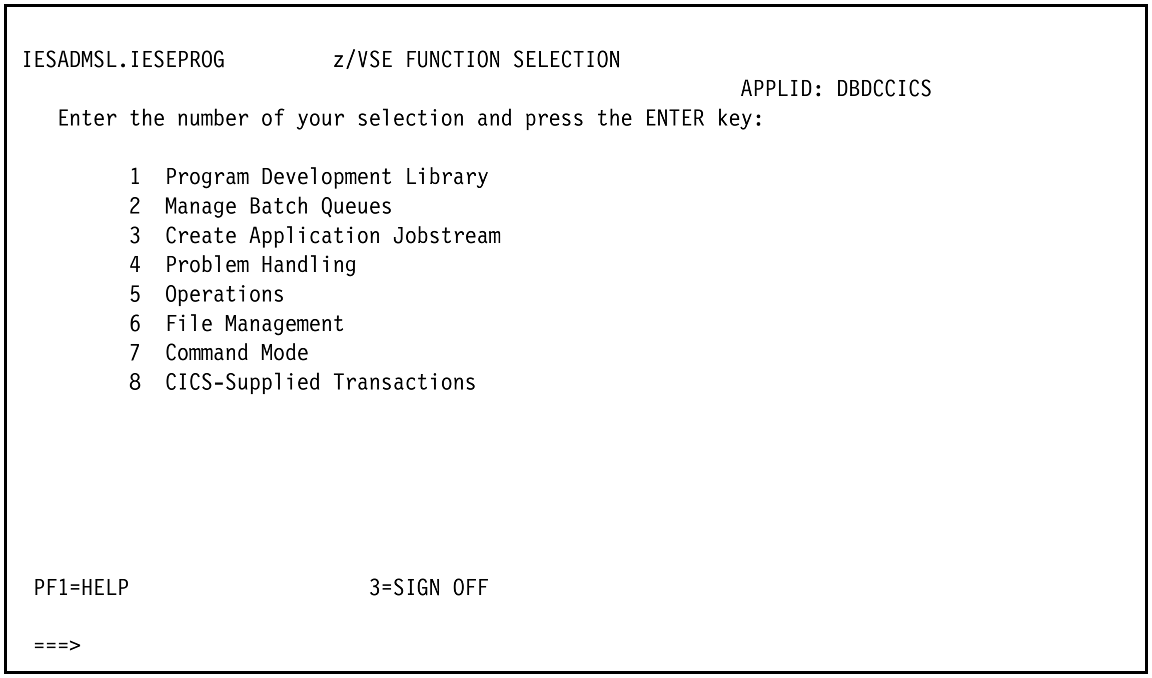
When you back up data, you save a copy of the data on magnetic tape or disk. If the data is later damaged on your system, you can restore the data to the system by reading the copy from the tape or disk. z/VSE uses various data and library formats. Because of this, z/VSE also offers different backup and restore programs (or dialogs) that are designed to fit those different formats. As an example, here the backup and restore of VSE libraries are shown.

The Backup VSE Library on Tape dialog backs up VSE libraries from disk to tape. You can back up libraries, sublibraries, or members.

Use fastpath 521 to enter the Backup VSE libraries Dialog and fastpath 522 to enter the Restore VSE libraries Dialog.

4.4.3 Important tasks for a programmer

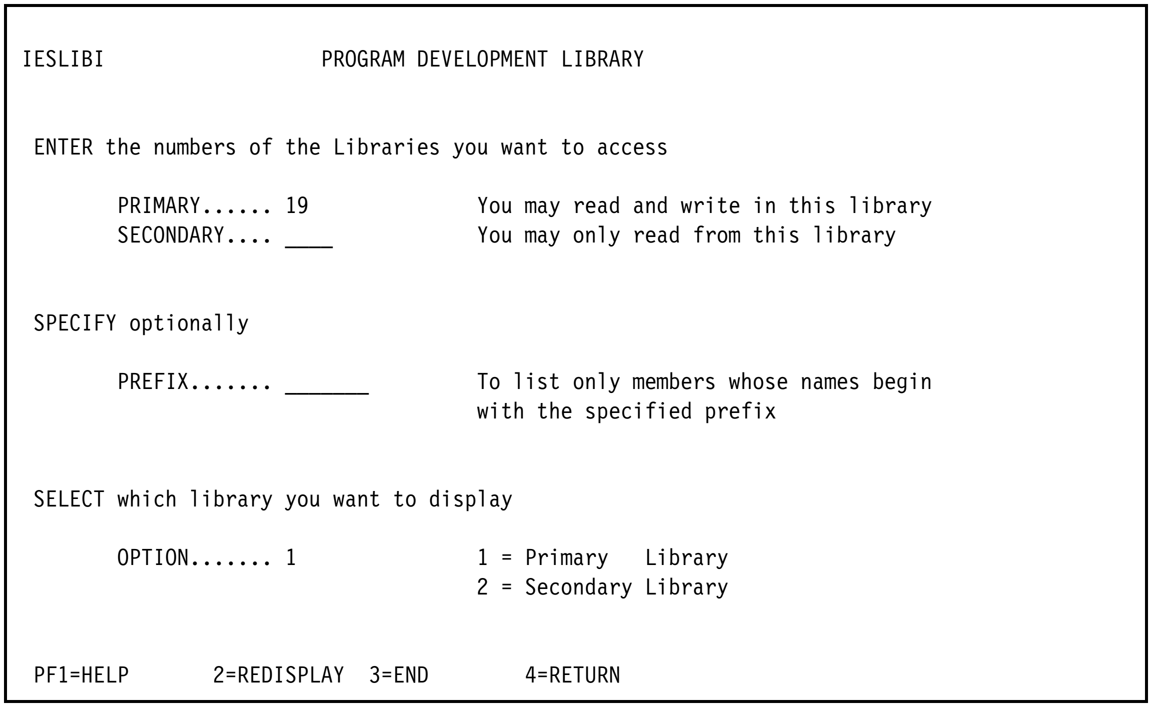
The primary selection panel in Figure 4-12 shows the tasks for a programmer.



```
IESADMSL.IESEPROG         z/VSE FUNCTION SELCTION
                                                            APPLID: DBDCCICS
   Enter the number of your selection and press the ENTER key:

         1  Program Development Library
         2  Manage Batch Queues
         3  Create Application Jobstream
         4  Problem Handling
         5  Operations
         6  File Management
         7  Command Mode
         8  CICS-Supplied Transactions





 PF1=HELP                    3=SIGN OFF

 ===>
```

Figure 4-12 Primary selection menu for a programmer

For example, a programmer develops software and enters 1 in her primary selection menu to select the program development functions. She gets the PROGRAM DEVELOPMENT LIBRARY panel (see Figure 4-13).



```
IESLIBI                  PROGRAM DEVELOPMENT LIBRARY


 ENTER the numbers of the Libraries you want to access

        PRIMARY...... 19              You may read and write in this library
        SECONDARY.... ____            You may only read from this library


 SPECIFY optionally

        PREFIX....... _______         To list only members whose names begin
                                      with the specified prefix


 SELECT which library you want to display

        OPTION....... 1               1 = Primary   Library
                                      2 = Secondary Library


 PF1=HELP       2=REDISPLAY  3=END        4=RETURN
```

Figure 4-13 PROGRAM DEVELOPMENT LIBRARY panel

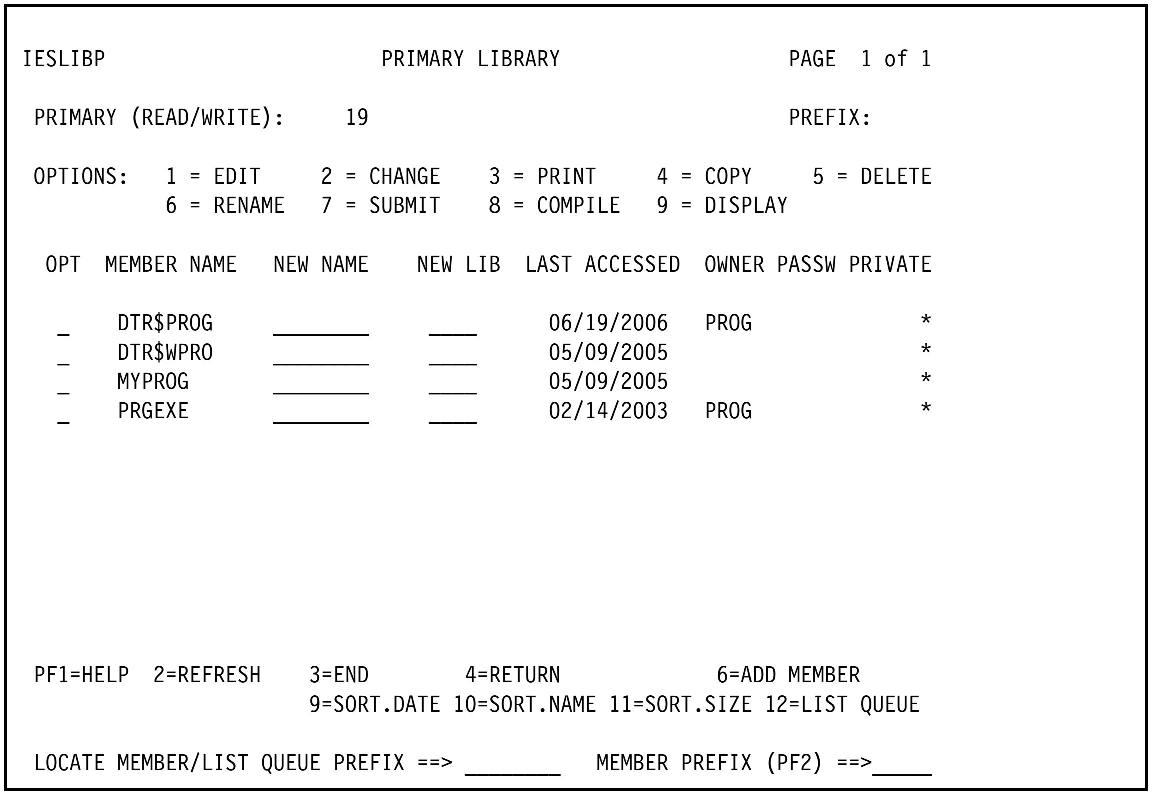
Working with the program development library

The program development library is an ICCF library[1]. In the PROGRAM DEVELOPMENT LIBRARY panel you specify which ICCF library you want to use for your program development.

You can access a library as either a primary or secondary library in your search chain. We recommend specifying a library as a SECONDARY library and setting SELECT OPTION 2 if this library is not your default library. This allows you to create a work copy at your default library without modifying the original member.

[1] More about the concept of ICCF libraries can be found in 5.6.2, “What is a VSE/ICCF library?” on page 153.

The panel in Figure 4-13 on page 128 shows your default primary library (as defined in your user profile) in the PRIMARY field. To access your default primary library, you simply have to press Enter. A new panel will be displayed that shows a list of all your members in your library, as in Figure 4-14.



```
IESLIBP                      PRIMARY LIBRARY                  PAGE  1 of 1

 PRIMARY (READ/WRITE):     19                                 PREFIX:

 OPTIONS:   1 = EDIT     2 = CHANGE    3 = PRINT     4 = COPY     5 = DELETE
            6 = RENAME   7 = SUBMIT    8 = COMPILE   9 = DISPLAY

  OPT  MEMBER NAME   NEW NAME    NEW LIB  LAST ACCESSED  OWNER PASSW PRIVATE

   _    DTR$PROG     ________     ____      06/19/2006   PROG             *
   _    DTR$WPRO     ________     ____      05/09/2005                    *
   _    MYPROG       ________     ____      05/09/2005                    *
   _    PRGEXE       ________     ____      02/14/2003   PROG             *









 PF1=HELP  2=REFRESH    3=END       4=RETURN            6=ADD MEMBER
                        9=SORT.DATE 10=SORT.NAME 11=SORT.SIZE 12=LIST QUEUE

 LOCATE MEMBER/LIST QUEUE PREFIX ==> ________   MEMBER PREFIX (PF2) ==>_____
```

Figure 4-14 PRIMARY LIBRARY member selection list

The members in the list are sorted by date in descending order. Members with the same date appear by name in alphabetical order. Besides the member name, the list shows the following attributes for each member:

- LAST ACCESSED: This shows when the member was last edited.
- OWNER: This is the user ID of the owner of the member. For users with programmer (type 2) profiles, the user ID is displayed only for the user's own members.
- PASSW: If the member is password protected, an asterisk (*) is displayed.
- PRIVATE: If the member has the private attribute, an asterisk (*) is displayed.

The options you can choose are at the top of the list. Enter the option number in the OPT column to the left of the members that you want to process. The options differ for primary and secondary libraries.

Important options are:

- 1 - Edit the member.
- 7 - Submit the member to the reader queue.
- 8 - Compile the member.
- 9 - Display (browse) the member.

Important PF keys are:

- PF10 - Sort by name.
- PF12 - Display the VSE/POWER List queue.

To continue with the program development, the programmer inserts a 1 in front of the member he would like to edit. This member will be opened by the VSE/ICCF editor.

Using the VSE/ICCF editor

When developing a program at the z/VSE host you need to store your program file in a library. Under z/VSE, this is normally the VSE/ICCF library.

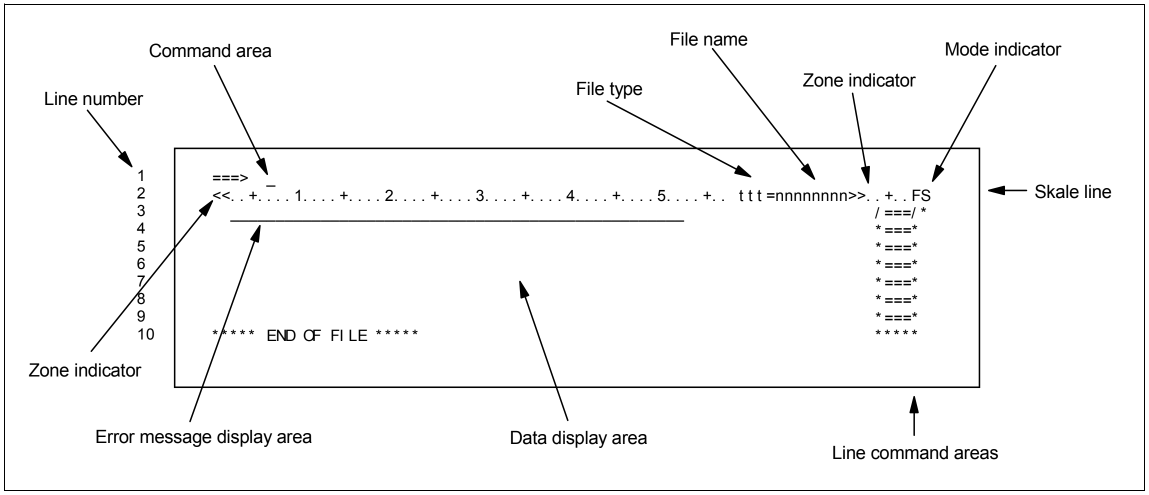
VSE/ICCF has two editors:

- A *context editor*, which was designed for typewriter terminals. This editor must be used in procedures and macros and in the DTSBATCH utility. The editor works on one line at a time—the line that is determined by locating a given string.
- A *full-screen editor* for IBM 3270 and similar display terminals. This editor utilizes the entire screen. It allows you to modify data simply by positioning the cursor at a point on the screen and typing the new data. Then, by entering a command or pressing a key, the modification is applied to the file. The file is called the VSE/ICCF library with the file name DTSFILE. It resides on a DASD device. The file access method is DA (direct access).

The full-screen editor is a disk editor. Any update is immediately written to the file, which means that there is no QUIT available as other full-screen editors normally provide. A big drawback is the absence of block commands for the line commands like DD for delete or CC for copy.

The full-screen editor is invoked by entering option 1 in the Interactive Interface panel IESLIBP (see Figure 4-14 on page 129) or by using the ED macro from the VSE/ICCF command line.

The basic screen layout, the default, is shown in Figure 4-15. The file being used for this example contains only blank lines. You can deviate from the default layout by entering the VERIFY command with suitable operands.



```
1   ===>  _
2   <<. . +. . . . 1. . . . +. . . . 2. . . . +. . . . 3. . . . +. . . . 4. . . . +. . . . 5. . . . +. .  t t t =nnnnnnnn>>. . +. . FS
3                                                                                                    / ===/ *
4                                                                                                    * ===*
5                                                                                                    * ===*
6                                                                                                    * ===*
7                                                                                                    * ===*
8                                                                                                    * ===*
9                                                                                                    * ===*
10  ***** END OF FILE *****                                                                          *****
```

Figure 4-15 ICCF editor screen layout

The various screen areas, as shown in Figure 4-15, are discussed below:

- Command area: Line 1 constitutes the command area. This area, identified by the characters ===>, normally consists of one line. Through the VERIFY command, you may expand this area to up to four lines. Any combination of editor commands separated by logical line end characters can be entered in the command area.
- Scale line: The scale line immediately follows the command area. It contains column indicators as an aid to working with fixed format data. It displays the following information:
  - A pair of two arrow heads designating the editing zone.
  - Error message, if any. A message issued by the editor temporarily overlays the first 52 columns of the scale line. If no error message is displayed and a display offset is in effect (see the LEFT and RIGHT editor commands in the reference section of this chapter), the offset amount will be displayed in the message display area.
  - File type. The file type ('ttt' in Figure 4-15) can be any of the following:
    - INP - the input area
    - MEM - to indicate one of the following: a library member, the print area $$PRINT, the punch area $$PUNCH.

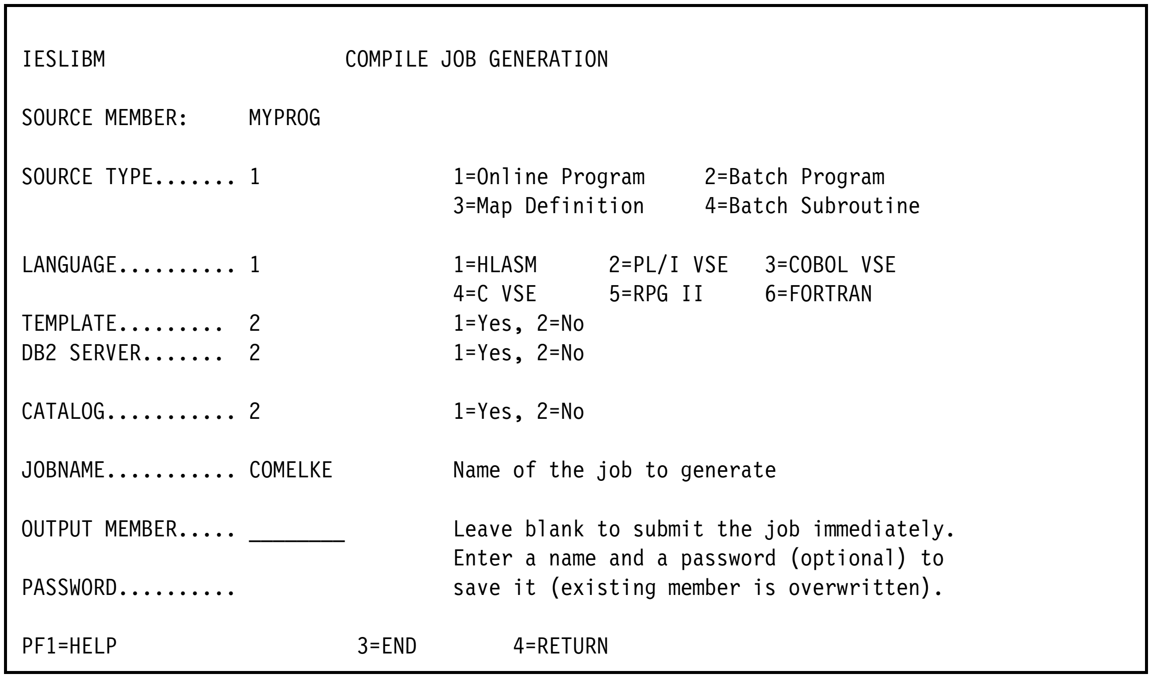
        - NEW - a file opened via the ENTER command without operand (that is, a file being newly created)
    - File name.
    - Mode indicator.
    - Throughout the edit session, this indicator appears as FS for full-screen edit mode. If mixed data are edited, the FS indicator is preceded by DB.
- Data display area: This is the area where the individual records are displayed. Data displayed in this area of the screen can be modified simply by positioning the cursor and typing in the new data. This way, you can alter several lines before pressing Enter.
- The current line record is identified by highlighting and, if the NUMBERS option is set off, the string /===/* in the line command area (see the next paragraph).
- Line command area: Following each record displayed on the screen is a line command area. In this area, you can enter line commands to manipulate individual lines or groups of lines on the screen. Line commands are used to add, delete, duplicate, shift, move, or copy lines. Line command areas are indicated by the strings *===*, /===/*, and /***/. You enter line commands by overlaying these characters in the line command area.

After all changes to the program source are done, exit the editor via the PF3 key. This returns you to the member list of your library, as shown in Figure 4-14 on page 129.

In the next step you could compile your program.

Compile a COBOL program

Assume that the program in the primary library member MYPROG is a COBOL program with DB2 access. To compile this program, put option 8 = Compile next to MYPROG in the primary library panel (see Figure 4-14 on page 129) and press Enter. The panel in Figure 4-16 is displayed.



```
IESLIBM                   COMPILE JOB GENERATION

SOURCE MEMBER:     MYPROG

SOURCE TYPE....... 1                1=Online Program     2=Batch Program
                                    3=Map Definition     4=Batch Subroutine

LANGUAGE.......... 1                1=HLASM      2=PL/I VSE   3=COBOL VSE
                                    4=C VSE      5=RPG II     6=FORTRAN
TEMPLATE.........  2                1=Yes, 2=No
DB2 SERVER.......  2                1=Yes, 2=No

CATALOG........... 2                1=Yes, 2=No

JOBNAME........... COMELKE          Name of the job to generate

OUTPUT MEMBER..... ________         Leave blank to submit the job immediately.
                                    Enter a name and a password (optional) to
PASSWORD..........                  save it (existing member is overwritten).

PF1=HELP                   3=END        4=RETURN
```

Figure 4-16 Using COMPILE JOB GENERATION

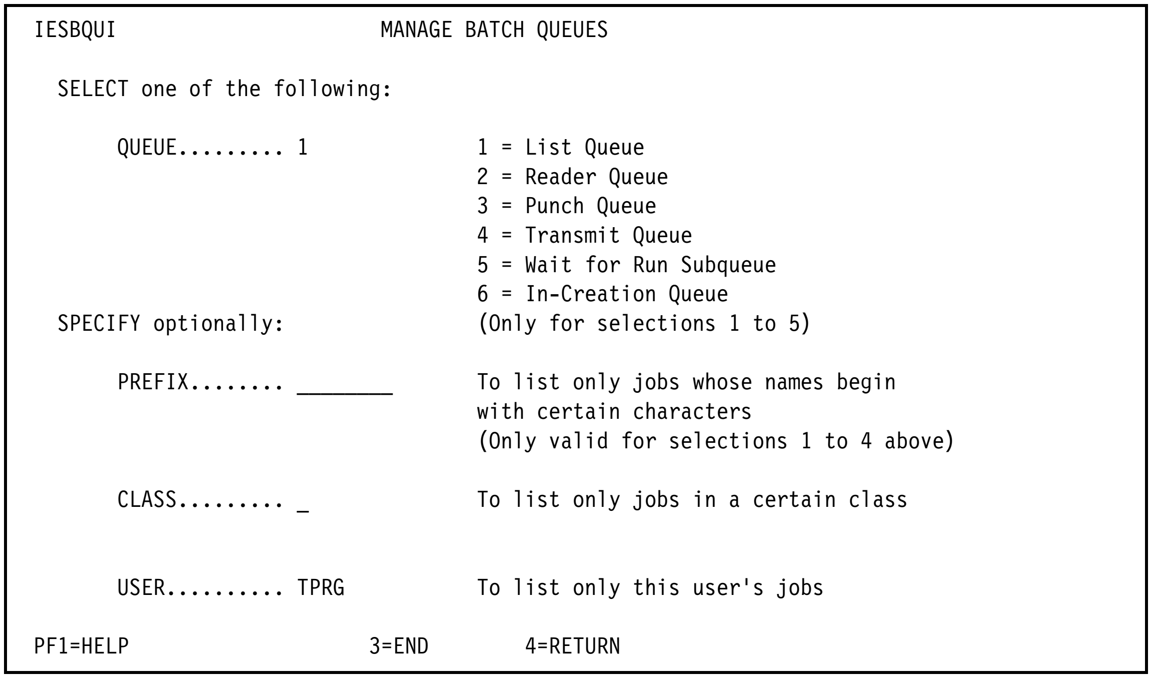
Now enter the correct data: 2=Batch for Source Type, Language is 3 COBOL, 1=YES for DB2 Server, and for output member enter a name like MYOUT. The following panel will ask you for DB2 Server preprocessor options. After having finished the dialog, you can check MYOUT, then submit the job via option 7 in the primary library panel (see Figure 4-14 on page 129).

To control the result of the your batch job you have to view the VSE/POWER list queue[2].

[2] POWER is the spooling system in z/VSE. For more details see Chapter 7, “Batch processing and VSE/POWER” on page 177.

Accessing the POWER list queue

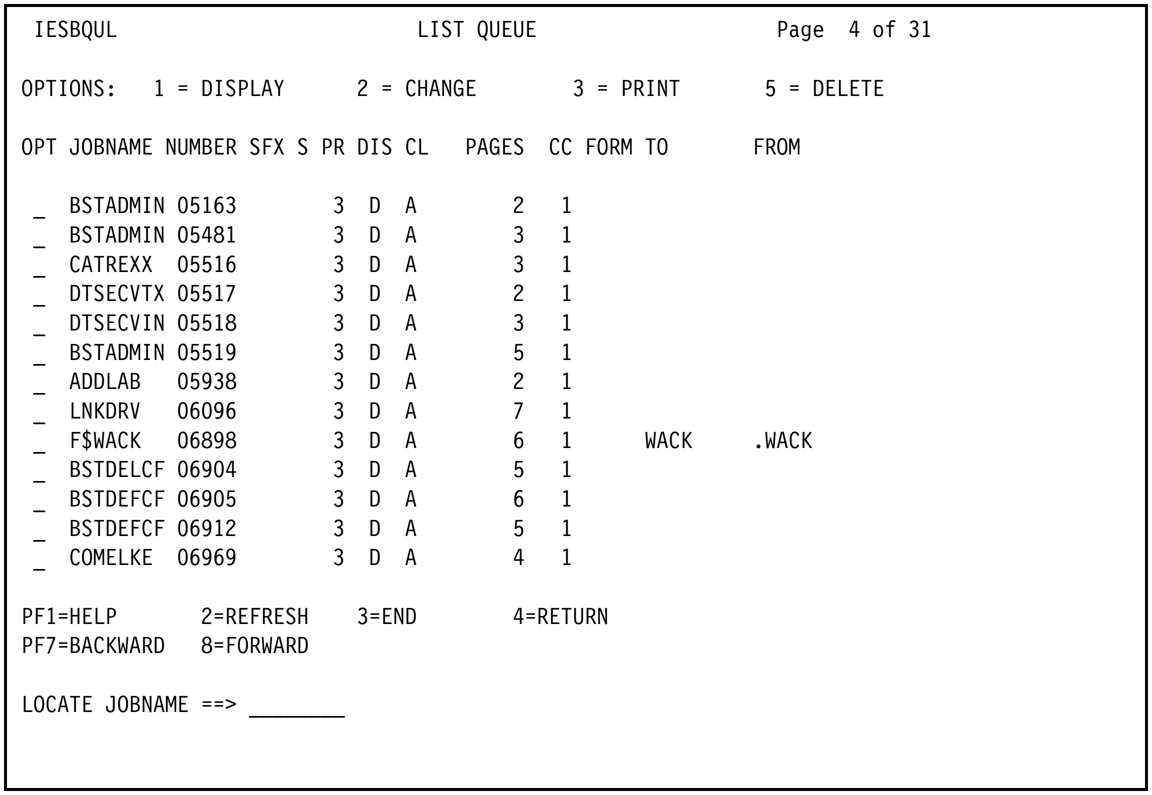
From the primary selection panel (see Figure 4-12 on page 127) select the Manage Batch Queues dialog. As a programmer you may use fastpath 21 to display the MANAGE BATCH QUEUE panel, as shown in Figure 4-17.



```
IESBQUI                      MANAGE BATCH QUEUES

  SELECT one of the following:

       QUEUE......... 1             1 = List Queue
                                    2 = Reader Queue
                                    3 = Punch Queue
                                    4 = Transmit Queue
                                    5 = Wait for Run Subqueue
                                    6 = In-Creation Queue
  SPECIFY optionally:               (Only for selections 1 to 5)

       PREFIX........ ________      To list only jobs whose names begin
                                    with certain characters
                                    (Only valid for selections 1 to 4 above)

       CLASS......... _             To list only jobs in a certain class


       USER.......... TPRG          To list only this user's jobs

PF1=HELP                    3=END        4=RETURN
```

Figure 4-17   MANAGE BATCH QUEUES panel

The panel in Figure 4-17 on page 134 shows the different VSE/POWER queues. You know that the output of your compile job will be in the list queue. Therefore you may select 1 to display the VSE/POWER List queue panels, as shown in Figure 4-18.



```
 IESBQUL                         LIST QUEUE                     Page  4 of 31

OPTIONS:   1 = DISPLAY      2 = CHANGE       3 = PRINT       5 = DELETE

OPT JOBNAME NUMBER SFX S PR DIS CL   PAGES  CC FORM TO       FROM

 _  BSTADMIN 05163        3  D  A        2   1
 _  BSTADMIN 05481        3  D  A        3   1
 _  CATREXX  05516        3  D  A        3   1
 _  DTSECVTX 05517        3  D  A        2   1
 _  DTSECVIN 05518        3  D  A        3   1
 _  BSTADMIN 05519        3  D  A        5   1
 _  ADDLAB   05938        3  D  A        2   1
 _  LNKDRV   06096        3  D  A        7   1
 _  F$WACK   06898        3  D  A        6   1      WACK     .WACK
 _  BSTDELCF 06904        3  D  A        5   1
 _  BSTDEFCF 06905        3  D  A        6   1
 _  BSTDEFCF 06912        3  D  A        5   1
 _  COMELKE  06969        3  D  A        4   1

PF1=HELP       2=REFRESH    3=END        4=RETURN
PF7=BACKWARD   8=FORWARD

LOCATE JOBNAME ==> ________
```

Figure 4-18 POWER LIST QUEUE panel

4.5 Summary

The Interactive Interface allows users to log on to z/VSE. It is mainly a menu-driven interface for user interaction with a z/VSE system.

The Interactive Interface provides three default users for the different user types, and a lot of dialogs to help perform the necessary tasks.

Key terms in this chapter		
Interactive Interface	3270 emulation	Online message explanation
Interactive computing and control facility (ICCF)	Sign-on/logon	Panels
Administrator	Programmer	Operator
SYSA	PROG	OPER

4.6 Questions for review

To help test your understanding of the material in this chapter, complete the following questions:

1. If you want more information about a specific z/VSE panel or help with a user error, what should be your first action?
2. If an error comes up when performing a function, where can you find more information about the problem?
3. Can IBM-provided panels of z/VSE be customized?
4. Which user IDs does z/VSE provide? What are the differences between them?

4.7 Exercises

The lab exercises in this chapter will help you develop skills in using z/VSE and the Interactive Interface. These skills are required for performing lab exercises in the remainder of this text.

To perform the lab exercises, each student or team will require a z/VSE user ID and password (for assistance, see your instructor).

The lab exercises are intended to teach z/VSE through trial and error, so mistakes are expected. Do not get frustrated. Instead, discuss the exercises with your classmates and work through them in teams, as needed.

The exercises teach the following:

- Logging on to z/VSE
- Using the z/VSE console and its functions
- Using the ICCF editor

- Be aware that your user ID is of type administrator.

4.7.1 Logging on to z/VSE

Establish a 3270 connection with z/VSE using a workstation 3270 emulator.

From the LOGON screen do the following:

1. Press PF2=TUTORIAL and read some of the text.
2. Press PF1=HELP and inform yourself about the needed input.
3. LOGON to z/VSE.

4.7.2 Navigating through some provided dialogs

The console

From the Primary Selection Menu do the following:

1. Enter Operations.
2. Enter the Console. What is the fastpath and the synonym?
3. Use the message explanation function.
4. Display only messages provided by partition BG.
5. Enter a command `D DYNC` to display the Dynamic Class Table (see "Static and dynamic partitions" on page 182).

Display a VTOC

See 5.4.3, "What is a VTOC?" on page 144, for a description of the VTOC.

From the Primary Selection Menu do the following:

1. Go to Resource Definition, then Display VTOC.
2. From the list of available volumes select the system volume **DOSRES** and display the volume layout.
3. Display the volume layout for SYSWK1.

Display a VSAM catalog

See 5.5.1, "What is a VSAM catalog?" on page 146, for a description of the VSAM catalog.

From the Primary Selection Menu do the following:

1. Go to Resource Definition, then File and Catalog Management.
2. The VSAM master catalog is already entered as the default. Just press Enter to display the files defined in the VSAM master catalog.

3. Repeat the steps for VSESP.USER.CATALOG.

4.7.3 Using the ICCF editor

From the Primary Selection Menu do the following:

1. Go to Program Development, then Program Development Library.
2. Select your primary library.
3. Create your first member (PF6).
4. Enter `I` for input mode.
5. Enter your test data.

5

Working with files

Objective: In working with the z/VSE operating system, you must understand how to work with its files. On IBM mainframe computers files were initially and mostly still referred to as *data sets*. These files can contain programs or data. The characteristics of traditional z/VSE files differ considerably from the file systems used in UNIX and PC systems.

After completing this chapter, you will be able to:

- Explain what a data set is.
- Describe file naming conventions and record formats.
- List some access methods for managing data and programs.
- Explain what catalogs and VTOCs are used for.
- Create, delete, and modify files.
- Explain the differences between UNIX file systems and z/VSE files.

5.1 What is a data set?

Along with other IBM mainframe operating systems, z/VSE manages data by means of *data sets*. The term *data set* refers to a file that contains one or more records. Any named group of records is called a data set. Data sets can hold information such as medical records or insurance records, to be used by a program running on the system. Data sets are also used to store information needed by applications or the operating system itself, such as source programs, macro libraries, or system variables or parameters. For data sets that contain readable text, you can print them or display them on a console (many data sets contain load modules or other binary data that is not really printable). For the rest of this chapter the word *file* will be used as opposed to IBM mainframe term *data set*.

Data set
A collection of logically related data records, such as a library of macros or a source program

In simplest terms, a *record* is a fixed number of bytes containing data. Often, a record collects related information that we treat as a unit, such as one item in a database or personnel data about one member of a department. The term *field* refers to a specific portion of a record used for a particular category of data, such as an employee's name or department.

The record is the basic unit of information used by a program running on z/VSE[1]. The records in a file can be organized in various ways, depending on how we plan to access the information. If you write an application program that processes things like personnel data, for example, your program can define a record format for each person's data.

During the long history of IBM mainframe operating systems many different methods have been developed to organize and access files. z/VSE supports sequential files, VSE library files, VSE/ICCF library files, and VSAM files.

In a *sequential file*, records are data items that are stored consecutively. To retrieve the tenth item in the data set, for example, the system must first pass the preceding nine items. Data items that must all be used in sequence, like the alphabetical list of names in a classroom roster, are best stored in a sequential data set.

A VSE library file contains one VSE library. The VSE library consists of *sublibraries* and *members*. A sublibrary holds the addresses of their members and thus makes it possible for programs or the operating system to access each member directly. Each member consists of the data, which can be a sequential string or a number of records with a fixed length. Members of these libraries typically contain data that must be made available to the system in order to run,

[1] UNIX files are different from the typical z/VSE data set because they are unstructured streams of bytes rather than organized in various logical records.

such as programs or procedures. The program that services, copies, and provides access to system and private VSE libraries is know as the *librarian*.

The VSE/ICCF library file contains all VSE/ICCF libraries. Each VSE/ICCF library consists of *members*. A member can contain programs, data, object decks, procedures, VSE/ICCF job streams, documentation, or any combination of these. Integrated into the z/VSE interactive system, ICCF also provides the ability to manage and edit files.

In a Virtual Storage Access Method (VSAM) key sequenced file (KSDS) records are data items that are stored with control information (keys) so that the system can retrieve an item without searching all preceding items in the data set. VSAM KSDS data sets are ideal for data items that are used frequently and in an unpredictable order. In general, VSAM is an access method for the indexed or sequential processing of records on direct access devices. Records can be of fixed-length or variable-length. VSAM is the preferred file management system for VSE environments.

We discuss the functionality provided by VSE/VSAM, the librarian, and the VSE/ICCF library later in this chapter.

5.2 Where are files stored?

z/VSE supports many different devices for data storage. Disks or tape are most frequently used for storing files on a long-term basis. Disk drives are known as *direct access storage devices (DASD)* because, although some files on them might be stored sequentially, these devices can handle direct access. Tape drives are known as sequential access devices because files on tape must be accessed sequentially.

The term *DASD* applies to disks or simulated equivalents of disks. All types of files can be stored on DASD (only sequential files can be stored on magnetic tape). You use DASD volumes for storing data and executable programs, including the operating system itself, and for temporary working storage. You can use one DASD volume for many different files, and reallocate or reuse space on the volume.

5.3 What are access methods?

An access method defines the technique that is used to store and retrieve data. Access methods have their own file structures to organize data, system-provided programs (or *macros*) to define files, and utility programs to process files.

Access methods are identified primarily by the file organization. z/VSE users, for example, use the basic access method (BAM).

This text does not describe all of the access methods available on z/VSE. Commonly used access methods include the following:

BAM	Basic Access Method (heavily used), which includes:
(B)DAM	(Basic) Direct Access Method (used for ICCF)
(B)SAM	(Basic) Sequential Access Method (for special cases)
VSAM	Virtual Sequential Access Method (used for more complex applications)

5.4 How are DASD volumes used?

DASD volumes are used for storing data and executable programs (including the operating system itself) and for temporary working storage. One DASD volume can be used for many different files, and space on it can be reallocated and reused.

On a volume, the name of a file must be unique. A file can be located by device type, volume serial number, and file name. This is unlike the file tree of a UNIX system. The basic z/VSE file structure is not hierarchical. z/VSE files have no equivalent to a path name.

Although DASD volumes differ in physical appearance, capacity, and speed, they are similar in data recording, data checking, data format, and programming. The recording surface of each volume is divided into many concentric *tracks*. The number of tracks and their capacity vary with the device. Each device has an access mechanism that contains read/write heads to transfer data as the recording surface rotates past them.

5.4.1 DASD terminology for UNIX and PC users

The disk and file characteristics of mainframe hardware and software differ considerably from UNIX and PC systems, and carry their own specialized terminology. Throughout this text, the following terms are used to describe various aspects of storage management on z/VSE:

- *Direct Access Storage Device* (DASD) is another name for a disk drive.
- A disk drive is also known as a disk volume, a disk pack, or a *Head Disk Assembly* (HDA). We use the term *volume* in this text except when discussing physical characteristics of devices.

- For *Count Key Data* (CKD)[2] devices:
  - A disk drive contains cylinders.
  - Cylinders contain tracks.
  - Tracks contain data records and are in CKD format.
- For FBA devices:
  - The disk drive is divided into blocks of fixed length.
  - Blocks contain the data.

5.4.2 What are DASD labels?

The operating system uses groups of labels to identify DASD volumes and the files they contain. Customer application programs generally do not use these labels directly. DASD volumes must use standard labels. Standard labels include a volume label, a file label for each file, and optional user labels. A volume label, stored at track 0 of cylinder 0, identifies each DASD volume.

The z/VSE system programmer or storage administrator uses the ICKDSF utility program to initialize each DASD volume before it is used on the system. ICKDSF generates the volume label and builds the volume table of contents (VTOC), a structure that contains the file labels. The system programmer can also use ICKDSF to scan a volume to ensure that it is usable and to reformat all of the tracks.

[2] Current devices actually use *Extended Count Key Data* (ECKD) protocols.

5.4.3 What is a VTOC?

z/VSE requires a particular format for disks, which is shown in Figure 5-1. Record 1 on the first track of the first cylinder provides the label for the disk. It contains the six-character volume serial number (volser) and a pointer to the volume table of contents (VTOC), which can be located anywhere on the disk.

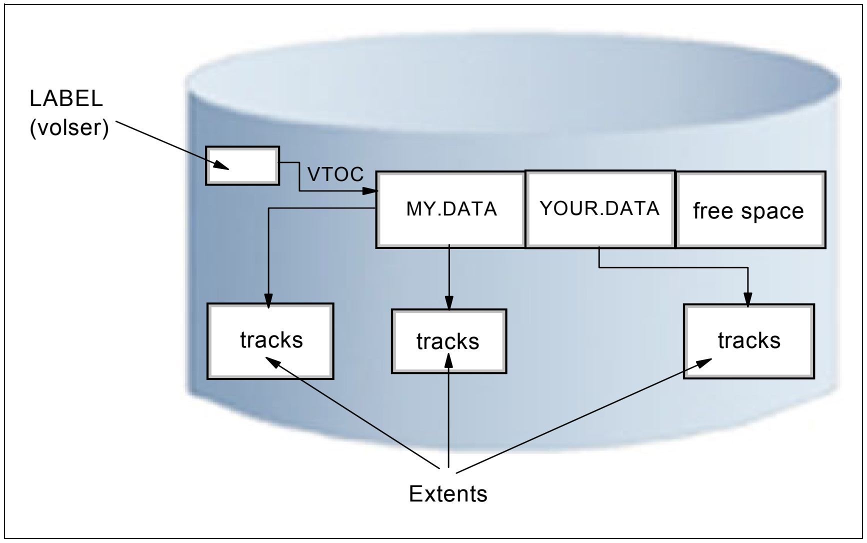


Figure 5-1 Disk label, VTOC, and extents

The VTOC lists the files that reside on its volume, along with information about the location and size of each file, and other file attributes. A standard z/VSE utility program, ICKDSF, is used to create the label and VTOC.

When a disk volume is initialized with ICKDSF, the owner can specify the location and size of the VTOC. The size can be quite variable, ranging from a few tracks to perhaps 100 tracks, depending on the expected use of the volume. More files on the disk volume require more space in the VTOC.

The VTOC also has entries for all the free space on the volume. Allocating space for a file causes system routines to examine the free space records, update them, and create a new VTOC entry. Files are always an integral number of tracks (or cylinders) and start at the beginning of a track (or cylinder).

A typical VTOC for z/VSE is shown in Figure 5-2.

```
                                             BEGIN     END     NUMBER SYSTEM
VTOC ENTRY                                  CYL TRK   CYL TRK  OF TRKs  USE
1...5...10...15...20...25...30...35...40....
VSE.SYSRES.LIBRARY                             0  1     63 14     959    *
DOS.LABEL.FILE.FF016F6A2084.AREA1             64  0     66 14      45    *
VSE.POWER.QUEUE.FILE                          67  0     67 14      15    *
Z9999996.VSAMDSPC.TBC1334D.T75E1E18           68  0     75 14     120    *
Z9999992.VSAMDSPC.TBC1334D.T9C01397           76  0    209 14    2010    *
***     FREE EXTENT     ***                  210  0    210 10      11    *
***     VTOC EXTENT     ***                  210 11    210 14       4    *
Z9999992.VSAMDSPC.TBC1334D.T7D7ABFC          211  0    392 14    2730    *
***     FREE EXTENT     ***                  393  0    408 14     240    *
VSE.SYSTEM.HISTORY.FILE                      409  0    413 14      75    *
DOS.PAGING.FILE.FF016F6A2084                 414  0    457 14     660    *
DOS.PAGING.FILE.FF016F6A2084                 458  0    865 14    6120    *
***     FREE EXTENT     ***                  866  0   1199 14    5010
```

Figure 5-2 VTOC of a DOSRES volume

5.5 What is VSAM?

VSAM
An access method for direct or sequential processing of fixed-length or variable-length records

The term *Virtual Storage Access Method* (VSAM) applies to both a file type and the access method used to manage various user files. As an access method, VSAM provides much more complex functions than other data access methods. VSAM keeps disk records in a unique format that is not understandable by other access methods.

VSAM is used primarily for applications. It is not used for source programs, JCL, or executable modules. VSAM files cannot be routinely displayed or edited.

You can use VSAM to organize records into four types of files: key-sequenced, entry-sequenced, relative record, or variable-length relative record. The primary difference among these types of files is the way their records are stored and accessed.

VSAM files are briefly described as follows:

- Key Sequence Data Set (KSDS)

  This is the most common use for VSAM. Each record has a key field and a record can be retrieved (or inserted) by key value. This provides random access of data. Records are variable length.

- Entry Sequence Data Set (ESDS)

  This form of VSAM keeps records in sequential order. Records can be accessed sequentially.

- Relative Record Data Set (RRDS)

  This VSAM format allows retrieval of records by number: record 1, record 2, and so forth. This provides random access and assumes that the application program has a way to derive the desired record numbers.

- Variable-length Relative-record Data Set (VRDS)

  This is the same as a Relative Record Data Set (RRDS), however, with the ability to handle variable length records.

5.5.1 What is a VSAM catalog?

VSAM uses a central file called a *catalog* to perform space and file management. The catalog itself is a VSAM file. It is self-describing and contains information about available space and files. The entry for a file in the catalog describes the file and indicates the volumes on which it is located. Through the catalog a file can be referred to by name without the user needing to specify where the file is stored.

For a given environment, you have a *master catalog,* and you can have one or more *user catalogs.*

Space management

In order to create VSAM files you need to have designated space on a volume as belonging to VSAM. In defining this VSAM space an entry is created in the VTOC. VSAM generates *names* for data spaces and enters the names in the VTOC of the applicable volume. You want to be able to recognize the names that relate to VSAM when you list the volume's VTOC.

The names generated by VSAM have the following format: For a data space that can be suballocated to VSAM files, the VSAM-generated name is:

```
Z999999n.VSAMDSPC.Taaaaaaa.Tbbbbbbb
```

Where:

n=2	If no catalog resides in the space
n=4	If a user catalog resides in the space
n=6	If the master catalog resides in the space
aaaaaaabbbbbbb	Is the time stamp value

See Figure 5-2 on page 145 for examples.

When you define a VSAM space on a volume, you set up a relationship between that space and a catalog. The space is owned by the catalog. You can define other spaces on that same volume or a different volume for the same catalog.

Master catalogs and user catalogs

A z/VSE system always has at least one master catalog. If a z/VSE system has a single catalog, this catalog would be the master catalog and the location entries for files would be stored in it. A single catalog, however, would be neither efficient nor flexible, so a typical z/VSE system uses a master catalog and numerous user catalogs connected to it. Every catalog exists on a single volume. It is independent of other catalogs and exclusively controls its own data spaces and files.

By placing information about your files and storage volumes into user catalogs, you decentralize control and reduce the time required to search a given catalog. Using several catalogs allows you to:

- Transfer files between the IBM operating systems z/VSE, z/VM, and z/OS. You can do so by using the EXPORT/IMPORT commands. ESDS, KSDS, and RRDS files are compatible between these operating systems. VRDS files are incompatible.
- Specify that one of the user catalogs is to be used as a job catalog. The job catalog will then be used to reference all the z/VSE files in the current job. You have the option of overriding the job catalog reference to a file through a z/VSE job control statement.

5.5.2 How VSAM files are named

When you define a new data set (or when the operating system does), you must give the data set a name. Specify a name that contains from 1 to 44 alphameric characters, national characters (@, #, and $), and two special characters (the hyphen and the 12-0 overpunch). Names containing more than eight characters must be segmented by periods. One through eight characters may be specified between periods. The first character of any name or name segment must be either an alphabetic or a national character. The last character in the name cannot be a period.

5.6 Libraries in z/VSE

There are two kinds of libraries in z/VSE:

- The VSE library
- The VSE/ICCF library

Below we will see the differences between these libraries.

5.6.1 What is a VSE library?

A VSE library can be considered as a file that is under the control of the librarian program LIBR.

A VSE library always consists of one or more sublibraries. A sublibrary contains members. A member keeps the real information. For example, programs, procedures, and dumps are stored as members in the sublibraries. Sublibraries vary in size. They are dynamically extended as required until the space assigned to the library as a whole is exhausted.

If library space is freed because a member has been deleted, this space is automatically available for reuse.

Sublibrary members are identified by member name and member type.

The librarian program addresses libraries by their names. Sublibraries are addressed by library and sublibrary name, for example:

```
LIB1.SUBN
```

If the librarian is to address members, an ACCESS or CONNECT command must be given first to specify in which sublibrary. For example:

```
ACCESS SUBLIBRARY=LIB1.SUBN
```

For other programs, the LIBDEF statement specifies in which sublibrary a member is to be searched for, or in which sublibrary it is to be stored. Sublibraries of one or more libraries may be chained (concatenated). Such a search chain is always related to a partition.

A VSE sublibrary also contains a directory. The directory contains an entry for each member in the sublibrary with a reference (or pointer) to the member. Member names are listed alphabetically in the directory, but member data is stored in library blocks in any order in the library. The directory allows the system to retrieve a particular member in the data set.

Types of VSE libraries

There are two types of VSE libraries: system libraries and private libraries.

- System library

  The system library (IJSYSRS) contains at least the predefined system sublibrary (SYSLIB). The system library is also referred to as the SYSRES file. After your system has been installed, IJSYSRS.SYSLIB contains all system programs needed for system startup and for the operation of your system. The system library occupies one extent only and resides on the disk volume used for performing IPL. This volume is referred to as DOSRES.

An alternate IJSYSRS.SYSLIB resides on the system volume SYSWK1. This is intended for use with system functions such as fast service upgrade and unattended node support.

- Private libraries

  User-written programs are usually stored in private libraries. It is preferable to have them on disk volumes of their own to reduce the disk arm movement on the SYSRES volume and to provide faster access to the programs in general.

  In addition, private libraries offer more flexibility with regard to size and organization. The following functions are supported for private libraries.

  A private library may be maintained in space managed by VSE/VSAM. This allows a dynamic extension of the library.

  Each private library may occupy several extents. These extents may be located on more than one disk volume.

  Besides being chained, libraries and their sublibraries may be shared among partitions and across systems. This reduces the effort for handling the libraries and increases system availability in general.

Types of VSE library members

Most of the members stored in your libraries belong to one of the predefined member types shown below:

source	A one-character, alphameric type: *source books* (source code to be processed by a language translator)
OBJ	*Object modules* (language translator output)
PHASE	*Phases* (machine-readable code processed by the linkage editor and ready for execution)
PROC	*Procedures* (sets of job control statements) or REXX programs
DUMP	*Dumps* (data collected by the dump routines of the system)

You may also define member types of your own. How to do this is discussed later in this section.

The *maximum member size* that the system can handle is about 2^{31} records for the fixed record format and about 2^{31} bytes (2 GB) for the undefined format. Usually, this is only of interest for members of type DUMP (when dumping a 2 GB partition).

- Source books

  Source books contain source code, either in the form of source statements or macro definitions, which are to be processed by a language translator. To specify the member type of a source book one of the following is allowed: A through Z; 0 through 9; #, $, or @. The letters A through I, P, R, and Z are reserved and used for system components.

  The sublibraries in which the language translator is to search for source books are specified in the job control statement:

  ```
  // LIBDEF SOURCE,SEARCH=library.sublibrary...
  ```

  This statement applies to all source member types.

  When the Assembler, for example, encounters a macro definition, the Assembler searches for the appropriate macro in the sublibraries specified (and in the system sublibrary) and replaces the statement with the source code found.

  Similarly, when a compiler encounters a reference to a source program, it searches for a SOURCE-type member of the same name in the sublibraries specified in the LIBDEF SOURCE,SEARCH statement.

- Object modules

  Language translators process source code and produce output in the form of object modules. These modules need to be processed by the linkage editor to become executable phases. During the link-editing of a module other modules may have to be included. If so, the linkage editor searches the sublibraries specified for the modules in question. In this way, sections of code that are used by a number of different programs need be written, translated, and cataloged in object format only once.

- Phases

  Phases are programs or sections of code that have been processed by the linkage editor and are ready to be loaded into storage for execution. Phases are cataloged by the linkage editor as members of the type PHASE in the sublibrary specified in the job control statement:

  ```
  // LIBDEF PHASE,CATALOG=library.sublibrary
  ```

  Normally, the linkage editor builds phases in relocatable format. For a relocatable phase, the loader of VSE/Advanced Functions modifies the addresses as required when the phase is being loaded. Such a phase can be loaded for execution into the address area of any partition.

Programs that have been written as self-relocatable are linked and cataloged by the linkage editor as self-relocatable phases. Any address relocation to be done for a self-relocatable phase must be handled within that phase itself after it has been loaded.

When a program is to be run, the phase name is specified in the job control EXEC statement. The loader searches for a member of the type PHASE with this name in the sublibraries specified in the job control statement and in the system sublibrary:

```
// LIBDEF PHASE,SEARCH=library.sublibrary...
```

- Procedures

  Procedures are frequently used sets of job control statements (and sometimes data) in card image format. These sets of control statements are referred to as cataloged procedures when they are stored in a sublibrary. You must store procedures as members of the type PROC, using the Librarian commands:

  ```
  ACCESS SUBLIB=library.sublibrary
  ```

  to specify the sublibrary to be used, and:

  ```
  CATALOG name.PROC...
  ```

  to name the procedure, followed by the job control statements (and data, if required) to be included in the procedure.

  A job stream being processed may call cataloged procedures for inclusion into that job stream.

  A cataloged procedure may contain inline data (that is, data that is read by the associated program from the device that is used for reading the job control statements (SYSIPT)). A procedure containing inline data must be cataloged with the operand DATA=YES in the CATALOG statement.

- Dumps

  Dumps contain system-relevant data and are created by the system's dump routines, for example, when an abnormal program termination occurs. The data is stored in the default dump library SYSDUMP:

  ```
  // LIBDEF DUMP,CATALOG=SYSDUMP.sublibrary
  ```

  The stored dumps can be analyzed later for problem determination with the Information/Analysis program. One dump sublibrary is available for each *static* partition with the partition identifier used as the sublibrary name: SYSDUMP.BG, SYSDUMP.F1, SYSDUMP.F2, and so on. Only one sublibrary is reserved for *dynamic* partitions: SYSDUMP.DYN.

  All these sublibraries should be used for dumps only.

- User-defined member types

  Besides the predefined member types discussed above, you may store any type of data in your libraries and assign your own member type to it.

  You can also change a predefined member type into a member type of your own definition, using the Librarian RENAME command.

  This allows you, for example, to maintain several versions of a cataloged procedure. You can achieve this by assigning the same member name but different member types (of your choice) to the different versions. One version, however, must always have the predefined member type for procedures, namely PROC. This version can be considered as activated, and when the procedure is called, it is always this version that is chosen. You can easily activate another version of this cataloged procedure by changing its member type to PROC. The same principle applies when you have different versions of programs, whether source programs or phases.

  A sublibrary may contain any or all member types used at an installation. This allows you to store, in one sublibrary, all members that are owned by one programmer, or that belong to one application.

Which programs make use of VSE library members?

Libraries are mainly accessed by the following programs:

- *Language translators* (Assembler or compiler), to retrieve source-type members for inclusion in the object module (OBJ) to be created.
- The *linkage editor*, to retrieve members of type OBJ for processing and to catalog members of type PHASE newly created.
- The *supervisor* fetch/load routine, to fetch or to load members of type PHASE into storage for execution.
- *Job control*, to retrieve members of type PROC called in EXEC PROC statements.
- *VSE/POWER*, to retrieve SLI members.
- *Dump programs*, to catalog members of type DUMP.
- The *info analysis* program, to work with members of type DUMP.
- The *librarian* program, to perform library service functions.
- User *application programs*, using the Librarian application program interface (API).

5.6.2 What is a VSE/ICCF library?

Different from VSE libraries, where each library is a separate file, VSE/ICCF libraries are all in the same file called DTSFILE. In addition, all ICCF users have to be defined in this DTSFILE, too.

The VSE/ICCF DTSFILE is allocated with approximately 40 MB and defines 199 VSE/ICCF libraries and users during initial installation. Other than VSE libraries, the VSE/ICCF libraries are identified by numbers (1 to 199). These libraries are also referred to as *program development* libraries. The VSE/ICCF DTSFILE consists of blocked records with a fixed record length of 88 bytes. 80 bytes are data and 8 bytes are used for forward and backward record chaining.

Concept

As a user of VSE/ICCF you have access to one or more libraries. You can own a library exclusively or you can share it with other users. Each library has a directory, and this directory contains an entry for each of the members in the library.

A single file of data in the library is called a *member*. Each member is represented by a member name and location in the directory. You can get a display of a library's directory by issuing the /LIBRARY command.

For reasons of efficiency and space management, it is of advantage to keep the number of members in your library as small as possible. Use the /PURGE command to remove entries from the library that are no longer needed.

Each member in the library consists of one or more 80-character records. The records can be programs, data, object decks, procedures, VSE/ICCF job streams, documentation, VSE JCL, or any combination of these.

You can transfer library members from the VSE/ICCF library file to a sublibrary of your VSE library system by invoking the LIBRC macro. By using the LIBRL and LIBRP macro you can copy a member from a VSE sublibrary to a VSE/ICCF library.

Members can be transferred to the VSE/POWER RDR queue, either using the SUBMIT function or via source library inclusion (* $$ SLI).

Members are created using the VSE/ICCF full screen editor, via VSE/ICCF macros or procedures or the batch utility DTSUTIL.

The VSE/ICCF DTSFILE is used to maintain the VSE/ICCF user profiles. Various authorization abilities can be specified. Although VSE/ICCF could be used on its own, normally VSE/ICCF is invoked via the Interactive Interface. This is especially true for the user profile and password maintenance. The Interactive

Interface holds this information in the VSE control file, which is part of the general VSE security concept.

Types of ICCF libraries

Basically, there are three types of a library in VSE/ICCF: PRIVATE, PUBLIC, and COMMON. The main difference between these three types is the degree of security that each type offers.

Table 5-1 shows the VSE/ICCF libraries and their use. Note that some libraries are reserved for z/VSE. The members that z/VSE ships in these libraries take up approximately 20% of the space reserved for the DTSFILE. After allocating a library for each user, determine the total requirement for the DTSFILE.

Table 5-1 VSE/ICCF libraries

Library	Type	Contents	Usage
1	Private	VSE/ICCF administrative library. Contents shipped with VSE/ICCF.	System
2	Common	Common library contains macros, procedures, VSE/ICCF, and z/VSE code members.	System
3 - 6	Public	Empty.	User
7	Private	Empty.	User
8	Private	Default primary library for operator profile.	User
9	Private	Default primary library for programmer profile.	User
10	Private	Default primary library for administrator profile.	User
11 - 49	Private	Empty.	User
50 - 58	Public	Reserved for z/VSE.	System
59	Public	z/VSE job streams, skeletons, CICS/VSE tables, and sample programs for the workstation file transfer support.	System
60 - 67	Public	Reserved for z/VSE.	System
68	Public	z/VSE members for personal computer tasks.	System
69	Public	Reserved for z/VSE.	System
70 - 199	Private	Empty.	User

5.7 Summary

A data set is a collection of logically related data. It can be a source program or a file of data records used by a processing program. Data set records are the basic unit of information used by a processing program.

Users must define the amount of space to be allocated for a file (before it is used), or these allocations must be automated through the use of VSAM. Almost all z/VSE data processing is record-oriented. Byte stream files are not present in traditional processing.

z/VSE VSAM files have names that contain from 1 through 44 alphameric characters, national characters (@, #, and $), and two special characters (the hyphen and the 12-0 overpunch). Names containing more than eight characters must be segmented by periods. One through eight characters may be specified between periods.

z/VSE libraries are z/VSE files whose content is managed and controlled by the z/VSE librarian. A library contains sublibraries, and a sublibrary contains different members. A member keeps the real data. There are different member types that define the kind of member data. In a z/VSE system there is one system library (IJSYSRS) with the sublibrary (SYSLIB) and multiple private libraries/sublibraries.

The ICCF library is used for program development. The ICCF library contains of members that can be modified with the ICCF editor. Each member in the library consists of one or more 80-character records. The records can be programs, data, object decks, procedures, VSE/ICCF job streams, documentation, VSE JCL, or any combination of these.

Key terms in this chapter		
VSE library	Catalog	Data set
VSE/ICCF library	System library	Private libraries
Member	Librarian program	DTSFILE
Phase	Virtual storage access method or VSAM	VTOC

5.8 Questions for review

1. What is a data set?

2. What types of VSAM files are used in z/VSE?
3. What is a VTOC and how is it different from a catalog?
4. What is the difference between the master catalog and user catalogs?
5. What is the main difference between UNIX files and a typical z/VSE file?
6. How could you transfer ICCF members to VSE/POWER for execution?
7. How could you transfer ICCF members to a VSE Library?
8. What is the name of the system library and its sublibrary? On which volume does it reside?
9. In a z/VSE system there are predefined member types and user members. Which are the predefined member types?
10. System programs make use of libraries, but is it also possible for user applications to work with z/VSE libraries?

5.9 Exercises

The lab exercises in this chapter help you develop skills in working with z/VSE files. These skills are required for performing lab exercises in the remainder of this text.

To perform the lab exercises, you or your team require a z/VSE user ID and password (for assistance, see the instructor).

The exercises teach the following:

- Defining a VSAM file with Interactive Interface
- Listing catalog contents with Interactive Interface
- Working with a batch job

5.9.1 Defining a VSAM file with Interactive Interface

The Interactive Interface provides a convenient method for defining VSAM files. In this exercise, you create a new file. The new file should be placed on the SYSWK1 volume and should be named *yourid*.TEST.FILE (where *yourid* is your student user ID).

For this exercise, assume that two tracks of primary space and two tracks for secondary extents is sufficient. Furthermore, we know we want to store 80-byte fixed-length records in the library. We can do this as follows:

1. Start at the Interactive Interface primary menu.

2. Use fastpath 22 or go to the option 2 (Resource Definition) and then go to option 2 (File and Catalog Management).
3. Enter `VSESPUC` (the label for the user catalog VSESP.USER.CATALOG that is delivered with the z/VSE system) in the field for the CATALOG NAME.
4. Go to option 2 (Define a New File).
5. Type the name of the new file in the FILE ID field.
6. Use your student user ID as the FILE NAME.
7. Complete the indicated fields and press Enter:
   - FILE ORGANIZATION = 1 (Non keyed ESDS)
   - FILE ADDRESSABILTY = 1 (Not extended)
   - FILE ACCESS = 1 (Multiple Read OR Single Write)
   - FILE USAGE = 1 (File is used as a Data File NOREUSE)
8. Select **SYSWK1** for the primary allocation.
9. Complete the indicated fields and press Enter:
   - ALLOCATION UNIT = 2 (Tracks)
   - PRIMARY ALLOCATION = 2
   - SECONDARY ALLOCATION = 2
   - AVERAGE RECORD SIZE = 80
   - MAXIMUM RECORD SIZE = 80
10. Select immediate execution and press Enter.

5.9.2 Listing catalog contents with Interactive Interface

1. Use fastpath 22 or go to the option 2 (Resource Definition) and then go to option 2 (File and Catalog Management).
2. Select option 5 (Display or Process a Catalog, Space) and press Enter. This should list all of the catalogs in the system.
3. A number of functions can be invoked by entering the appropriate number before a catalog name. For example, position the cursor before one of the catalog names and press PF1 (Help).

   The Help panel lists all of the commands that can be used from the catalog list of the IESFILCL1 panel:

```
1  SHOW SPACE
2  DEFINE ALTERNATE NAME
3  PRINT CATALOG CONTENTS
4  DEFINE SPACE
5  DELETE CATALOG
6  DELETE SPACE
```

4. Select option 3 (PRINT CATALOG CONTENTS) for catalog VSESP.USER.CATALOG.
5. Select immediate execution and press Enter.
6. Locate the JOB output in the LIST queue and display it. Locate the entry for the previously defined file in the output.

5.9.3 Working with a batch job

To build a new job, submit it, and check the result:

1. You need a programmer type user ID.
2. Log on and select 1 Program Development Library.
3. Press PF6 to create a new member.

   Enter this job:

   ```
   * $$ JOB JNM=TEST,DISP=D,CLASS=A
   // JOB TEST
   * MY FIRST VSE JOB
   // EXEC DTSDUMMY
   /*
   /&
   * $$ EOJ
   ```

4. Press PF3 to save this member.
5. Press PF3 again to return to fulist.
6. Press PF2 to refresh the fulist.
7. Enter Option 7 for the member you just created. This submits your job to VSE/POWER.
8. Check in the POWER LIST queue for your job output TEST and BROWSE it.

6

Using JCL

Objective: As a technical professional in the world of mainframe computing, you will need to know JCL, the language that tells z/VSE which resources will be needed to process a batch job.

After completing this chapter, you will be able to:

- Explain how JCL works with the system, an overview of JCL coding techniques, and a few of the more important statements and keywords.
- Create a simple job and submit it for execution.
- Check the output of your job through the Interactive Interface.

6.1 What is JCL?

For every batch job that you submit, you need to tell z/VSE where to find the appropriate input, how to process that input (that is, what program or programs to run), and what to do with the resulting output. You use *job control language* (JCL) to convey this information to z/VSE through a set of statements known as job control statements.

JCL
Language used to tell z/VSE what program to execute and defines its inputs and outputs

The set of job control statements is quite large, which allows you to provide a great deal of information to z/VSE. Most jobs, however, can be run using a very small subset of these control statements. Once you become familiar with the characteristics of the jobs you typically run, you may find that you need to know the details of only some of the control statements. This chapter discusses just a selected few.

Who uses JCL? While application programmers need some knowledge of JCL, the production control analyst responsible must be *highly* proficient with JCL, to create, monitor, correct, and rerun the company's daily batch workload. System programmers also tend to be expert users of JCL.

Related reading: For descriptions of the full set of JCL statements, see the IBM publications *z/VSE System Control Statements* and *z/VSE Guide to System Function*s. They are available at the z/VSE Internet Library Web site:

`http://www.ibm.com/servers/eserver/zseries/zvse/`

6.2 Basic JCL statements

Within each job, the control statements are grouped into job steps. A job step consists of all of the control statements needed to run one program. If a job needs to run more than one program, the job would contain a different job step for each of those programs.

There are six basic JCL statements:

JOB	Provides a name for the job (jobname to the system) for control purposes, as well as providing user accounting information.
ASSGN	Provides a physical device for a logical unit, used for the inputs and outputs of the execution program on the EXEC statement. This statement links a physical I/O device to a logical unit coded in the program. ASSGN statements are associated with a particular job.
DLBL	Provides the inputs and outputs to the execution program on the EXEC statement. This statement links a data set to a file name

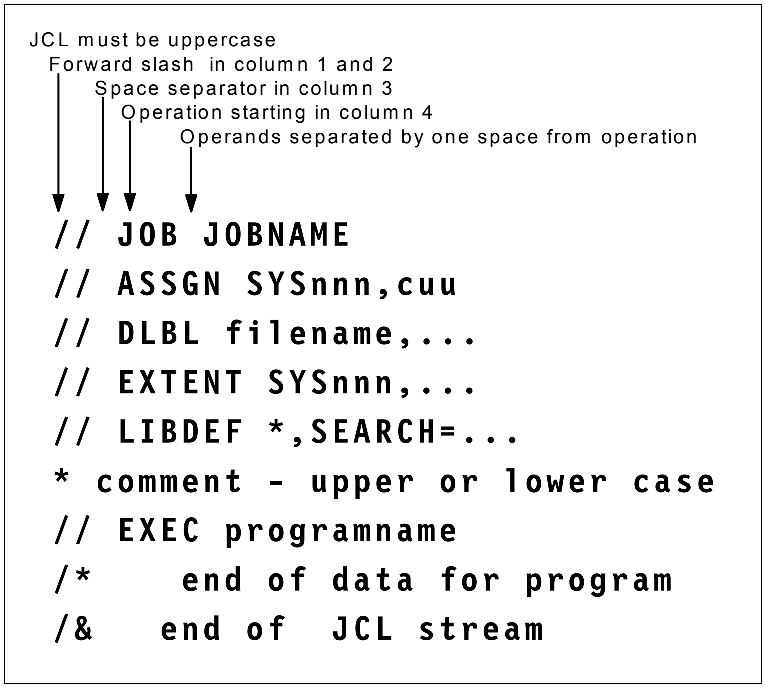
coded in the program. DLBL statements are usually associated with a particular job.

EXTENT Provides the extents of inputs and outputs to the execution program on the EXEC statement. This statement links the extents of a data set to a file name coded in the program. EXTENT statements are associated with a particular DLBL statement.

LIBDEF The LIBDEF statement defines which sublibraries are to be searched for members of a specified type or types, and, where appropriate, the sublibrary in which new phases or dumps are to be stored. The defined sequence is referred to as a search chain. Different chains can be defined for different member types, or a common chain can be established for all types except DUMP.

EXEC Provides the name of a program to execute. There can be multiple EXEC statements in a job. Each EXEC statement within the same job is a *job step*.

Figure 6-1 shows the coding syntax of the basic JCL statements.


JCL must be uppercase
Forward slash in column 1 and 2
Space separator in column 3
Operation starting in column 4
Operands separated by one space from operation

```
// JOB JOBNAME
// ASSGN SYSnnn,cuu
// DLBL filename,...
// EXTENT SYSnnn,...
// LIBDEF *,SEARCH=...
* comment - upper or lower case
// EXEC programname
/*    end of data for program
/&   end of  JCL stream
```

Figure 6-1 Basic JCL coding syntax

A complete job may look like Example 6-1.

Example 6-1 JCL example

```
// JOB MYJOB
// ASSGN SYS001,160
// DLBL INFILE,'MYPROG.INPUT',2009/001
// EXTENT SYS001,,1,0,100,400
// ASSGN SYS002,160
// DLBL OUTFILE,'MYPROG.OUTPUT',2010/365
// EXTENT SYS002,,1,0,500,500
// LIBDEF PHASE,SEARCH=(MYLIB.MYSUBA),TEMP
// EXEC MYPROG
/&
```

Job name
The name by which a job is known to the system (JCL statement)

When this job was submitted for execution:

MYJOB	Is a job name the system associates with this workload.
SYS001	Is the logical unit. The SYS001 logical unit is coded in the MYPROG program as an input unit.
INFILE	On the DLBL statement is the file name. The INFILE file name is coded in the MYPROG program as a program input.
SYS002	Is the logical unit. The SYS002 logical unit is coded in the MYPROG program as an output unit.
OUTFILE	On the DLBL statement is the file name. The OUTFILE file name is coded in the MYPROG program as a program output.
MYLIB.MYSUBA	Is the sublibrary that contains the program MYPROG.
MYPROG	Is the name of the program that the system has to execute.

6.3 Parameters for the basic JCL statements

The JOB, ASSGN, DLBL, EXTENT, LIBDEF, and EXEC statements have many parameters to allow you to specify instructions and information. Describing them all would fill an entire book, such as the IBM publication, *z/VSE System Control Statements*.

This section provides only a brief description of a few of the more commonly used operands for the JOB, ASSGN, DLBL, EXTENT, LIBDEF, and EXEC statements.

6.3.1 JOB statement parameters

The JOB statement is used to identify the job (the *job name*) and, optionally, specify the job's accounting information, which is sometimes used for charging system users. The information in the JOB statement applies to all job steps within the job. For example, the following statement is a JOB statement with name MYJOB and an accounting field set to 1:

```
// JOB MYJOB 1
```

6.3.2 ASSGN statement parameters

The ASSGN statement is used to assign a logical I/O unit to a physical device. For example:

```
// ASSGN SYS001,200
```

ASSGN statement Assigns a logical I/O unit to a physical device.

The ASSGN statement is followed by a logical unit name and a physical unit address. It remains in effect only until the next change in assignment or until the end of the job.

The logical unit name can be either of the following:

- A programmer logical unit SYS*nnn*, where *nnn* can be a decimal number from 000 to 254
- A system logical unit, for example:
  - SYSRDR, which is used to read input
  - SYSLST, which is used to write printed output
  - SYSPUN, which is used to write punched output
  - SYSLNK, which is used as input to the linkage editor
  - SYSLOG, which is used for communication with the operator.

6.3.3 DLBL statement parameters

The DLBL statement is used to specify the data inputs and data outputs to the program that is specified on the EXEC statement. The DLBL statement links a data set to a *file name* that is coded in the program. DLBL statements are associated usually with a particular job, but may be associated also with all jobs or just a particular job step. For example, the following DLBL statement has a file name of INFILE:

DLBL statement Specifies inputs and outputs for the program in the EXEC statement

```
// DLBL INFILE,'MYPROG.INPUT',2009/001
```

The DLBL supports many operands. The first four are positional operands and must be coded in a predefined sequence. All operands are optional, except the first one, which specifies the file name.

First operand	Specifies the file name that is coded in the program. This operand is mandatory.
Second operand	Specifies the unique data set name or file identifier (file-id) on the physical device with the file name. The file-id can be from one to 44 characters, contained within quotes. If fewer than 44 characters are used, the field is left-justified and padded with blanks. If the operand is omitted, the file-id is equal to the file name.
Third operand	Is used to determine the expiration date of the data set. You may specify, for example:
yy/ddd	Specifies the date: The data set expires on day number ddd in year yy.
Nnn	Specifies the retention period in days. The data set expires after nnn days.
no operand	If this operand is omitted, a retention period of 7 days is assumed.
Fourth operand	Indicates the type of the data set. You may specify, for example:
SD	For a sequential data set.
DA	For a direct accessed data set.
VSAM	For a Virtual Storage Access Method accessed data set.
no operand	If this operand is omitted, SD is assumed.

The most used operand following the fourth operand is:

DSF	Specifies that a data-secured data set is to be processed. When the data set is opened, a warning message is issued to the operator, who then decides whether the data set may be accessed.

When VSAM has been specified, the mostly used operands are the following ones, which are all keyword operands:

RECSIZE=	Specifies the average record length of the data set.
RECORDS=	Specifies the number of records to be considered when allocating space for the data set.

DISP= Specifies the status of the data set. This field consists of up to three values describing the status at different times:

```
DISP=(status,normal end,abnormal end)
DISP=(status,normal end)
DISP=status
```

Where status can be NEW, OLD, DELETE, KEEP, or DATE. For a detailed description of the meaning of these keywords and the default values see *VSE/VSAM User's Guide and Application programming*.

The *normal end* parameter indicates what to do with the data set (the *disposition*) if the current job step ends normally. Likewise, the abnormal end parameter indicates what to do with the data set if the current job step abnormally ends.

Why z/VSE uses symbolic file names

z/VSE normally uses symbolic file names, and this is another defining characteristic of this operating system. It applies a naming redirection between a data set-related name used in a program and the actual data set used during execution of that program. This is illustrated in Figure 6-2.

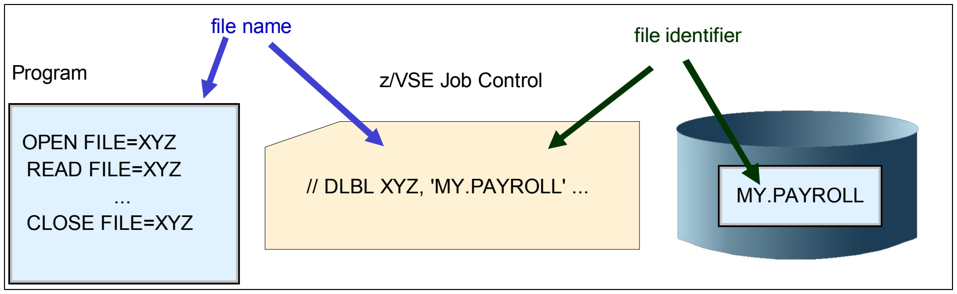


Figure 6-2 File name and file identifier

Symbolic file name
A naming redirection between a data set-related name used in a program and the actual data set used during execution of that program

In this illustration we have a program, in some arbitrary language, that needs to open and read a data set[1]. When the program is written, the name XYZ is arbitrarily selected to reference the data set. The program can be compiled and stored as an executable. When someone wants to run the executable program, a JCL statement must be supplied that relates the name XYZ to an actual data set name (file-id). This JCL statement is a DLBL statement. The symbolic name used in the program is a file name and the real name of the data set is a file identifier.

[1] The pseudo-program uses the term *file*, as is common in most computer languages.

The program can be used to process different input data sets simply by changing the file identifier in the JCL. This becomes significant for large commercial applications that might use dozens of data sets in a single execution of the program. A payroll program for a large corporation is a good example. This can be an exceptionally complex application that might use hundreds of data sets. The same program might be used for different divisions in the corporation by running it with different JCL. Likewise, it can be tested against special test data sets by using a different set of JCL.

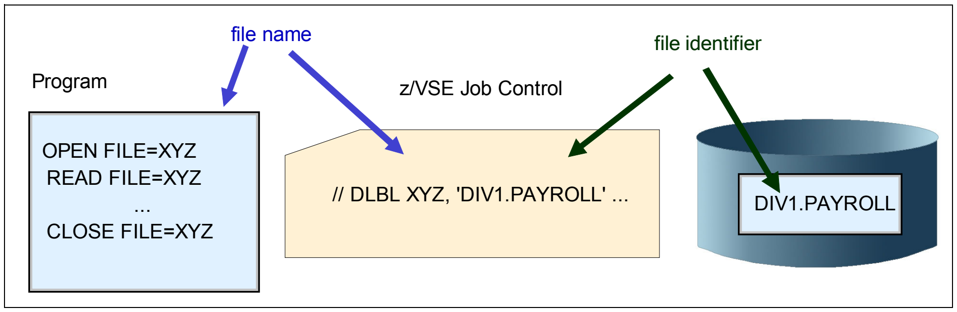


Figure 6-3 Symbolic file name - same program, but another data set

The company could use the same company-wide payroll application program for different divisions and only change a single parameter in the JCL card (the DLBL XYZ,'DIV1.PAYROLL'). The parameter value DIV1.PAYROLL would cause the program to access the data set for division 1. This example demonstrates the power and flexibility afforded by JCL and symbolic file names.

This file-name--JCL--file-id processing applies to all traditional z/VSE work, although it might not always be apparent. For example, when ICCF is used to edit an ICCF library member, the data set that contains the ICCF libraries is already assigned to the file name DTSFILE.

Reserved file names

A programmer can select *almost* any name for a file name. However, using a meaningful name (within the 7-character limit) is recommended. There are some reserved file names for system usage that a programmer cannot use:

```
// DLBL IJSYSRS,...                     used for system library
// DLBL IJxxxxx,...                     used for various system tasks
// DLBL DFHxxxx,...                     used by CICS
// DLBL DTSFILE,..                      used by ICCF
// DLBL IESnnnn,..                      used by Interactive User Interface
// DLBL BLNDMF,.. or BLNXTRN or SYSDUMP used by dump management
```

6.3.4 EXTENT statement parameters

The EXTENT statement defines each area, or extent, of a disk or diskette. One or more EXTENT statements must directly follow each DLBL statement (except for VSAM files).

The following are samples of an EXTENT statement:

```
// EXTENT SYS001,,1,0,100,400
// EXTENT ,VOLUM1,1,0,100,400
```

Common operands found on the EXTENT statements are:

First operand	Specifies the logical unit of the volume for which this extent is effective. If this operand is omitted, the logical unit of the preceding EXTENT statement, if any, is used. If this operand is omitted on the first or only EXTENT statement, the logical unit specified in the program is assumed.
Second operand	Specifies the volume serial number of the volume for which this extent is effective. If this operand is omitted, the volume serial number of the preceding EXTENT statement, if any, is used. If this operand is omitted, the volume serial number is not checked and files may be destroyed, because the wrong volume was mounted.
Third operand	Specifies the type of the extent, as follows: 1 = data area (mostly used) 2 = overflow area (for indexed sequential data sets) 4 = index area (for index sequential data sets) 8 = split cylinder for data area of SAM data sets
Fourth operand	Specifies the sequence number (a decimal number between 0 and 255) of this extent within a multi-extent file.
Fifth operand	Specifies the number of the track (a decimal number of 0 and 999,999), relative to zero, where the extent of the data set begins.
Sixth operand	Specifies the number of tracks (a decimal number between 1 and 999,999) to be allocated for the extent.

6.3.5 LIBDEF statement parameters

The LIBDEF statement is used to define which sublibraries are to be searched for members of a specified type or types. The specified sublibraries are searched in the sequence entered in the LIBDEF statement.

The system sublibrary IJSYSRS.SYSLIB is normally added at the end[2] in the search chain, unless it is explicitly included at a different position in the chain.

Some common LIBDEF statement parameters include:

First operand	Specifies the type of the search chain as follows:
PHASE	Defines a sublibrary chain to be used for loading program phases for execution.
OBJ	Defines a sublibrary chain to be used by the linkage editor when searching for object modules. Only members of the type OBJ are searched for.
SOURCE	Defines a sublibrary chain to be used, for example, by language translators, when searching for one of the predefined SOURCE types (A–Z, 0–9, #, $, @). The CATALOG operand is not applicable.
PROC	Defines a sublibrary chain to be used by job control when searching for a procedure to be executed. Only members of the type PROC are searched for.
*****	Indicates that the LIBDEF statement applies to all member types except DUMP and user types.
Second operand	Specifies the function and its sublibraries:
SEARCH=	Specifies the names and the sequence of the sublibrary to be searched.
CATALOG=	Specifies the sublibrary into which the linkage editor output is to be cataloged.
Third operand	Specifies the duration of the definition given in the statement.
TEMP	For TEMP, the defined chain will be dropped: At end-of-job When overridden by a new LIBDEF...TEMP statement When a LIBDROP...TEMP statement is issued
PERM	For PERM, the defined chain will remain valid until: The partition is deactivated A new LIBDEF statement overrides the definition wholly or in part When a LIBDROP...PERM statement is issued

[2] Special rules apply for system phases. See *z/VSE System Control Statements*.

In 6.4, “Specifying continuation and concatenation” on page 170, there is an example for the LIBDEF statement with more than one sublibrary defined.

6.3.6 EXEC statement parameters

In JCL, the EXEC statement is used to specify the name of a program to run. A job often runs multiple programs, and therefore contains multiple EXEC statements. Each EXEC statement within the same job is a job step. For example:

```
// EXEC MYPROG
```

Or:

EXEC statement
JCL that gives the name of a program to be executed

```
// EXEC PGM=MYPROG
```

The EXEC statement is followed by an executable program name (MYPROG) or by the PGM=executable program name. z/VSE locates the program by searching program libraries in the following order:

1. Temporary JOBLIB, if present
2. Permanent JOBLIB, if present
3. System library (IJSYSRS.SYSLIB).

Common operands found on the EXEC statement are:

PARM=	Parameters known by and passed to the program.
SIZE=	Specifies the size of that part of the partition that will be directly available to the program. The remainder of the partition may be used as additional storage (GETVIS area) for other modules or data required by the program in that partition.
DSPACE=	Specifies the maximum size (nK or nM) of a data space (in units of KB or MB).

Example:

```
// EXEC MYPROG,PARM='TRACE',SIZE=400K,DSPACE=3M
```

When the program to be executed has just been processed by the linkage editor, the program name may be omitted.

Example:

```
// EXEC
```

JCL statements may be catalogued together in a JCL procedure (see 6.5, “Using JCL procedures” on page 170) and may be executed by specifying PROC= in the // EXEC statement.

Example:

```
// EXEC PROC=MYPROC
```

6.4 Specifying continuation and concatenation

As a consequence of the limitations of the number of characters that could be contained in single 80-column punched cards used in earlier systems, early mainframe operating systems introduced the concepts of continuation and concatenation. z/VSE retains these conventions in order to minimize the impact on previous applications and operations.

If a JCL statement consists of more than 71 characters, it must be split into more than one *JCL card*. The first 71 characters of the JCL statement must be on the first JCL card. Any non-blank character in column 72[3] indicates JCL continuation. The JCL statement must be continued in column 16 of the next JCL card, whereby the first 15 columns must contain blanks. In the following sample the EXEC statement consists of 76 characters (including the single quote at the end):

```
// EXEC MYPROG,PARM='MONDAY,JANUARY,1,10,100,1000,1000,10000,10000000072345'
```

The JCL statement must then be coded in two cards, as follows (whereby the x in column 72 indicates JCL continuation and does not belong to the parameter passed to the program MYPROG):

```
// EXEC MYPROG,PARM='MONDAY,JANUARY,1,10,100,1000,1000,10000,1000000007x
               2345'
```

Another sample for continuation is:

```
// LIBDEF PHASE,SEARCH=(MYLIB.MYSUBA,MYLIB.MYSUBE,MYLIB.MYSUBX),    C
                 CATALOG=YOURLIB.YSUBA,                             C
                 TEMP
```

In this sample we also see the concatenation of the three sublibraries MYLIB.MYSUBA, MYLIB.MYSUBE, and MYLIB.MYSUBX to be searched for program phases.

6.5 Using JCL procedures

Some programs and tasks require a larger amount of JCL than a user can easily enter. JCL for these functions can be kept in procedure libraries. A procedure library member contains *part* of the JCL for a given task (usually the fixed,

[3] Columns 73 through 80 are reserved for something called card sequence numbers.

unchanging part of JCL). The user of the procedure supplies the variable part of the JCL for a specific job. In other words, a JCL procedure is like a macro.

Such a procedure is known as a *cataloged procedure*. Do not confuse a cataloged procedure with a cataloged data set. A cataloged procedure is a named collection of JCL stored in a data set (or library), and a cataloged data set is a data set whose name is recorded by the system (as we discussed in Chapter 5, "Working with files" on page 139).

Example 6-2 Cataloging a JCL procedure

```
// JOB CATALOG
// EXEC LIBR
 ACCESS SUBLIB=IJSYSRS.SYSLIB
 CATALOG MYPROC.PROC
// ASSGN SYS001,160
// DLBL INFILE,'MYPROG.INPUT',2009/001
// EXTENT SYS001,,1,0,100,400
// ASSGN SYS002,160
// DLBL OUTFILE,'MYPROG.OUTPUT',2010/365
// EXTENT SYS002,,1,0,500,500
/+
/*
/&
```

Much of this JCL should be recognizable now. New JCL functions presented here include:

- The EXEC statement calls a program LIBR[4], which is a z/VSE utility used to maintain system libraries.
- The utility statement ACCESS specifies the library in which the procedure should be cataloged. IJSYSRS.SYSLIB is the name of the z/VSE system sublibrary.
- The utility statement CATALOG specifies the name of the procedure whereby the type (specified after the '.') must be PROC for a procedure.
- The /+ statement indicates the end of the procedure to be cataloged.

[4] More about the VSE librarian can be found in 5.6.1, "What is a VSE library?" on page 148.

In Example 6-3 we include the cataloged procedure in our job stream.

Example 6-3 Invoking a procedure

```
// JOB MYJOB
// EXEC PROC=MYPROC
// EXEC MYPROG

/&
```

When MYJOB is executed, the JCL from procedure MYPROC is inserted.

6.5.1 Using symbolic parameters

The flexibility of jobs can further be increased by using symbolic parameters. Symbolic parameters can be specified in any job or cataloged procedure. By doing so you are able to set up job streams in advance and modify them at processing time as needed. A symbolic parameter is defined by a string of 1 to 7 alphameric characters preceded by an ampersand (&). A symbolic parameter is referenced by the same character string without the ampersand.

Example 6-4 Sample procedure MYPSYM with symbolic parameters

```
// ASSGN SYS001,160
// DLBL INFILE,'MYPROG.INPUT',2009/001
// EXTENT SYS001,,1,0,100,400
// ASSGN SYS002,&SYS02
// DLBL OUTFILE,'&FIDOUT',2010/365
// EXTENT SYS002,,1,0,500,500
```

Example 6-5 Invoking sample procedure MYPSYM with symbolic parameters

```
// JOB MYJOB
// EXEC PROC=MYPSYM,SYS02=250,FIDOUT=TESTFILE
// EXEX MYPROG
/&
```

Example 6-6 Generated jobstream by using procedure MYPSYM with symbolic parameters

```
// JOB MYJOB
// ASSGN SYS001,160
// DLBL INFILE,'MYPROG.INPUT',2009/001
// EXTENT SYS001,,1,0,100,400
// ASSGN SYS002,250
// DLBL OUTFILE,'TESTFILE',2010/365
// EXTENT SYS002,,1,0,500,500
// EXEC MYPROG
/&
```

6.6 What are utilities?

Utility
Program that provides a useful batch function

z/VSE includes a number of programs useful in batch processing called *utilities*. These programs provide many small, obvious, and useful functions. A basic set of system-provided utilities is described in Appendix B, “Utility programs” on page 429.

Customer sites often add their own customer-written utility programs (although most users refrain from naming them *utilities*), and many of these are widely shared by the user community. Independent software vendors also provide many similar products (for a fee).

6.7 What are system libraries?

z/VSE has some standard system libraries. A brief description of several libraries is appropriate here. The traditional libraries include:

- IJSYSRS. This library contains:
  - System execution modules that keep the operating system running. Those are, for example, modules of the supervisor, JCL, POWER, and VSAM.
  - System procedures that are processed when the system is initialized
- PRD1. This library contains base products, for example, VTAM, CICS, TCP/IP, and REXX.
- PRD2. This library contains optional products, like compilers.
- SYSDUMP. This library contains the dump produced when a program terminates abnormally.

For more information about system libraries see 15.2, “Customizing the system” on page 373.

6.8 Summary

For every job that you submit, you need to tell z/VSE where to find the appropriate input, how to process that input (that is, what program or programs to run), and what to do with the resulting output. You use JCL to convey this information to z/VSE through a set of statements known as job control statements. A set of job control statements is quite large, enabling you to provide a great deal of information to z/VSE. Most jobs, however, can be run using a very small subset of these control statements. Once you become familiar with the characteristics of the jobs you typically run, you may find that you need to know the details of only some of the control statements.

Within each job, the control statements are grouped into job steps. A job step consists of all the control statements needed to run one program. If a job needs to run more than one program, the job would contain a different job step for each of those programs.

Basic JCL contains six types of statements: JOB, ASSGN, DLBL, EXTENT, LIBDEF, and EXEC. A job can contain several EXEC statements (steps), and each step might have several LIBDEF, ASSGN, DLBL, and EXTENT statements. JCL provides a wide range of parameters and controls. Only a basic subset is described here.

A batch job uses artificial names (file names) internally to access data sets. A JCL DLBL statement connects the file name to a specific data set (DS name) for one execution of the program. A program can access different groups of data sets (in different jobs) by changing the JCL for each job.

System users are expected to write simple JCL, but normally use JCL procedures for more complex jobs. A cataloged procedure is written once and can then be used by many users. z/VSE supplies many JCL procedures, and locally written ones can be added easily. A user must understand how to override or extend statements in a JCL procedure in order to supply the parameters (usually ASSGN and DLBL statements) needed for a specific job.

Key terms in this chapter		
JOB statement	DLBL statement	EXTENT statement
EXEC statement	LIBDEF statement	ASSGN statement
Jobname	Cataloged procedure	Utility
Continuation	Concatenation	Library

6.9 Questions for review

To help test your understanding of the material in this chapter, complete the following questions:

1. Explain the relationship between a data set name, a DLBL name, and the file name within a program.
2. Which JCL statement (JOB, EXEC, or DLBL) has the most operands? Why?
3. What is the difference between JCL and a JCL PROC? What is the benefit of using a JCL PROC?
4. How can an operand in a JCL statement be changed at the time the job is processed? What has to be observed when specifying the operand to be changed?

6.10 Topics for further discussion

This material is intended to be discussed in class, and these discussions should be regarded as part of the basic course text.

1. Why has the advent of database systems potentially changed the need for large numbers of DLBL statements?
2. The second positional parameter of a JOB statement is an accounting field. How important is accounting for mainframe usage? Why?

6.11 Exercises

The lab exercises in this chapter help you develop skills in creating batch jobs and submitting them for execution on z/VSE. These skills are required for performing lab exercises in the remainder of this text.

To perform the lab exercises, you or your team require an IUI user ID and password (for assistance, see the instructor).

The exercises teach the following:

- Creating simple jobs and procedures
- Submitting job and checking the results

6.11.1 Creating simple jobs and procedures using dialogs

1. Check for free space:
   a. Use the LVTOC utility to search for free space on a disk device.
   b. To have a general solution, create a procedure with the variable parameter &VOLID used in the ASSIGN statement for the LVTOC utility (SYS004 must be used for the disk device, SYS005 for the list output).
   c. Catalog the procedure in the system library.
   d. Invoke the procedure with the VOLID SYWK1.
   e. When asked for, list secured files, too.
   f. Check the output for free space on the disk. Use Interactive Interface dialogs to search the list output for free space.
2. Define a private library and sublibrary using the utility LIBR.
3. Define an REXX program that is cataloged into the previously defined library as a procedure, getting as input a character string and issuing a message containing this character string.

   To include the name of the called procedure in the message, you may use the following REXX statement, which returns in rexxproc the name of the called procedure (code the points '.' as indicated):

   ```
   parse source . . rexxproc .
   ```
4. Call REXX to process the cataloged procedure and pass some character string as input.

7

Batch processing and VSE/POWER

Objective: As a mainframe professional, you will need to understand the ways in which the system processes your company's core applications, such as payroll. Such workloads are usually performed through *batch processing,* which involves executing one or more *batch jobs* in a sequential flow.

Further, you will need to understand how the *VSE job entry and spooling subsystem* (VSE/POWER) enables batch processing. VSE/POWER helps z/VSE receive jobs, schedule them for processing, and determine how job output is processed.

After completing this chapter, you will be able to:

- Give an overview of batch processing and how work is initiated and managed in the system.
- Explain how VSE/POWER governs the flow of work through a z/VSE system.

7.1 What is batch processing?

The term *batch job* originated in the days when punched cards contained the directions for a computer to follow when running one or more programs. Multiple card decks representing multiple jobs would often be stacked on top of one another in the hopper of a card reader, and be run in batches.

Batch jobs
Programs that can be executed with minimal human interaction, typically executed at a scheduled time

Today, jobs that can run without end-user interaction, or can be scheduled to run as resources permit, are called batch jobs. A program that reads a large file and generates a report, for example, is considered to be a batch job.

There is no direct counterpart to z/VSE batch processing in PC or UNIX systems. Batch processing is for those frequently used programs that can be executed with minimal human interaction. They are typically executed at a scheduled time or on an as-needed basis. Perhaps the closest comparison is with processes run by an AT or CRON command in UNIX, although the differences are significant.

You might also consider batch processing as being somewhat analogous to the printer queue, as it is typically managed on an Intel-based operating system. Users submit jobs to be printed, and the print jobs wait to be processed until each is selected by priority from a print spool.

To enable the processing of a batch job, z/VSE professionals use job control language (JCL) to tell z/VSE which programs are to be executed and which files will be needed by the executing programs. JCL allows the user to describe certain attributes of a batch job to z/VSE, such as:

- Who you are (the submitter of the batch job)
- What program to run
- Where input and output is located.

After the user submits the job to the system, there is normally no further human interaction with the job until it is complete.

The use of JCL is covered in detail in Chapter 6, “Using JCL” on page 159, but for now understand that JCL is the means by which a batch job requests resources and services from the operating system.

7.2 What is VSE/POWER?

z/VSE uses a *job entry and spooling subsystem* named VSE/POWER to receive jobs into the operating system, to schedule them for processing by z/VSE, and to control their output processing. VSE/POWER is the component of the operating system that provides supplementary job management, and task management functions such as scheduling, control of job flow, and spooling.

z/VSE manages work as tasks and subtasks. Both transactions and batch jobs are associated with an internal task queue that is managed on a priority basis. VSE/POWER is a component of z/VSE that works on the front end of program execution to prepare work to be executed. VSE/POWER is also active on the back end of program execution to help clean up after work is performed. This includes managing the printing of output generated by active programs.

More specifically, VSE/POWER manages the input and output job queues and data. Therefore, it uses its own Job Entry Control Language (JECL) together with operator commands to control jobs and outputs. One or more jobs are wrapped in VSE/POWER JECL and build a VSE/POWER job. All VSE/POWER JECL starts with * $$ and contains processing information for jobs and outputs (class, disposition, priority, and so on). The processing information for output is specified in the VSE/POWER job, which generates the output, so output itself does not contain any JECL. To keep the processing information, a control record (called queue record) is built, which is linked with the data of the job or output.

To organize the queue records, VSE/POWER uses three separate work queues: the RDR queue containing jobs, the LST queue containing output for printers, and the PUN queue containing output for punches. In each queue 36 classes (A–Z, 0–9) are defined, and each class is divided into a dispatchable part (disposition 'D' or 'K') and a non-dispatchable part (disposition 'H' or 'L'). All jobs and outputs are added into the matching queue and class according their type (job or output), class, disposition, and priority. For local processing, VSE/POWER dispatches queue entries only from the dispatchable part of the matching class chains.

Additionally, a forth queue, the XMT queue, is used to organize transfer of jobs and outputs to other systems connected to VSE/POWER via an NJE link. Connection to other NJE nodes is driven by VSE/POWER networking (PNET).

VSE/POWER handles the following aspects of batch processing for z/VSE:

- Receive jobs into the operating system.
- Schedule them for processing by z/VSE.
- Control their output processing.

The most basic functions are as follows:

- Accept jobs submitted in various ways:
  - From IUI and ICCF through the SUBMIT function/command
  - Over a network using the Network Job Entry protocol (NJE)

  - From a running program, which can submit other jobs through the Cross Partition Communication Control interface (XPCC)
  - From a card reader (emulated by z/VM)
  - From a Remote Job Entry workstation (RJE)
- Queue jobs waiting to be executed in the VSE/POWER RDR queue.
- Pass jobs to job control, which is a system program that handles job execution.

Job control
A system program that manages the running of batch jobs, one at a time

- Accept printed and punched output from the job while it is running and queue the output according to the specification of the * $$ LST statement, respectively, the * $$ PUN statement.
- Optionally, send output to a printer or a punch, or save it on *spool* for PSF, InfoPrint, or another output manager to retrieve.

VSE/POWER uses two disk data sets for *spooling*. The VSE/POWER queue file contains the processing information derived from the * $$ JOB/* $$LST/*$$ PUN statement and links to the data of the job or output that is stored in the VSE/POWER data file. In a small z/VSE system the spool data sets might be a few hundred cylinders of disk space. In a large installation the data file might be up to 32 complete volumes of disk space.

Spool simply means to queue and hold data in card-image format (for input), printed format (for list output), or card-image format (for punch output). The basic elements of batch processing are shown in Figure 7-1.

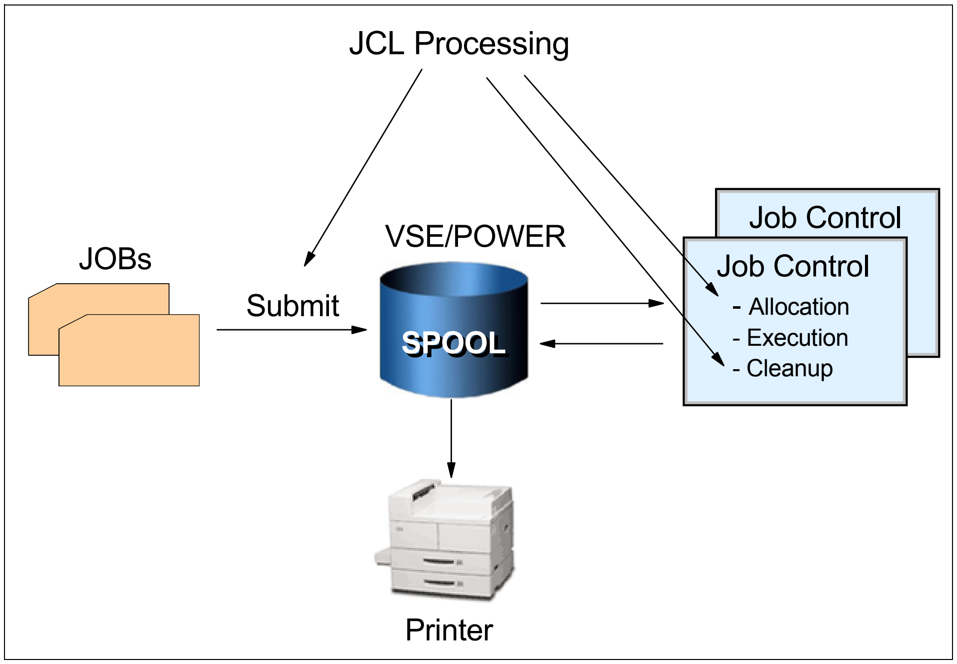


Figure 7-1 Basic batch flow

Job control is an integral part of z/VSE that reads, interprets, and executes the JCL. It is normally running in all active partitions. Job control manages the running of batch jobs, one at a time, in the same partition. If ten partitions are active, then ten batch jobs can run at the same time. VSE/POWER handles JECL processing and feeds the job cards to job control, which does the key JCL work.

The jobs in Figure 7-1 represent JCL and perhaps data intermixed with the JCL. Source code input for a compiler is an example of data (the source statements) that might be intermixed with JCL. Another example is an accounting job that prepares the weekly payroll for different divisions of a firm (presumably, the payroll application program is the same for all divisions, but the input and master summary files may differ).

The diagram represents the jobs as punched cards (using the conventional symbol for punched cards), although real punched card input is very rare now. Typically, a job consists of card images (80-byte fixed-length records).

7.3 Static and dynamic partitions

All jobs in z/VSE are executed in partitions. A partition defines an area of virtual storage available for program execution. Twelve partitions are started during IPL, namely BG, F1, F2, F3,…, FA, FB, and controlled by VSE/POWER with the exception of F1, where VSE/POWER is executing itself, and FB, where the Basic Security Manager (BSM) is running. When a partition has been started under VSE/POWER control, between 1 and 4 VSE/POWER job classes have been associated with the partition together with a list of devices, which will be emulated by VSE/POWER. This list contains 1 card reader, up to 14 printers, and up to 14 punches. Furthermore, job control has been loaded into the partition, and some basic definitions have been processed, like assignments for input and output devices, JCL options, library and dump definitions. Job control will then issue a request for the first input card from the card reader. VSE/POWER intercepts the request and handles it. It searches in its RDR queue for a job with a matching class. If a job is found, it is passed to job control card image by card image, thus emulating a card reader. If no job is found, an empty card reader is emulated to job control.

In the past the number of 12 partitions were no longer sufficient for the growing VSE workload, and new partitions were added to the system. The new partitions are called dynamic partitions (and the 12 partitions defined at IPL are now referred to as static partitions). The dynamic partitions are started and stopped by VSE/POWER for a single VSE/POWER job by use of the definitions contained in the *dynamic class table*. Example 7-1 shows the VSE/POWER display of such a table. Dynamic partitions are grouped into classes, and for each class the table defines the maximum number of active partitions allowed working in parallel, the allocation values, the number of Logical Unit Blocks (LUBS), and a profile with basic definitions which is executed when the partition is started. It contains assignments for input and output devices, JCL options, library and dump definitions. Up to 23 different classes can be specified (C–E, G–Z) in the dynamic class table, and different dynamic class tables can be generated for different workloads.

Example 7-1 Dynamic class table

```
pdisplay dync
AR 0015 1C39I COMMAND PASSED TO VSE/POWER
F1 0001 1Q6AI  ******   ACTIVE DYNAMIC CLASS TABLE DTR$DYNC.Z    ******
F1 0001 1Q6AI  CLS STATE   ACT/MAX ALLOC    SIZE  SP-GETV  PROFILE LUBS
F1 0001 1Q6AI   C  ENAB      1  9     1M    500K    128K   STDPROF   50
F1 0001 1Q6AI   P  ENAB      0 32     1M    512K    128K   PWSPROF   50
F1 0001 1Q6AI   R  ENAB      2  3     8M   1024K    128K   STDPROF  100
F1 0001 1Q6AI   S  ENAB      0  2    15M   1024K    128K   STDPROF  100
F1 0001 1Q6AI   Y  ENAB      0  8     3M   1024K    128K   STDPROF   50
F1 0001 1Q6AI   Z  ENAB      0  3     5M   1024K    128K   STDPROF   50
```

When a job becomes available in a class defined in the dynamic class table, VSE/POWER verifies first that the maximum amount of active partitions (for that particular class) is not yet reached. Then a request is sent to the VSE supervisor to start a new dynamic partition for the selected class. The supervisor establishes the partition and loads job control into it, which executes the profile. Now VSE/POWER can pass the job cards to job control for processing. When the job has been passed completely and job control asks for another job, the dynamic partition is stopped instead by VSE/POWER sending a request to the VSE supervisor.

7.4 What job control does

To run multiple jobs asynchronously, the system must perform a number of functions:

1. Select jobs from the input queues (VSE/POWER does this).
2. Ensure that multiple jobs do not conflict in data set usage.
3. Ensure that single-user devices, such as tape drives or printers, are allocated correctly.
4. Find the executable programs requested for the job.
5. Clean up after the job ends and then request the next job.

Most of this work is done by job control, based on JCL information for each job. The primary purpose of JCL is to tell job control what is needed for the job.

7.5 Job and output management with VSE/POWER

Let us further explore the VSE/POWER concept through the following scenario.

Imagine that you are a z/VSE application programmer and you are developing a program for non-skilled users. Your program is supposed to read a couple of files, write to another couple of files, and produce a printed report. This program will run as a batch job on z/VSE.

What sorts of functions are needed in the operating system to fulfill the requirements of your program? And how will your program access those functions?

First, you need a sort of special language to inform the operating system about your needs. On z/VSE, this is *Job Control Language* (JCL). The use of JCL is covered in detail in Chapter 6, “Using JCL” on page 159, but for now assume that

JCL provides the means for you to request resources and services from the operating system for a batch job.

Specifications and requests you might make for a batch job include the functions you need to compile and execute the program, and allocate storage for the program to use as it runs.

With JCL, you can specify the following:

- Who you are (important for security reasons).
- Which resources (programs, files, memory) and services are needed from the system to process your program. You might, for example, need to do the following:
- Load the compiler code in memory.
- Make accessible to the compiler your source code. That is, when the compiler asks for a read, your source statements are brought to the compiler memory.
- Allocate some amount of memory to accommodate the compiler code, I/O buffers, and working areas.
- Make accessible to the compiler an output disk data set to receive the object code, which is usually referred to as the *object deck* (OBJ).
- Make accessible to the compiler a print file where it will tell you your eventual mistakes.
- Conditionally, have z/VSE load the newly created object deck into memory (but skip this step if the compilation failed).
- Allocate some amount of memory for your program to use.
- Make accessible to your program all of the input and output files.
- Make accessible to your program a printer for eventual messages.

In turn, you require the operating system to:

- Understand JCL (correcting eventual errors).
- Convert JCL to control blocks that describe the required resources.
- Allocate the required resources (programs, memory, files).
- Schedule the execution on a timely basis, for example, your program only runs if the compilation succeeds.
- Free the resources when the program is done.

The parts of z/VSE that perform these tasks are VSE/POWER and job control.

Think of VSE/POWER as the manager of the jobs waiting in a queue. It manages the priority of the set of jobs and their associated input data and output results.

Job control uses the statements on the JCL cards to specify the resources required of each individual job once it has been released (dispatched) by VSE/POWER.

Your JCL, as described, is called a *job*—in this case formed by two sequential steps, the compilation and execution. The steps in a job are always executed sequentially. The job must be submitted to VSE/POWER in order to be executed.

Example 7-2 shows a JCL procedure that can compile, link-edit, and execute a program.

1. The first step identifies the High Level Assembler as declared in // EXEC ASMA90. The statement `* $$ SLI ICCF=(SOURCE),LIB=(0019,0002)` describes the input of the compiler (the source).
2. The generated object deck is the input for the second step, which performs link-editing (through the program LNKEDT). Link-editing is needed to resolve external references and *bring in* or *link* the previously developed common routines (a type of code re-use). `// OPTION …,LINK` informs the Linkage Editor to store the generated phase temporarily in the Virtual I/O area, and the `// EXEC` statement loads the generated phase into the partition for execution.
3. In the third step, the program is executed.

Example 7-2 Job to compile, link-edit, and execute a program

```
* $$ JOB JNM=COMPRUN,DISP=D,CLASS=S
* $$ LST LST=FEE,DISP=D,CLASS=Q,PRI=3
// JOB COMPRUNC COMPILE PROGRAM SOURCE AND EXECUTE IT
// OPTION ERRS,SXREF,SYM,NODECK,LINK
// EXEC ASMA90,SIZE=(ASMA90,64K),PARM='EXIT(LIBEXIT(EDECKXIT)),SIZE(MAXC
               -200K,ABOVE)'
* $$ SLI ICCF=(SOURCE),LIB=(0019,0002)
/*
// EXEC LNKEDT,SIZE=256K
// EXEC
/&
* $$ EOJ
```

`* $$ JOB JNM=COMPRUN,DISP=D,CLASS=S` specifies the VSE/POWER job name, its disposition D (start job when partition active for specified class is found and delete job after execution), and its class (S).

`* $$ LST LST=FEE,DISP=D,CLASS=Q,PRI=3` specifies options for output spooled for printer FEE and is interpreted by VSE/POWER during job execution.

`* $$ SLI ICCF=(SOURCE),LIB=(0019,0002)` is handled by VSE/POWER during job execution. VSE/POWER reads the member SOURCE from ICCF Library 19 or 02 and passes it to job control.

`* $$ EOJ` specifies the end of the VSE/POWER job.

When VSE/POWER reads in these statements it analyses and removes the * $$ JOB and EOJ statements and creates an entry in the queue file (RDR queue), which keeps the processing information (class S, disposition D). The statements 0002 to 00011 are kept in the data file part of the job. Whenever a new job with Disp=D (or DISP=K) is added to the RDR queue, VSE/POWER first checks the dynamic class table, if a partition for class S can be started. If no dynamic partition can be started, VSE/POWER scans all active static partitions whether or not they are waiting for work for class S. If no active waiting partition is found, the job will not be dispatched.

During the execution of a step, the program is controlled by job control, not by VSE/POWER (Figure 7-2). Also, a spooling function is needed at this point in the process.

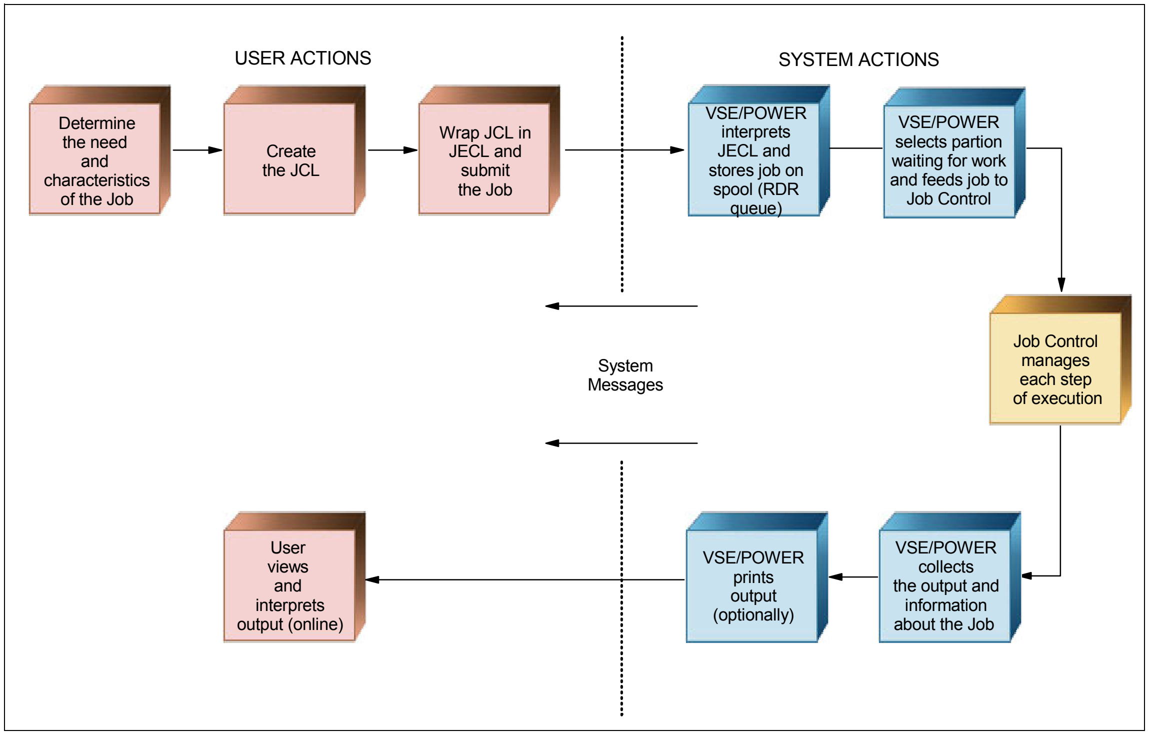

USER ACTIONS
SYSTEM ACTIONS
Determine the need and characteristics of the Job
Create the JCL
Wrap JCL in JECL and submit the Job
VSE/POWER interprets JECL and stores job on spool (RDR queue)
VSE/POWER selects partion waiting for work and feeds job to Job Control
Job Control manages each step of execution
System Messages
VSE/POWER collects the output and information about the Job
VSE/POWER prints output (optionally)
User views and interprets output (online)

Figure 7-2 Related actions with the job control

Spooling is the means by which the system manipulates its work, including:

- Using storage on *direct access storage devices* (DASD) as buffer storage to reduce processing delays when transferring data between peripheral equipment and a program to be run
- Reading and writing input and output streams on an intermediate device for later processing or output
- Performing an operation such as printing while the computer is busy with other work

There are two sorts of spooling: input and output. Both improve the performance of the program reading the input and writing the output. Both are done automatically by VSE/POWER.

Later, when the program is executed and asks to read this data, VSE/POWER picks up the records in the spool and delivers them to the program (at disk speed).

7.6 Job flow through the system

Let us look in more detail at how a job is processed through the combination of VSE/POWER and job control

During the life of a job, VSE/POWER and the base control program of z/VSE control different phases of the overall processing. The RDR queue contains dispatchable jobs that are waiting to run, jobs that are currently running, and non-dispatchable jobs (disposition ‘H’ or ‘L’) that are waiting for being set as dispatchable. For each class a set of dispatchable jobs and another set of non-dispatchable jobs may exist.

Generally speaking, a job goes through the following phases:

- Input
- Processing
- Output
- Print/punch (hard copy)
- Purge for job
- Purge for output

Figure 7-3 shows the different phases of a job during batch processing.

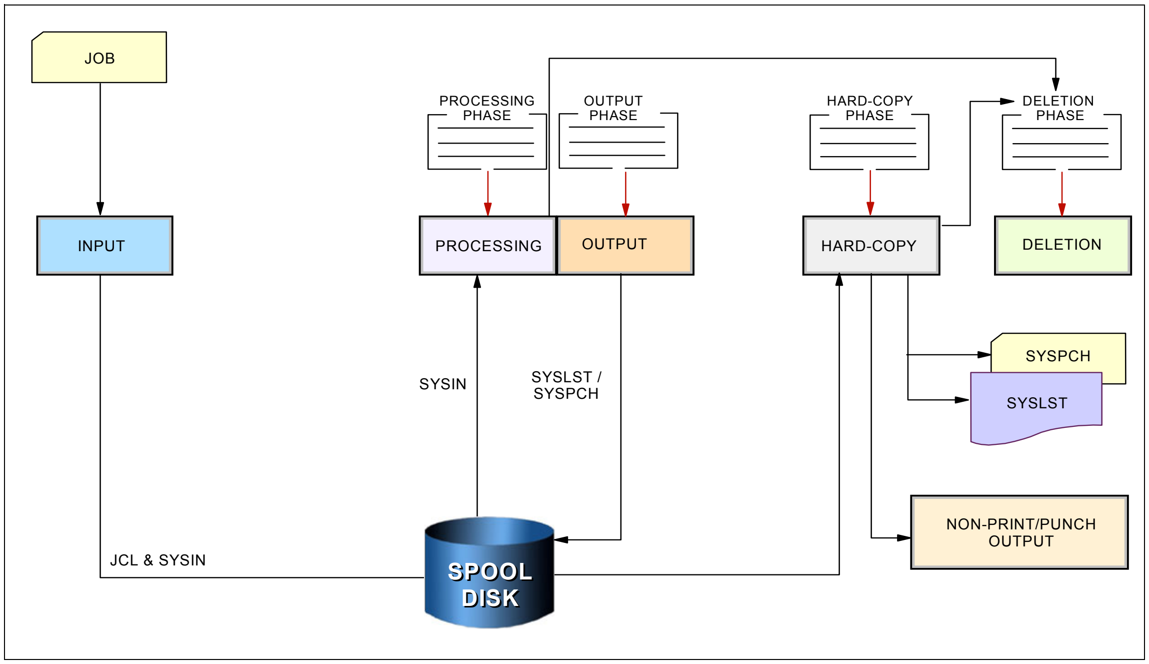


Figure 7-3 Job flow through the system

The job flow, as shown in Figure 7-3, is:

1. Input phase

   VSE/POWER accepts jobs, in the form of an input stream, from input devices, from other programs through XPCC, and from other nodes in a job entry network. VSE/POWER accepts LST/PUN outputs from other network nodes and from XPCC programs.

   XPCC uses the Spool Access Support interface to submit jobs, outputs, and commands to VSE/POWER. Any job running in z/VSE can use XPCC to pass an input stream to VSE/POWER. VSE/POWER can receive multiple jobs and outputs simultaneously through multiple XPCC communication paths.

   VSE/POWER reads the input stream and builds a queue entry for the RDR queue for a job (and for the LST or PUN queue for an output). If the job or output is not destined for local execution, VSE/POWER will put the queue entry into the XMT (transmit) queue. VSE/POWER assigns a job number to the job/output and extracts information like job name, class, priority, and target user from the * $$ JOB statement. VSE/POWER places the job's JCL, optional VSE/POWER JECL control statements, and SYSIN data onto the VSE/POWER data file. VSE/POWER then selects jobs from the RDR queue for running. (Output from the LST/PUN queue is selected for printing/punching.)

2. Processing phase

   In the processing phase, VSE/POWER responds to requests for jobs from job control. VSE/POWER selects jobs that are waiting to run (dispatchable) from the RDR queue and sends them to the partitions.

   Job control is a system program belonging to z/VSE, which starts a job allocating the required resources to allow it to compete with other jobs that are already running.

   Job control is started by the operator or by VSE/POWER automatically when the system initializes a partition. The installation associates each partition with one or more job classes in order to obtain an efficient use of available system resources. VSE/POWER selects jobs whose classes match the class assigned to the partition where job control is requesting a new job, obeying the priority of the queued jobs.

3. Output phase

   VSE/POWER controls output processing. All output written to POWER-controlled printers and punches is collected by VSE/POWER. Separate LST and PUN outputs are built, one for each device. This output includes system messages that must be printed, as well as data sets requested by the user that must be printed or punched. During job execution VSE/POWER assigns the characteristics of the job's output by analyzing the * $$ LST and * $$ PUN statements in the job stream. VSE/POWER queues the output for print or punch processing.

4. Hardcopy phase

   VSE/POWER selects output for local processing from the output queues (LST and PUN queue) by disposition ('D' or 'K'), output class, priority, and other criteria. Output for other nodes is stored in the XMT queue and is handled by the VSE/POWER networking function (PNET). After processing of the output, VSE/POWER either deletes it for disposition 'D' or changes the disposition from 'K' to 'L'.

Delete
Releasing the spool space assigned to a job, when the job completes

5. Deletion phase

   When all processing for a job completes, VSE/POWER releases the spool space assigned to the job or output, making the space available for allocation to subsequent jobs and outputs.

7.7 Summary

Batch processing is the most fundamental function of z/VSE. Many batch jobs are run in parallel, and JECL and JCL are used to control the operation of each job. Correct use of JECL and JCL parameters allows parallel, asynchronous execution of jobs that may need access to the same data sets.

Job control is a system program that processes JCL, sets up the necessary environment in a VSE/AF partition, and runs the batch job in the same partition. Job control is reloaded when the executing program stops itself.

A goal of an operating system is to process work while making the best use of system resources. To achieve this goal, resource management is needed during key phases:

- Before job processing, reserve input and output resources for jobs.
- During job processing, manage spooled SYSIN and SYSOUT data.
- After job processing, free all resources used by the completed jobs, making the resources available to other jobs.

z/VSE shares with VSE/POWER the management of jobs and resources. VSE/POWER receives jobs into the system, schedules them for processing by z/VSE, and controls their output processing. VSE/POWER is the manager of the jobs waiting in a queue. It manages the priority of the jobs and their associated input data and output results. Job control uses the statements in the JCL records to specify the resources required of each individual job after it is released (dispatched) by VSE/POWER.

During the life of a job, both VSE/POWER and the z/VSE base control program control different phases of the overall processing. Jobs and their outputs are managed in queues. Jobs that are waiting to run or are running (RDR queue), their output is waiting for printing or punching (LST / PUN queue), and when finished both jobs and output are waiting to be purged from the system (DEL queue).

Key terms in this chapter		
Batch job	Class	Execution
Job control	Job entry and spooling subsystem	Output
VSE/POWER	Disposition (DISP)	Static partition
Dynamic partition	Queue	Spool

7.8 Questions for review

To help test your understanding of the material in this chapter, complete the following questions:

1. What is batch processing?
2. What are two defining characteristics of z/VSE that are discussed in this chapter?
3. Why does z/VSE need VSE/POWER?
4. During the life of a job, what types of processing does VSE/POWER typically perform?

7.9 Exercises

1. To build a simple VSE/POWER job:
   a. Access the ICCF editor and create a VSE/POWER job. Specify a job name in the * $$ JOB card. This job should run in partition F6 only and should be deleted after execution. What JOB card operands must be specified to achieve this?
   b. The job should execute a PAUSE statement, then program LSERV, and then another PAUSE statement.
2. To display the status of a job:
   a. Submit the job.
   b. While the job executes the PAUSE statement, use the VSE/POWER PDISPLAY A command and note the output produced by the job.
   c. Use Interactive interface to look at the job and at its output.
   d. End the PAUSE statement.
3. To start a job at a certain time:
   a. Modify the VSE/POWER job, remove the PAUSE statements, and add DUETIME=hhmm to the * $$ JOB card.

      Specify hhmm = current time + 10 Minutes and submit the job.
   b. Display the job in the RDR queue by PDISPLAY RDR. What happens after 10 minutes?
4. To delete a job:
   a. Determine the jobname and jobnumber of the output.
   b. Delete it by PDELETE LST,jobname,jobnumber.

Part 2

Application programming on z/VSE

In this part we introduce the tools and utilities for developing a simple program to run on z/VSE. The chapters that follow guide you through the process of application design, choosing a programming language, and using a runtime environment.

8

Designing and developing applications for z/VSE

Objective: As your company's newest z/VSE application designer or programmer, you will be asked to design and write new programs, or modify existing programs to meet your company's business goals. Such an undertaking will require that you fully understand the various user requirements for your application and know which z/VSE system services to exploit.

This chapter provides a brief review of the common design, code, and test cycle for a new application. Much of this information is applicable to all computing platforms in general, not just mainframes.

After completing this chapter, you will be able to:

- Describe the roles of the application designer and application programmer.
- List the major considerations for designing an application for z/VSE.
- Describe the advantages and disadvantages of using batch versus online for an application.
- Briefly describe the process for testing a new application on z/VSE.
- List three advantages for using z/VSE as the host for a new application.

8.1 Application designers and programmers

The tasks of *designing* an application and *developing* one are distinct enough to treat each in a separate textbook. In larger z/VSE sites, separate departments might be used to carry out each task. This chapter provides an overview of these job roles and shows how each skill fits into the overall view of a typical application development life cycle on z/VSE.

The application designer is responsible for determining the best programming solution for an important business requirement. The success of any design will depend in part on the designer's knowledge of the business itself, awareness of other roles in the mainframe organization such as programming and database design, and an understanding of the business's hardware and software. In short, the designer must have a global view of the entire project.

Another role involved in this process is the business systems analyst. This person is responsible for working with users in a particular department (accounting, sales, production control, manufacturing, and so on) to identify business needs for the application. Like the application designer, the business systems analyst requires a broad understanding of the organization's business goals, and the capabilities of the information system.

Application
A set of files that make up software for the user

The application designer gathers requirements from business systems analysts and end users. The designer also determines which IT resources will be available to support the application. The application designer then writes the design specifications for the application programmers to implement.

The application programmer is responsible for developing and maintaining application programs. That is, the programmer builds, tests, and delivers the application programs that run on the mainframe for the end users. Based on the application designer's specifications, the programmer constructs an application program using a variety of tools. The build process includes many iterations of code changes and compiles, application builds, and unit testing.

During the development process, the designer and programmer must interact with other roles in the enterprise. The programmer, for example, often works on a team with other programmers who are building code for related application modules.

When the application modules are completed, they are passed through a testing process that can include functional, integration, and system tests. Following this testing process, the application programs must be acceptance-tested by the user community to determine whether the code actually accomplishes what the users desire.

Besides creating new application code, the programmer is responsible for maintaining and enhancing the company's existing mainframe applications. In fact, this is frequently the primary job for many application programmers on the mainframe today. While many mainframe installations still create new programs with COBOL or PL/I, languages such as Java have become popular for building new applications on the mainframe, just as on distributed systems.

8.2 Designing an application for z/VSE

During the early design phases, the application designer makes decisions regarding the characteristics of the application. These decisions are based on many criteria, which must be gathered and examined in detail to arrive at a solution that is acceptable to the user. The decisions are not independent of each other, in that one decision will have an impact on others and all decisions must be made taking into account the scope of the project and its constraints. Figure 8-1 shows the context of VSE applications as an example.

Design
The task of determining the best programming solution for a given business requirement

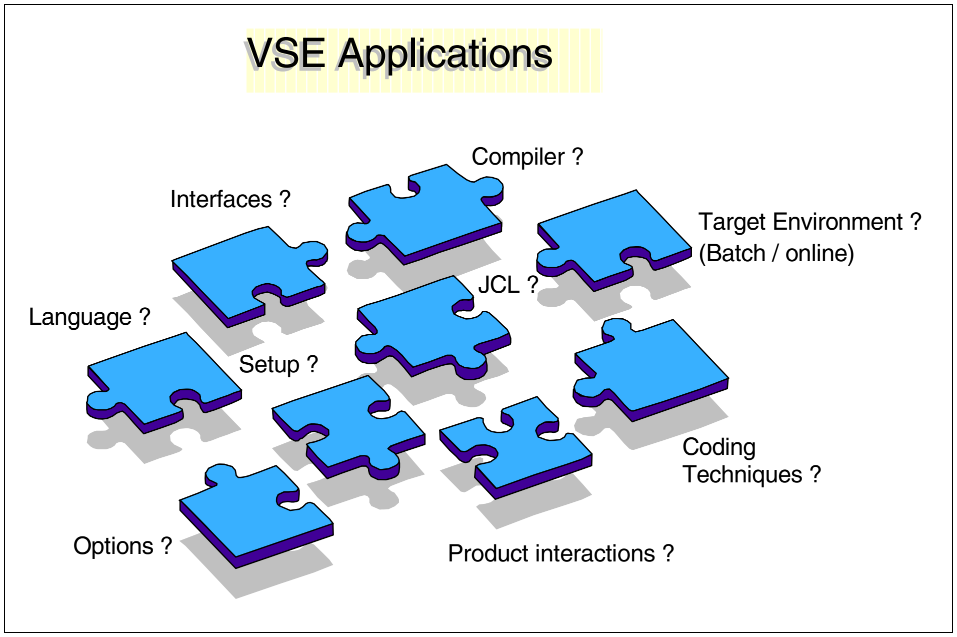


Figure 8-1 VSE application context

Designing an application to run on z/VSE shares many of the steps followed for designing an application to run on other platforms, including the distributed environment. z/VSE, however, introduces some special considerations. This chapter provides examples of the decisions that the z/VSE application designer makes during the design process for a given application. The list is not meant to be exhaustive, but rather to give you an idea of the process involved:

- Designing for z/VSE: batch or online?
- Designing for z/VSE: data sources and access methods
- Designing for z/VSE: availability and workload requirements
- Designing for z/VSE: exception handling

Beyond these decisions, other factors that might influence the design of a z/VSE application might include the choice of one or more programming languages and development environments. Other considerations discussed in this chapter include the following:

- Using mainframe character sets in "Using the EBCDIC encoding scheme" on page 205
- Use of an interactive development environment (IDE) in "Using application development tools" on page 208
- Differences between the various programming languages in Chapter 9, "Using programming languages on z/VSE" on page 217

Keep in mind that the best designs are those that start with the end result in mind. We must know what it is that we are striving for before we start to design.

8.2.1 Designing for z/VSE: batch or online?

When designing an application for z/VSE and the mainframe, a key consideration is whether the application will run as a batch program or an online program. In some cases, the decision is obvious, but most applications can be designed to fit either paradigm. How, then, does the designer decide which approach to use?

Reasons for using batch or online:

- Reasons for using batch
  - Data is stored on tape.
  - Transactions are submitted for overnight processing.
  - User does not require online access to data.
- Reasons for using online
  - User requires online access to data.
  - High response time requirements.

8.2.2 Designing for z/VSE: data sources and access methods

Here, the designer's considerations typically include the following:

- What data must be stored?
- How will the data be accessed? This includes a choice of access method.
- Are the requests ad hoc or predictable?
- Will we choose VSE libraries, VSAM, or a database management system (DBMS) such as DB2?

8.2.3 Designing for z/VSE: availability and workload requirements

For an application that will run on z/VSE, the designer must be able to answer the following questions:

- What is the quantity of data to store and access?
- Is there a need to share the data?
- What are the response time requirements?
- What are the cost constraints of the project?
- How many users will access the application at once?
- What is the availability requirement of the application (24 hours a day 7 days a week or 8:00 a.m. to 5:00 p.m. weekdays, and so on)?

8.2.4 Designing for z/VSE: exception handling

Are there any unusual conditions that might occur? If so, we need to incorporate these in our design in order to prevent failures in the final application. We cannot always assume, for example, that input will always be entered as expected.

8.3 Application development life cycle: an overview

An application is a collection of programs that satisfies certain specific requirements (resolves certain problems). The solution could reside on any platform or combination of platforms, from a hardware or operating system point of view.

As with other operating systems, application development on z/VSE is usually composed of the following phases:

1. Design phase
   a. Gather requirements (user, hardware, and software requirements).

   b. Perform analysis.
   c. Develop the design in its various iterations:
      - High-level design
      - Detailed design
   d. Give the design to application programmers.

Develop
Build, test, and deliver an application program.

2. Code and test the application.
3. Perform user tests. User tests application for functionality and usability.
4. Perform system tests.
   a. Perform integration test. (Test application with other programs to verify that all programs continue to function as expected.)
   b. Perform performance (volume) test using production data.
5. Production - Give to operations. Ensure that all documentation is in place (user training, operation procedures).
6. Maintenance phase - ongoing day-to-day changes and enhancements to application.

Figure 8-2 shows the process flow during the various phases of the application development life cycle.

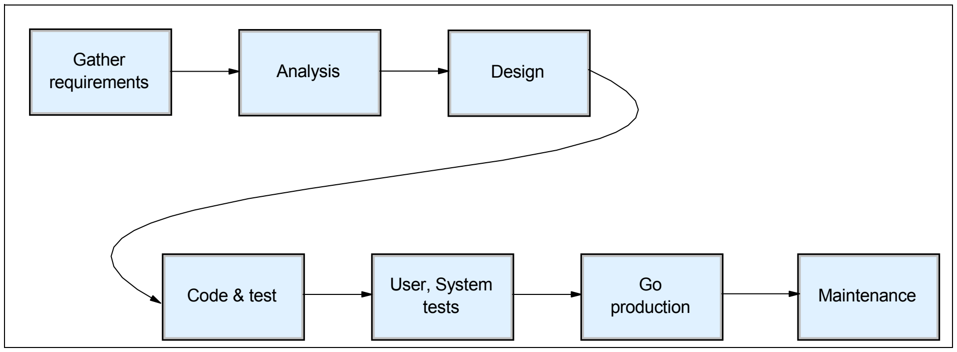


Figure 8-2 Application development life cycle

Figure 8-3 depicts the design phase up to the point of starting development. Once all of the requirements have been gathered, analyzed, verified, and a design has been produced, we are ready to pass on the programming requirements to the application programmers.

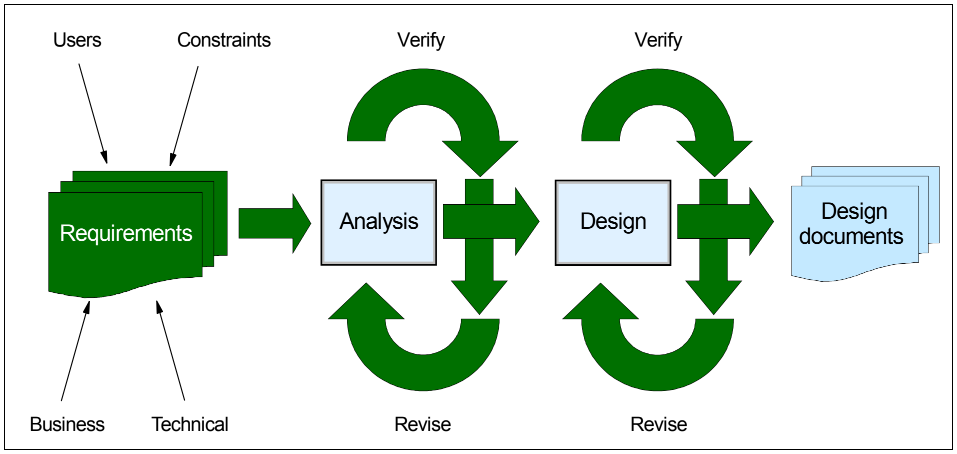


Figure 8-3 Design phase

The programmers take the design documents (programming requirements) and then proceed with the iterative process of coding, testing, revising, and testing again, as we see in Figure 8-4.

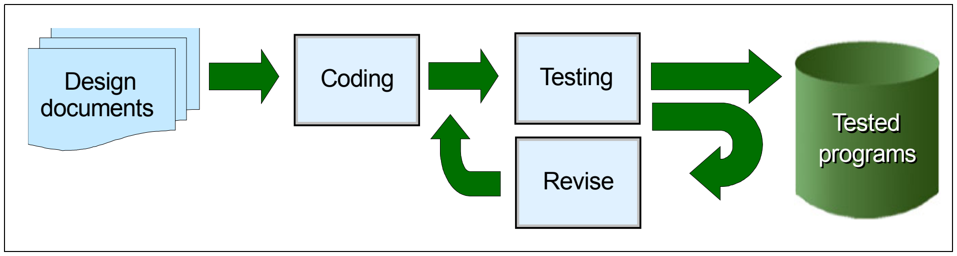


Figure 8-4 Development phase

After the programs have been tested by the programmers, they will be part of a series of formal user and system tests. These are used to verify usability and functionality from a user point of view, as well as to verify the functions of the application within a larger framework (Figure 8-5).

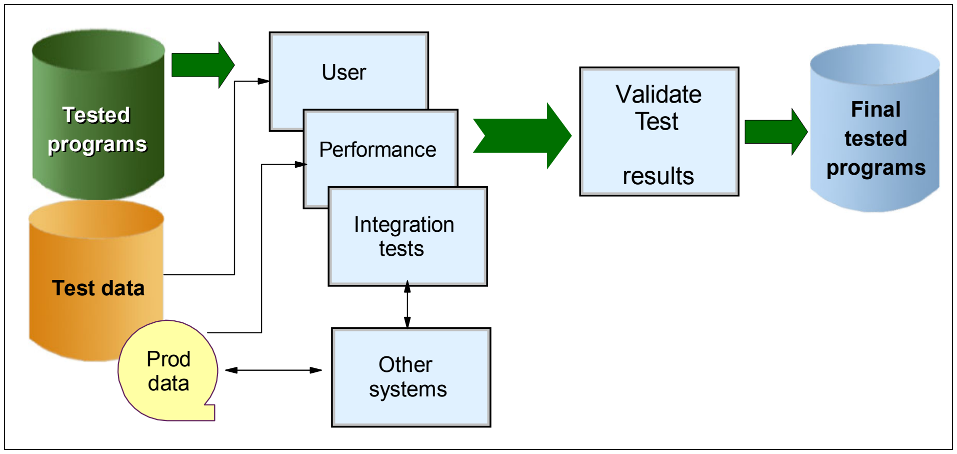


Figure 8-5 Testing

The final phase in the development life cycle is to go to production and become a steady state. As a prerequisite to going to production, the development team needs to provide documentation. This usually consists of user training and operational procedures. The user training familiarizes the users with the new application. The operational procedures documentation enables operations to take over responsibility for running the application on an on-going basis.

In production, the changes and enhancements are handled by a group (possibly the same programming group) that performs the maintenance. At this point in the life cycle of the application, changes are tightly controlled and must be rigorously tested before being implemented into production (Figure 8-6).

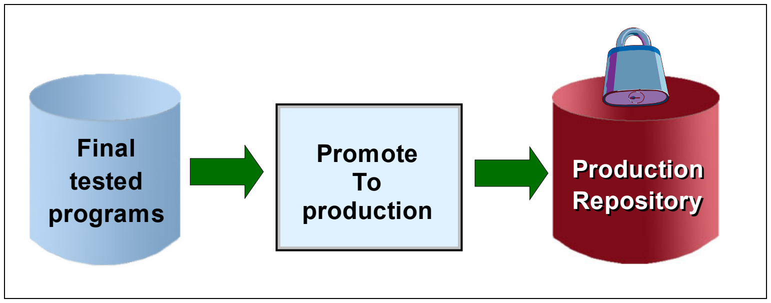


Figure 8-6 Production

As mentioned before, to meet user requirements or solve problems, an application solution might be designed to reside on any platform or a combination of platforms. As shown in Figure 8-7, our specific application can be located in any of the three environments: Internet, enterprise network, or central site. The operating system must provide access to any of these environments.

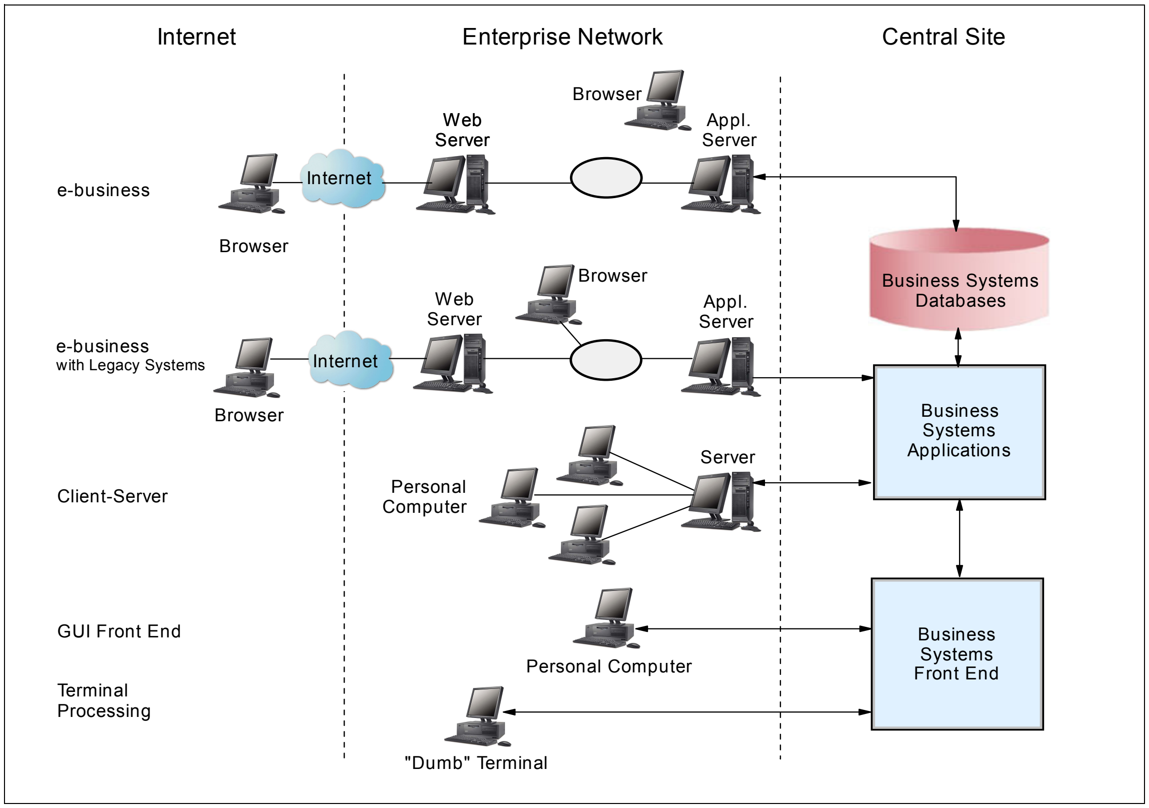


Figure 8-7 Growing infrastructure complexity

To begin the design process, we must first assess what we need to accomplish. Based on the constraints of the project, we determine how and with what we will accomplish the goals of the project. To do so, we conduct interviews with the users (those requesting the solution to a problem), as well as the other stakeholders.

The results of these interviews should inform every subsequent stage of the life cycle of the application project. At certain stages of the project, we again call upon the users to verify that we have understood their requirements and that our solution meets their requirements. At these milestones of the project, we also ask the users to sign off on what we have done, so that we can proceed to the next step of the project.

8.3.1 Gathering requirements for the design

When designing applications, there are many ways to classify the requirements: functional requirements, non-functional requirements, emerging requirements, system requirements, process requirements, constraints on the development and in the operation—to name a few.

Computer applications operate on data, which resides somewhere and that needs to be accessed from either a local or a remote location. The applications manipulate the data, performing some kind of processing on it, and then present the results to whomever was asking for in the first place.

This simple description involves many processes and many operations that have many different requirements, from computers to software products.

Although each application design is a separate case and can have many unique requirements, some of these are common to all applications that are part of the same system, not only because they are part of the same set of applications that comprise a given information system, but also because they are part of the same installation, which is connected to the same external systems.

Platform
Often refers to an operating system, implying both the OS and the hardware (environment)

One of the problems faced by systems as a whole is that components are spread across different machines, different platforms, and so forth, each one performing its work in a *server farm* environment.

An important advantage to the System z approach is that applications can be maintained using tools that reside on the mainframe. Some of these mainframe tools make it possible to have different platforms sharing resources and data in a coordinated and secure way according to workload or priority.

The following is a list of the various types of requirements for an application. The list is not exclusive. Some items already include others.

- Accessibility
- Irrecoverably
- Serviceability
- Availability
- Security
- Connectivity
- Performance objectives
- Resource management
- Usability
- Frequency of data backup

- Portability
- Web services
- Changeability
- Inter-communicable
- Failure prevention and fault analysis

8.4 Developing an application on the mainframe

After the analysis has been completed and the decisions have been made, the process passes on to the application programmer. The programmer is not given free rein, but rather must adhere to the specifications of the designer. However, given that the designer is probably not a programmer, there may be changes required because of programming limitations. But at this point in the project, we are not talking about design changes, merely changes in the way the program does what the designer specified it should do.

The development process is iterative, usually working at the module level. A programmer will usually follow this process:

1. Code a module.
2. Test a module for functionality.
3. Make corrections to the module.
4. Repeat from step 2 until successful.

After testing has been completed on a module, it is signed off and effectively frozen to ensure that if changes are made to it later, it will be tested again. When sufficient modules have been coded and tested, they can be tested together in tests of ever-increasing complexity.

This process is repeated until all of the modules have been coded and tested. Although the process diagram shows testing only after development has been completed, testing is continuously occurring during the development phase.

8.4.1 Using the EBCDIC encoding scheme

z/VSE text files are encoded in the Extended Binary Coded Decimal Interchange (EBCDIC) encoding scheme. This is an 8-bit scheme that was developed by IBM before the 7-bit American Standard Code for Information Interchange (ASCII) became commonly used. 7-bit ASCII was based on the English character set and provided little space for national characters in other languages. Therefore it was expanded later to the 8-bit extended ASCII scheme, which is used in most systems that you are familiar with.

Both EBCDIC and ASCII allow the definition of national characters. Therefore, various countries defined their own EBCDIC and ASCII character encoding tables, the so-called code pages.

You need to be aware of the difference in encoding schemes and code pages when moving data from ASCII-based systems to EBCDIC-encoded systems. When transferring executable programs these must not be converted and a binary transfer must be specified.

However, when text data is transferred, the correct code pages must be defined for the sending and the receiving system. Converting simple character strings between ASCII and EBCDIC is trivial, and conversion tables are available for most languages and code pages. The situation is more difficult if the character being converted is not available in the target code page, for example, if the text is converted to the code page of another language.

A mapping of EBCDIC and 7-bit ASCII code point[1] assignments is presented in Appendix C, "EBCDIC - 7-bit ASCII table" on page 435, and might be useful for this discussion. The difference between ASCII and EBCDIC is the way they assign bits for specific characters. The following are a few examples:

Character	EBCDIC	ASCII
A	11000001 (x'C1')	01000001 (x'41')
B	11000010 (x'C2')	01000010 (x'42')
a	10000001 (x'81')	01100001 (x'61')
1	11110001 (x'F1')	00110001 (x'31')
space	01000000 (x'40')	00100000 (x'20')

A code page has a binary collating sequence, corresponding to the binary value of the character bits. For example, A has a lower value than B in both ASCII and EBCDIC. The binary collating sequence is important for sorting and for almost any program that scans and manipulates character strings. While the results are predictable and useful in some contexts, they are incorrect and unacceptable in sorting where a cultural sort order is expected. This order is defined by the language settings of the system and needs to be respected by sort programs. The main difference between the binary collating sequence in EBCDIC and ASCII is as follows:

	EBCDIC	ASCII
Lowest value:	space	space
	punctuation	punctuation
	lower case	numbers
	upper case	upper case
Highest value:	numbers	lower case

[1] A code point is a unique bit pattern that represents a character in a code page.

For example, *a* is less than *A* in EBCDIC, but *a* is greater than *A* in ASCII. Numeric characters are less than any alphabetic letter in ASCII but are greater than any letter in EBCDIC. A–Z and a–z are two contiguous sequences in ASCII. In EBCDIC there are gaps between some letters.

Traditional mainframe programming does not use special characters to terminate fields. In particular, nulls and new line characters (or CL/LF character pairs) are not used. There is no concept of a *binary* versus a *text* file. Bytes can be interpreted as EBCDIC or ASCII or something else if programmed properly. If such files are sent to a mainframe printer, it will attempt to interpret them as EBCDIC characters because the printer is sensitive to the character set. Providing that no one attempts to print the ASCII files on a mainframe printer (or display them on a 3270), the system does not care what character set is being used.

The Asian languages Chinese, Japanese, Korean, and Thai use double-byte character sets (DBCS) on the mainframe and on the PC. Each character is represented by 2*8 bits.

8.4.2 Unicode on mainframe

Unicode, an industry standard to simplify data exchange between different systems and different languages, is a 16-bit character set intended to represent text and symbols in all modern languages and IT protocols. Mainframes, PCs, and various RISC systems use the same Unicode assignments.

Unicode is maintained by the Unicode Consortium:

http://www.unicode.org/

There is increasing use of Unicode in mainframe applications. The latest System z mainframes include a number of unique hardware instructions for Unicode. At the time of writing, Unicode usage on mainframes is primarily in Java. However, z/VSE middleware products and protocols (for example, WebServices, XML, and HTTP) are also beginning to use Unicode, and this is certainly an area of change for the near future.

8.4.3 Interfaces for z/VSE application programmers

When operating systems are developed to meet the needs of the computing marketplace, applications are written to run on those operating systems. Over the years, many applications have been developed that run on z/VSE and, more recently, UNIX.

The most common interface for z/VSE developers is the panel-driven Interactive Interface, using a 3270 terminal. Generally, developers use 3270 terminal emulators running on personal computers, rather than actual 3270 terminals. Emulators can provide developers with auxiliary functions, such as multiple sessions, and uploading and downloading code and data from the PC.

Program development on z/VSE typically involves the use of a line editor to manipulate source code files, the use of batch jobs for compilation, and a variety of mechanisms for testing the code. Interactive debuggers, based on 3270 terminal functions, are available for common languages. This chapter introduces the tools and utilities for developing a simple program to run on z/VSE.

Alternate methods are available in conjunction with various middleware products. For example, the WebSphere products provide GUI development facilities for personal computers.

8.4.4 Using application development tools

Producing well-tested code requires the use of tools on the mainframe. The primary tool for the programmer is the ICCF editor.

When developing traditional, procedural programs in languages such as COBOL and PL/I, the programmer often logs on to the mainframe and uses an IDE or the ICCF editor to modify the code, compile it, and run it.

Executable
A program file ready to run in a particular environment

When the source code changes are complete, the programmer submits a JCL file to compile the source code, build the application modules, and create an executable for testing. The programmer conducts *unit tests* of the functionality of the program. The programmer uses job monitoring and viewing tools to track the running programs, view the output, and make appropriate corrections to source code or other objects. Sometimes a program will create a *dump* of memory when a failure occurs. The programmer can also use tools to interrogate the dump output and to trace through executing code to identify the failure points.

Some mainframe application programmers have now switched to the use of Interactive Development Environment (IDE) tools to accelerate the edit/compile/test process. IDEs allow application programmers to edit, test, and debug source code on a workstation instead of directly on the mainframe system. The use of the IDE is particularly useful for building *hybrid* applications that employ host-based programs or transactional systems, but also contain a Web browser-like user interface.

After the components are developed and tested, the application programmer packages them into the appropriate deployment format and passes them to the team that coordinates production code deployments.

Application enablement services available on z/VSE include:

- Language Environment
- Debug Tool
- DFSORT
- HLASM Toolkit
- C language
- Traditional languages such as COBOL, PL/I, and Fortran

Java applications can access z/VSE resources from outside z/VSE. In order to develop Java applications exploiting VSE Connector Beans, the following Eclipse-based IBM tools can help:

- IBM WebSphere Studio Application/Enterprise Developer

  Allows professional application developers to quickly and easily build, test, integrate, and deploy Java and Java 2 Enterprise Edition applications

- Rational Application Developer for WebSphere Software.

  A comprehensive IDE for designing, developing, analyzing, testing, profiling, and deploying applications using Java, J2EE™, Web, Web services, service-oriented architecture, and portal technologies

8.4.5 Conducting a debugging session

The application programmer conducts a unit test to test the functionality of a particular module being developed. The programmer uses job monitoring and viewing software such as VSE/POWER and Interactive Interface to track the running compile jobs, view the compiler output, and verify the results of the unit tests. If necessary, the programmer makes the appropriate corrections to source code or other objects.

Sometimes a program will create a dump of memory when a failure occurs. When this happens, a z/VSE application programmer might use tools such as IBM Debug Tool for VSE/ESA (see “Using Debug Tool for VSE/ESA” on page 211) to trace through executing code to find the failure or misbehaving code.

A typical development session follows these steps:

1. Log on to z/VSE.
2. Enter ICCF and open/check the source code from the ICCF library (or where ever you have stored the source code).
3. Edit the source code to make necessary modifications.
4. Submit JCL to build the application and do a test run.
5. Switch to VSE/POWER to view the running job status.

6. View the job output in VSE/POWER to check for errors.
7. View the dump output to find bugs[2].
8. Re-run the compile/link/go job and view the status.
9. Check the validity of the job output.

Some mainframe application programmers have now switched to the use of Interactive Development Environment (IDE) tools to accelerate the edit/compile/test process. IDE tools such as the WebSphere Studio Enterprise Developer and Enterprise Generation Language (EGL) are used to edit source code on a workstation instead of directly on the host system, to run compiles *off-platform*, and to perform remote debugging.

The use of the IDE is particularly useful if hybrid applications are being built that employ host-based programs in COBOL or transaction systems such as CICS and DB2, but also contain a Web browser-like user interface. The IDE provides a unified development environment to build both the online transaction processing (OLTP) components in a high-level language and the HTML front-end user interface components. Once the components are developed and tested, they are packaged into the appropriate deployment format and passed to the team that coordinates production code deployments.

Transaction
An activity or request. They update master files for orders, changes, additions, and so on.

Besides new application code, the application programmer is responsible for the maintenance and enhancement of existing mainframe applications. In fact, this is the primary job for many high-level language programmers on the mainframe today. And, while most z/VSE customers are still creating new programs with COBOL or PL/I, languages such as Java have become popular for building new applications on distributed platforms.

However, for those of us interested in the traditional languages, there is still widespread development of programs on the mainframe in high-level languages such as COBOL and PL/I. There are many thousands of programs in production on mainframe systems around the world, and these programs are critical to the day-to-day business of the corporations that use them. COBOL and other high-level language programmers are needed to maintain existing code and make updates and modifications to those programs.

Also, many corporations continue to build new application logic in COBOL and other traditional languages, and IBM continues to enhance the high-level

[2] The origin of the term *programming bug* is often attributed to U.S. Navy Lieutenant Grace Murray Hopper in 1945. As the story goes, Lt. Hopper was testing the Mark II Aiken Relay Calculator at Harvard University. One day, a program that worked previously, mysteriously failed. Upon inspection, the operator found that a moth was trapped between the circuit relay points and had created a short circuit (early calculators occupied many square feet, and consisted of tens of thousands of vacuum tubes). The September 9, 1945 log included both the moth and the entry: “First actual case of a bug being found,” and that they had “debugged the machine.”

language compilers to include new functions and features that allow these languages to continue to exploit newer technologies and data formats.

Using Debug Tool for VSE/ESA

The Debug Tool for VSE/ESA, as a separate product, is a powerful source-level debugger to allow z/VSE programmers to debug applications written in high-level languages supported by the LE for z/VSE run-time environment.

While a program is running, programmers can control and examine its execution and isolate problems based on functions such as:

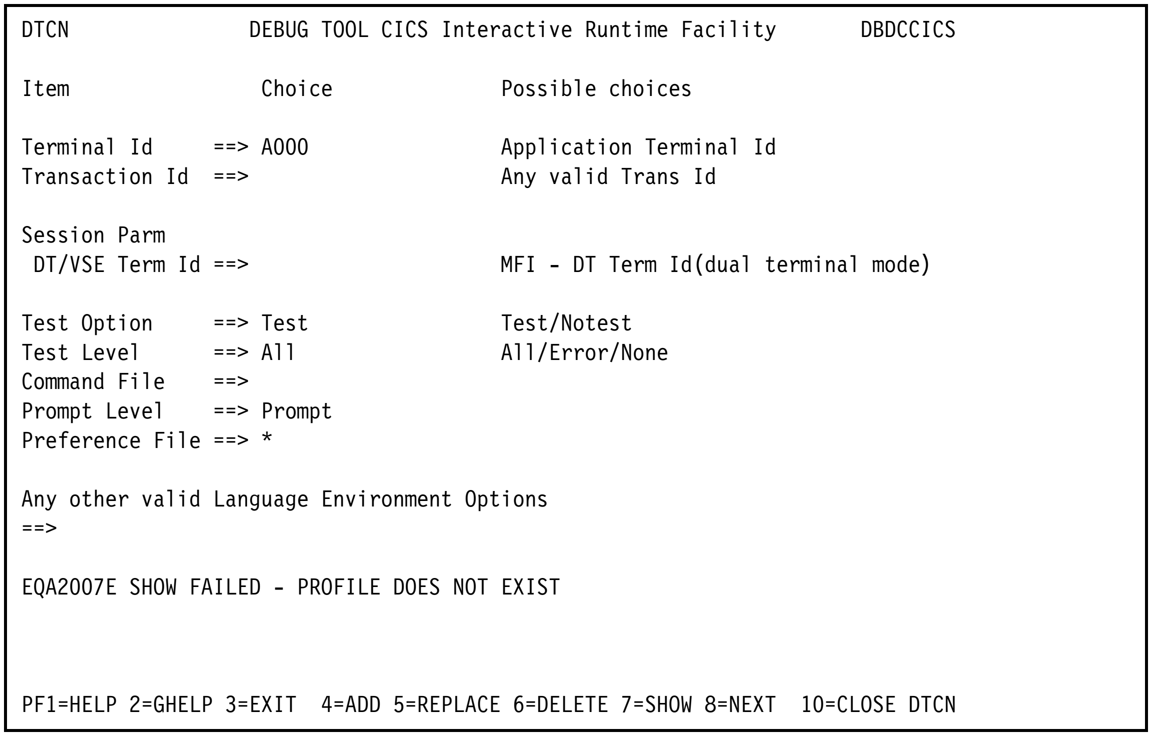
- Viewing the source listing and stepping through the source
- Monitoring the values of program variables
- Modifying program and variable storage
- Setting dynamic break points and enabling conditional-based proceeding
- Debugging mixed-language applications
- Recording a debug session to a log file (for replay purposes)

There are various ways to conduct to debugging sessions or mode. For example, by:

- Using batch commands files. This mechanism is based on predefined command files performed at execution time. Note that neither terminal input nor user interaction is available in this mode.
- Establishing an interactive debugging session to 3270 terminals with full-screen capabilities (batch and CICS).
- Compiling programs with hooks (instructions generated by the compiler to gain control at certain execution points).
- Further choices are the use of LE/VSE[3] callable service “CEETEST” or language-specific functions such as __ctest (in C) or PLITEST built-in (PL/I).

[3] LE/VSE is described in 9.10, “What is Language Environment for z/VSE?” on page 238.

- Adding a customized LE/VSE user exit to the application in order to use supplied debugging transaction “DTCN”(CICS-only). The associated prerequisite is to add “INCLUDE EQADCCXT” at application link-edit time. Subsequently, transaction “DTCN” can then be invoked at a 3270 terminal session (see Figure 8-8).



```
DTCN              DEBUG TOOL CICS Interactive Runtime Facility       DBDCCICS

Item                Choice              Possible choices

Terminal Id     ==> A000                Application Terminal Id
Transaction Id  ==>                     Any valid Trans Id

Session Parm
 DT/VSE Term Id ==>                     MFI - DT Term Id(dual terminal mode)

Test Option     ==> Test                Test/Notest
Test Level      ==> All                 All/Error/None
Command File    ==>
Prompt Level    ==> Prompt
Preference File ==> *

Any other valid Language Environment Options
==>

EQA2007E SHOW FAILED - PROFILE DOES NOT EXIST



PF1=HELP 2=GHELP 3=EXIT  4=ADD 5=REPLACE 6=DELETE 7=SHOW 8=NEXT  10=CLOSE DTCN
```

Figure 8-8 Initial DTCN screen with no user profile

The chosen method may take into consideration debugging capabilities, size of application, and performance aspects.

Debug Tool for VSE/ESA itself is distributed as part of the full function offering of the following IBM high-level language compilers:

- IBM C for VSE/ESA
- IBM COBOL for VSE/ESA
- IBM PL/I for VSE/ESA

Related reading: You can find more information about debugging under z/VSE at:

- Debug Tool for VSE/ESA Home Page

  `http://www-306.ibm.com/software/awdtools/debugtoolvse/`
- *Debug Tool/VSE V1R1 Installation and Customization Guide*, SC26-8798

  `http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/EQAVI000/CCONTENTS`
- *Debug Tool/VSE V1R1 User's Guide and Reference*, SC26-8797

  `http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/EQAVU000/CCONTENTS`

8.4.6 Performing a system test

The difference between the testing done at this stage and the testing that was done during the development phase is that we are now testing the application as a whole, as well as in a running system with other applications. We also carry out tests that can only be done once the application coding has been completed because we need to know how the whole application performs, not just a portion of it.

The tests performed during this phase are:

- User testing - Testing the application for functionality and usability.
- Integration testing - The new application is tested together with other applications to see whether they interface as expected.
- Performance or stress testing - The application is tested using real production data or at least real production data volume to see how well the application performs when there is high demand.

The results of the user and integration tests need to be verified to ensure that they are satisfactory. In addition, the performance of the application must match the requirements. Any issues coming out of these tests need to be addressed before going into production. The number of issues encountered during the testing phase are a good indication of how well we did our design work.

8.5 Going into production on the mainframe

The act of *going into production* is not simply turning on a switch that says now the application is production-ready. It is much more complicated than that. And from one project to the next, the way in which a program goes into production can change. In some cases, where we have an existing system that we are replacing, we might decide to run in parallel for a period of time prior to switching over to the new application. In this case, we run both the old and the new systems against the same data and then compare the results. If after a certain period of time we are satisfied with the results, we switch to the new application. If we discover problems, we can correct them and continue the parallel run until there are no longer any new problems.

In other cases, we deal with a new system, and we might have a cut-over day when we start using it. Even in the case of a new system, we are usually replacing some form of system, even if it is a manual system, so we could still do a parallel test if we wanted to.

Whichever method is used to go into production, there are still all of the loose ends that need to be taken care of before we hand the system over to operations. One of the tasks is to provide documentation for the system, as well as procedures for running and using it. We need to train anyone who interacts with the system.

When all of the documentation and training has been done, then we can hand over responsibility for the running of the application to operations and responsibility for maintaining the application to the maintenance group. In some cases, the development group also maintains applications.

At this point, the application development life cycle reaches a steady state and we enter the maintenance phase of the application. From this point onward, we only apply enhancements and day-to-day changes to the application. Because the application now falls under a change control process, all changes require testing according to the process for change control, before they are accepted into production. In this way, a stable, running application is ensured for end users.

8.6 Summary

This chapter describes the roles of the application designer and application programmer. The discussion is intended to highlight the types of decisions that are involved in designing and developing an application to run in the mainframe environment. This is not to say that the process is much different on other platforms, but some of the questions and conclusions can be different.

This chapter then describes the life cycle of designing and developing an application to run on z/VSE. The process begins with the requirement gathering phase, in which the application designer analyses user requirements to see how best to satisfy them. There might be many ways to arrive at a given solution. The object of the analysis and design phases is to ensure that the optimal solution is chosen. Here, *optimal* does not mean *quickest*, although time is an issue in any project. Instead, optimal refers to the best overall solution, with regard to user requirements and problem analysis.

The EBCDIC character set is different from the ASCII character set. On a character-by-character basis, translation between these two character sets is trivial. When collating sequences are considered, the differences are more significant, and converting programs from one character set to the other can be trivial or it can be quite complex. The EBCDIC character set became an established standard before the current 8-bit ASCII character set had significant use.

At the end of the design phase, the programmer's role takes over. The programmer must now translate the application design into error-free program code. Throughout the development phase, the programmer tests the code as each module is added to the whole. The programmer must correct any logic problems that are detected and add the updated modules to the completed suite of tested programs.

An application rarely exists in isolation. Rather, an application is usually part of a larger set of applications, where the output from one application is the input to the next application. To verify that a new application does not cause problems when incorporated into the larger set of applications, the application programmer conducts a system test or integration test. These tests are themselves designed, and many test results are verified by the actual application users. If any problems are found during system test, the problems will need to be resolved and the test repeated before the process can proceed to the next step.

Following a successful system test, the application is ready to go into production. This phase is sometimes referred to as promoting an application. Once promoted, the application code is now more closely controlled. A business would not want to introduce a change into a working system without being sure of its reliability. At most z/VSE sites, strict rules govern the promotion of applications (or modules within an application) to prevent untested code from contaminating a *pure* system.

At this point in the life cycle of an application, it has reached a steady state. The only changes that will be made to a production application are enhancements, functional changes (for example, tax laws change, so payroll programs need to change), or corrections.

Key terms in this chapter		
Application	ASCII	Database
Design	Develop	EBCDIC
Executable	Platform	Transaction

8.7 Questions for review

To help test your understanding of the material in this chapter, complete the following questions:

1. What are the differences between an application designer and an application programmer? Which role must have a global view of the entire project?
2. In which phase of the application development life cycle does the designer conduct interviews?
3. What is the purpose for using a repository to manage source code?
4. What are the phases in an application development life cycle? State briefly what happens in each phase.
5. If you were a designer on a specific project and the time line for getting the new application into production was very short, what decisions might you make to reduce the overall time line of the project?
6. As part of your system testing phase, you do a performance test on the application. Why would you use production data to do this test?
7. Give some possible reasons for deciding to use batch for an application versus online.
8. Why not store all documents in ASCII format, so they would not have to be converted from EBCDIC?

9

Using programming languages on z/VSE

Objective: As your company's newest z/VSE application programmer, you will need to know which programming languages are supported on z/VSE, and how to determine which is best for a given set of requirements.

After completing this chapter, you will be able to:

- List several common programming languages for the mainframe.
- Explain the differences between a compiled language and an interpreted language.
- Create a simple REXX program.
- Choose an appropriate data file organization for an online application.
- Compare the advantages of a high-level language to those of Assembler language.
- Explain the relationship between a data set name, a DLBL name, and the file name within a program.
- Explain how the use of Language Environment for z/VSE affects the decisions made by the application designer.

9.1 Overview of programming languages

A computer language is the way that a human communicates with a computer. It is needed because a computer works only with its machine language (bits and bytes). This is slow and cumbersome for humans to use. Therefore, we write programs in a computer language, which then gets converted into machine language for the computer to process.

Programming language
The means by which a human communicates with a computer

There are many computer languages, and they have been evolving from machine language into a more natural way of writing. Some languages have been adapted to the kind of application that they intended to solve and to the kind of approach used in the design. The word *generation* has been used to indicate this evolution.

A classification of computer languages follows.

1. Machine language, the first generation, direct machine code.
2. Assembler, second generation, using mnemonics to present the instructions to be translated later into machine language by an assembly program, such as Assembler language.

Generation
Stages in the evolution of computer languages

3. Procedural languages, third generation, also known as high-level languages (HLL), such as Pascal, FORTRAN, Algol, COBOL, PL/I, Basic, and C. The coded program, called a source program, has to be translated through a compilation step.
4. Non-procedural languages, fourth generation, also known as 4GL, used for predefined functions in applications for databases, report generators, queries, such as RPG, CSP, QMF, and Enterprise Generation Language (EGL).
5. Visual Programming languages that use a mouse and icons, such as VisualBasic and VisualC++.
6. HyperText Markup Language, used for writing of World Wide Web documents.
7. Object-oriented language, OO technology, such as Smalltalk, Java, and C++.
8. Other languages, for example, 3D applications.

Each computer language evolved separately, driven by the creation of and adaptation to new standards. In the following sections we describe several of the most widely used computer languages supported by z/VSE:

- Assembler - “Using Assembler language on z/VSE” on page 220
- COBOL - “Using COBOL on z/VSE” on page 222
- PL/I - “Using PL/I on z/VSE” on page 229
- C - “Using C on z/VSE” on page 232
- REXX - “Using REXX on z/VSE” on page 232

For the computer languages under discussion, we have listed their evolution and classified them. There are procedural and non-procedural, compiled and interpreted, and machine-dependent and non-machine-dependent languages.

Assembler language programs are machine-dependent, because the language is a symbolic version of the machine's language on which the program is running. Assembler language instructions can differ from one machine to another, so an Assembler language program written for one machine might not be portable to another. Rather, it would most likely need to be rewritten to use the instruction set of the other machine. A program written in a high-level language (HLL) would run on other platforms, but it would need to be recompiled into the machine language of the target platform.

Most of the HLLs that we touch upon in this chapter are procedural languages. This type is well-suited to writing structured programs. The non-procedural languages, such as SQL and RPG, are more suited for special purposes, such as report generation.

Most HLLs are compiled into machine language, but some are interpreted. Those that are compiled result in machine code, which is very efficient for repeated executions. Interpreted languages must be parsed, interpreted, and executed each time that the program is run. The trade-off for using interpreted languages is a decrease in programmer time, but an increase in machine resources.

The advantages of compiled and interpreted languages are further explored in 9.9, "Compiled versus interpreted languages" on page 237.

9.2 Choosing a programming language for z/VSE

In developing a program to run on z/VSE, your choice of a programming language might be determined by the following considerations:

- What type of application?
- What are the response time requirements?
- What are the budget constraints for development and ongoing support?
- What are the time constraints of the project?
- Do we need to write some of the subroutines in different languages because of the strengths of a particular language versus the overall language of choice?
- Do we use a compiled or an interpreted language?

- Will there be a need or benefit to combine programs written in different languages?

The sections that follow look at considerations for several languages commonly supported on the mainframe.

9.3 Using Assembler language on z/VSE

Assembler
A compiler for Assembler language programs

Assembler language is a symbolic programming language that can be used to code instructions instead of coding in machine language. It is the symbolic programming language that is closest to the machine language in form and content. Therefore, Assembler language is an excellent candidate for writing programs in which:

- You need control of your program, down to the byte or bit level.
- You must write subroutines[1] for functions that are not provided by other symbolic programming languages, such as COBOL or PL/I.

Assembler language is made up of statements that represent either instructions or comments. The instruction statements are the working part of the language, and they are divided into the following three groups:

- A *machine instruction* is the symbolic representation of a machine language instruction of instruction sets, such as:
  - IBM Enterprise Systems Architecture/390 (ESA/390)
  - IBM z/Architecture

  It is called a machine instruction because the Assembler translates it into the machine language code that the computer can execute.
- An *Assembler instruction* is a request to the Assembler to do certain operations during the assembly of a source module (for example, defining data constants, reserving storage areas, and defining the end of the source module).
- A macro instruction or macro is a request to the Assembler program to process a predefined sequence of instructions called a macro definition. From this definition, the Assembler generates machine and Assembler instructions, which it then processes as if they were part of the original input in the source module.

Compiler
Software that converts a set of high-level language statements into a lower-level representation

The Assembler produces a program listing containing information that was generated during the various phases of the assembly process[2]. It is really a compiler for Assembler language programs.

[1] Subroutines are programs that are invoked frequently by other programs, and by definition should be written with performance in mind. Assembler language is a good choice for a subroutine.

Linkage editor
Binds (link-edits) object decks into load modules

The Assembler also produces information for other processors, such as a *binder* (*linker*, or *linkage editor*, as called in VSE). This output is also referred to as object code. Before the computer can execute the program, the object code (called an object deck or simply OBJ) has to be run through another process to resolve the addresses where instructions and data will be located. This process is called *linkage-editing* (or *link-editing* for short) and is performed by the linkage editor.

Load module
Produced by the linkage editor from object modules. is ready to be loaded and run.

The linkage editor (see 10.3.4, “How is a linkage editor used?” on page 257) uses information in the object decks to combine them into load modules. At program fetch time, the load module produced by the linkage editor is loaded into virtual storage. After the program is loaded, it can be run. Figure 9-1 illustrates these steps.

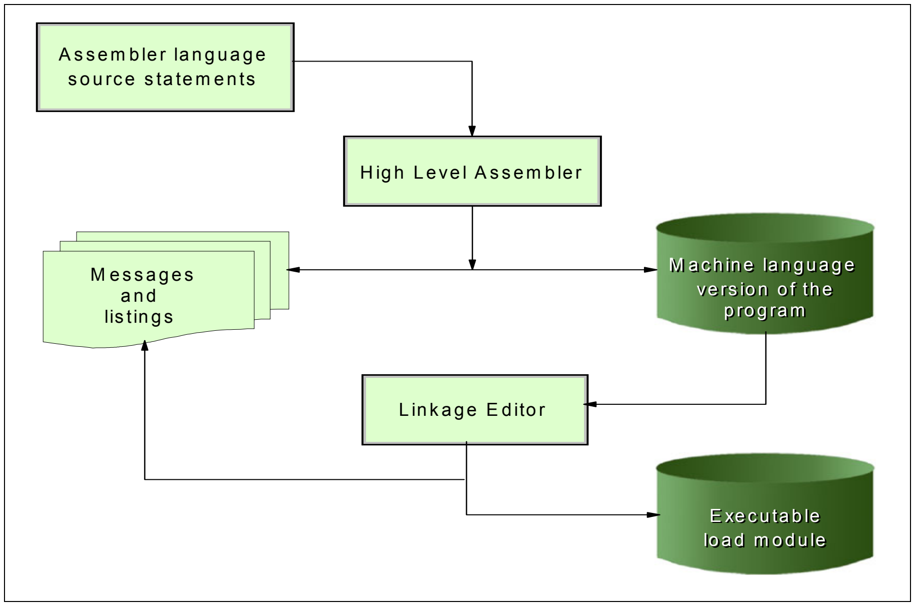


Figure 9-1 Assembler source to executable module

[2] A program listing does not contain *all* of the information that is generated during the assembly process. To capture all of the information that could possibly be in the listing (and more), the z/VSE programmer can specify an Assembler option called ADATA to have the Assembler produce a SYSADAT file as output. The SYSADAT file is not human-readable—its contents are in a form that is designed for a tool to process. The use of a SYSADAT file is simpler for tools to process than the older custom of extracting similar data through *listing scrapers*.

Related reading: You can find more information about using Assembler language on z/VSE in the IBM publications *HLASM General Information,* GC26-4943, and *HLASM Language Reference*, SC26-4940. These books are available on the Web at:

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

9.4 Using COBOL on z/VSE

Common Business-Oriented Language (COBOL) is a programming language similar to English that is widely used to develop business-oriented applications in the area of commercial data processing. COBOL has been almost a generic term for computer programming in this kind of computer language. However, as used in this chapter, COBOL refers to the product IBM COBOL for VSE/ESA.

The COBOL compiler produces a program listing containing all of the information that it generated during the compilation. The compiler also produces information for other processors, such as the object deck for the linkage editor.

Before the computer can execute your program, the object deck has to be run through another process to resolve the addresses where instructions and data will be located. This process is called linkage edition and is performed by the linkage editor.

The linkage editor uses information in the object decks to combine them into load modules. These are further discussed in 10.3.4, “How is a linkage editor used?” on page 257. At program fetch time, the load module produced by the linkage editor is loaded into virtual storage. When the program is loaded, it can then be run. Figure 9-2 illustrates the process of translating the COBOL source language statements into an executable load module. This process is similar to that of Assembler language programs. In fact, this same process is used for all of the HLLs that are compiled.

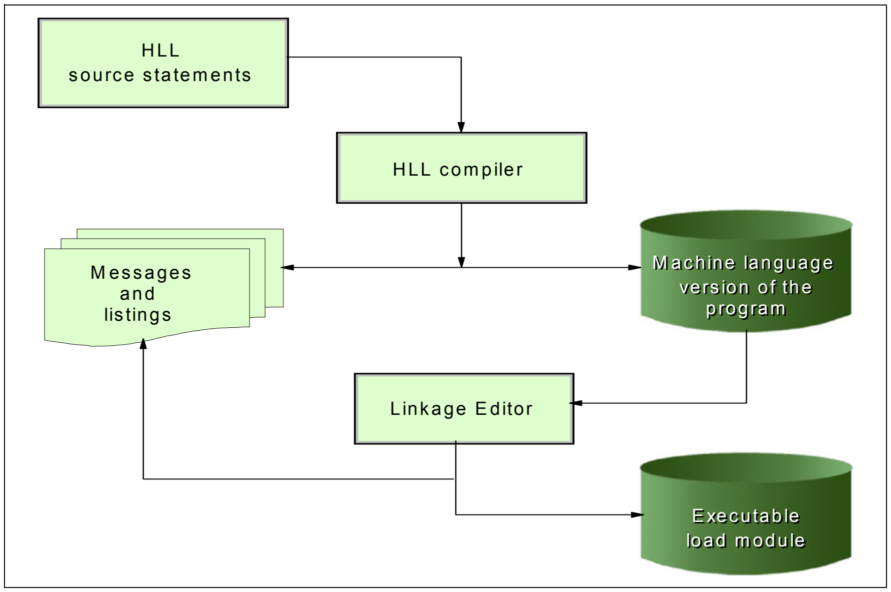


Figure 9-2 HLL source to executable module

9.4.1 COBOL program format

With the exception of the COPY and REPLACE statements and the end program marker, the statements, entries, paragraphs, and sections of a COBOL source program are grouped into the following four divisions:

- IDENTIFICATION DIVISION, which identifies the program with a name and, if you want, gives other identifying information.
- ENVIRONMENT DIVISION, where you describe the aspects of your program that depend on the computing environment.
- DATA DIVISION, where the characteristics of your data are defined. These are defined in one of the following sections in the DATA DIVISION:
  - FILE SECTION, to define data used in input-output operations
  - LINKAGE SECTION, to describe data from another program

When defining data developed for internal processing:

  - WORKING-STORAGE SECTION, to have storage statically allocated and remain for the life of the run unit
  - LINKAGE SECTION, to describe data from another program

- PROCEDURE DIVISION, where the instructions related to the manipulation of data and interfaces with other procedures are specified. The PROCEDURE DIVISION of a program is divided into sections and paragraphs, which contain sentences and statements, as described here:
  - Section - a logical subdivision of your processing logic. A section has a section header and is optionally followed by one or more paragraphs. A section can be the subject of a PERFORM statement. One type of section is for declaratives, representing a set of one or more special purpose sections, written at the beginning of the Procedure Division, the first of which is preceded by the key word DECLARATIVES, and the last of which is followed by the key word END DECLARATIVES.
  - Paragraph - a subdivision of a section, procedure, or program. A paragraph can be the subject of a statement.
  - Sentence - a series of one or more COBOL statements ending with a period.
  - Statement - performs a defined step of COBOL processing, such as adding two numbers.
  - Phrase - a subdivision of a statement.

Examples of COBOL divisions

Example 9-1 and Example 9-2 are examples of different divisions in a COBOL program.

Example 9-1 IDENTIFICATION DIVISION

```
IDENTIFICATION DIVISION.
Program-ID. Helloprog.
Author. A. Programmer.
Installation.  Computing Laboratories.
Date-Written.  08/21/2002.
```

Example 9-2 ENVIRONMENT DIVISION

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. computer-name.
OBJECT-COMPUTER. computer-name.
SPECIAL-NAMES.
    special names entries
```

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT [OPTIONAL] file-name-1
          ASSGN TO system-name.
I-O-CONTROL.
    SAME [RECORD] AREA FOR file-name-1 ... file-name-n.
```

Example of input-output coding

Explanations of the user-supplied information follow input and output files in FILE-CONTROL. See Example 9-3.

Example 9-3 Input and output files in FILE-CONTROL

```
IDENTIFICATION DIVISION.
 . . .
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
       SELECT filename ASSIGN TO assignment-name
       ORGANIZATION IS org ACCESS MODE IS access
FILE STATUS IS file-status
. . .
DATA DIVISION.
FILE SECTION.
FD  filename
01  recordname
    nn . . . fieldlength & type
    nn . . . fieldlength & type
WORKING-STORAGE SECTION
01 file-status  PICTURE 99.
. . .
PROCEDURE DIVISION.
    OPEN iomode filename
    . . .
    READ filename
    . . .
    WRITE recordname
    . . .
    CLOSE filename
    . . .
    STOP RUN.
```

org indicates the organization, which can be SEQUENTIAL, INDEXED, or RELATIVE.

access indicates the access mode, which can be SEQUENTIAL, RANDOM, or DYNAMIC.

iomode is for INPUT or OUTPUT mode. If you are only reading from a file, code INPUT. If you are only writing to it, code OUTPUT or EXTEND. If you are both reading and writing, code I-O.

Others like *filename*, *recordname*, *fieldname* (nn in the example), *fieldlength*, and *type* are also specified.

9.4.2 COBOL relationship between JCL and program files

Example 9-4 depicts the relationship between JCL statements and the files in a COBOL program. By not referring to physical locations of data files in a program, we achieve device independence. That is, we can change where the data resides and what it is called without having to change the program. We would only need to change the JCL.

Example 9-4 COBOL relationship between JCL and program files

```
// JOB MYJOB
*  STEP1 COMPILE AND LINK
// EXEC IGYCRCTL
...
   INPUT-OUTPUT SECTION.
   FILE-CONTROL.
     SELECT INPUT ASSIGN TO INPUT1 .....
     SELECT DISKOUT ASSIGN TO OUTPUT1 ...
   FILE SECTION.
     FD INPUT1
        BLOCK CONTAINS...
        DATA RECORD IS RECORD-IN
     01 INPUT-RECORD
...
     FD OUTPUT1
        DATA RECORD IS RECOUT
     01 OUTPUT-RECORD
...
/*
// EXEC LNKEDT ...
/*
*  STEP2 RUN PROGRAM
// DLBL INPUT1,'MY.INPUT',...
// DLBL OUTPUT1,'MY.OUTPUT'...
// EXEC PROG...
...
```

Example 9-4 shows a COBOL compile, link, and go job stream, listing the file program statements and the JCL statements to which they refer.

The COBOL SELECT statements make the links between the DLBL names INPUT1 and OUTPUT1, and the COBOL FDs INPUT1 and OUTPUT1, respectively. The COBOL FDs are associated with group items INPUT-RECORD and OUTPUT-RECORD.

The DLBL cards INPUT1 and OUTPUT1 are related to the data sets MY.INPUT and MY.OUTPUT, respectively. The end result of this linkage in our example is that records read from the file INPUT1 will be read from the physical data set MY.INPUT and records written to the file OUTPUT1 will be written to the physical data set MY.OUTPUT. The program is completely independent of the location of the data and the name of the data sets.

Figure 9-3 shows the relationship between a COBOL program accessing a physical data set and the corresponding VSE job control (JCL).

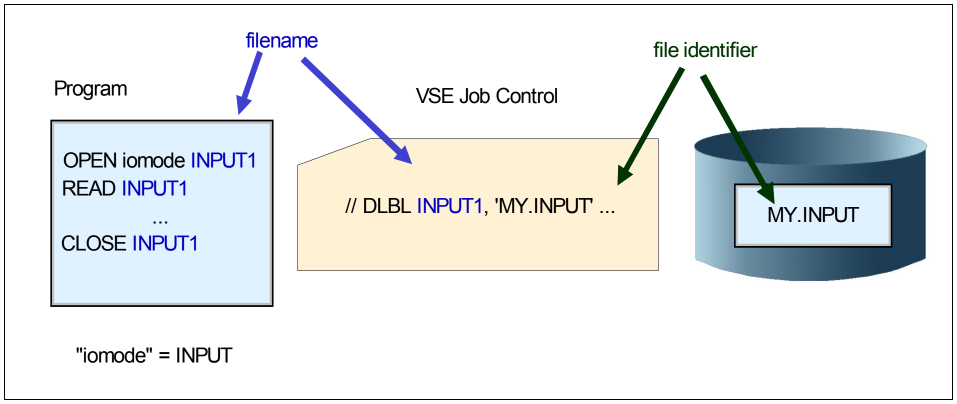


Figure 9-3 Relationship between JCL, program, and data set

Again, because the program does not make any reference to the physical data set, we would not need to recompile the program if the name of the data set or its location were to change.

9.5 HLL relationship between JCL and program files

In the previous section we learned how to isolate a COBOL program from changes in data set name and data set location. The technique of referring to physical files by a symbolic file name is not restricted to COBOL. It is used by all HLLs and even in Assembler language. Example 9-5 shows a generic HLL example of a program that references data sets through symbolic file names.

Example 9-5 HLL relationship between JCL and program files

```
// JOB MYJOB
// STEP1 COMPILE AND LINK
// EXEC IGYCRCTL
...
   OPEN FILE=INPUT1
   OPEN FILE=OUTPUT1
   READ FILE=INPUT1
...
   WRITE FILE=OUTPUT1
...
   CLOSE FILE=INPUT1
   CLOSE FILE=OUTPUT1
/*
// EXEC LNKEDT ...
/*
*  STEP2 RUN PROGRAM
// DLBL INPUT1,'MY.INPUT',...
// DLBL OUTPUT1,'MY.OUTPUT'...
// EXEC PROG...
...
```

Isolating your program from changes to data set name and location is the normal objective. However, there could be cases when a program needs to access a specific data set at a specific location on a direct access storage device (DASD). This can be accomplished in Assembler language and even in some HLLs.

The practice of *hard-coding* data set names or other such information in a program is not usually considered a good programming practice. Values that are hard-coded in a program are subject to change and would therefore require that the program be recompiled each time a value changed. Externalizing these values from programs, as with the case of referring to data sets within a program by a symbolic name, is a more effective practice that allows the program to continue working even if the data set name changes.

For a more detailed explanation of using a symbolic name to refer to a file, see “Why z/VSE uses symbolic file names” on page 165.

9.6 Using PL/I on z/VSE

Programming Language/I (PL/I, pronounced *P-L one*) is a full function, general-purpose, high-level programming language suitable for the development of:

- Commercial applications
- Engineering/scientific applications
- Many other applications

The process of compiling a PL/I source program and then link-editing the object deck into a load module is basically the same as it is for COBOL. See Figure 9-2 on page 223; 10.3.4, “How is a linkage editor used?” on page 257; and Figure 9-3 on page 227.

The relationship between JCL and program files is the same for PL/I as it is for COBOL and other HLLs. See Figure 9-3 on page 227 and Example 9-5 on page 228.

9.6.1 PL/I program structure

PL/I is a block-structured language consisting of procedures, begin blocks, statements, expressions, and built-in functions, as shown in Figure 9-4.

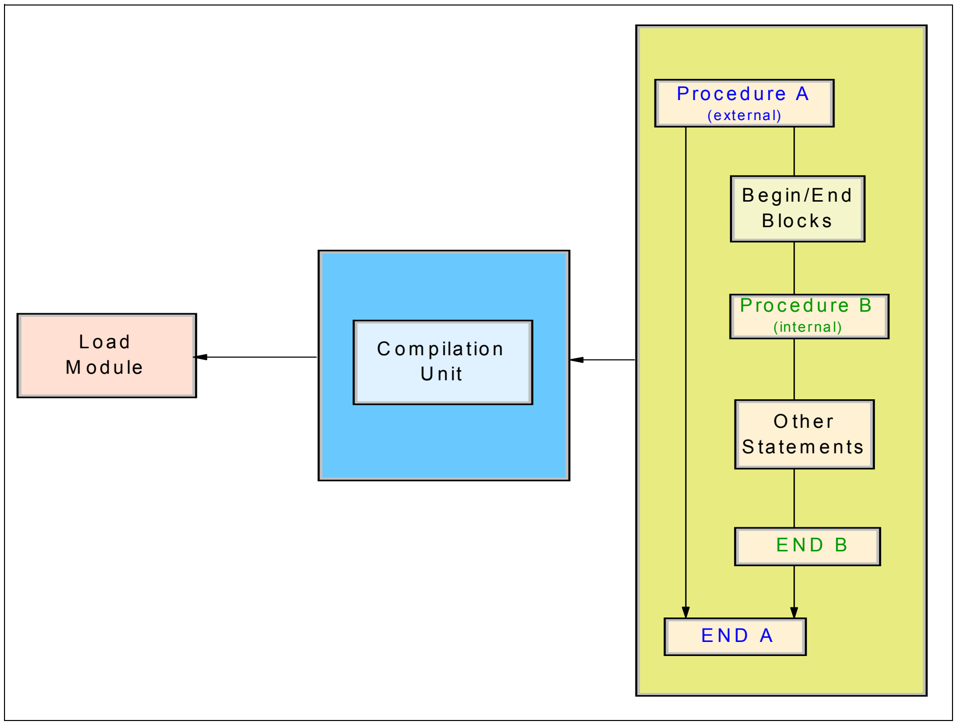


Figure 9-4 PL/I application structure

Variable
Holds data assigned to it until a new value is assigned

PL/I programs are made up of blocks. A *block* can be either a subroutine or just a group of statements. A PL/I block allows you to produce highly modular applications, because blocks can contain declarations that define variable names and storage classes. Thus, you can restrict the scope of a variable to a single block or a group of blocks, or you can make it known throughout the compilation unit or a load module.

A PL/I application consists of one or more separately loadable entities, known as a *load modules*. Each load module can consist of one or more separately compiled entities, known as *compilation units*. Unless otherwise stated, a program refers to a PL/I application or a compilation unit.

A compilation unit is an external procedure. A PL/I external or internal procedure contains zero or more blocks.

A PL/I block is either a PROCEDURE or a BEGIN block, any of which contains zero or more statements or zero or more blocks.

A procedure is a sequence of statements delimited by a PROCEDURE statement and a corresponding END statement, as shown in Example 9-6. A procedure can be a main procedure, a subroutine, or a function. An application must have exactly one external procedure that has OPTIONS(MAIN). It may have further internal PROCEDUREs.

Example 9-6 A PROCEDURE block

```
A: procedure_name;
      statement-1
      statement-2
      .
      .
      .
      statement-n
      end A;
```

A BEGIN block is a sequence of statements delimited by a BEGIN statement and a corresponding END statement, as shown in Example 9-7. A program is terminated when the main procedure is terminated.

Example 9-7 BEGIN block

```
B:  begin;
     statement-1
     statement-2
     .
     .
     statement-n
     end B;
```

9.6.2 Preprocessor

The PL/I compiler allows you to select an integrated preprocessors for use in your program. A number of options are available to tailor processing to your needs.

Macro preprocessor

Macros allow you to write commonly used PL/I code in a way that hides implementation details and the data that is manipulated, and exposes only the operations. In contrast with a generalized subroutine, macros allow generation of only the code that is needed for each individual use.

Related reading: For more information about using PL/I on z/VSE see the IBM publications *PL/I VSE Language Reference*, SC26-8054, and *PL/I VSE Programming Guide*, SC26-8053. These books are available on the Web at:

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

9.7 Using C on z/VSE

C is a programming language designed for a wide variety of programming purposes, including:

- System-level code
- Text processing
- Graphics

The C language contains a concise set of statements with functionality added through its library. This division enables C to be both flexible and efficient. An additional benefit is that the language is highly consistent across different systems.

The process of compiling a C source program and then link-editing the object deck into a load module is basically the same as it is for COBOL. See Figure 9-2 on page 223; 10.3.4, "How is a linkage editor used?" on page 257; and Figure 9-3 on page 227 to see this process.

The relationship between JCL and program files is the same for PL/I as it is for COBOL and other HLLs. See Figure 9-3 on page 227 and Example 9-5 on page 228.

Related reading: For more information about using C on z/VSE see the IBM publications, *C/VSE Language Reference*, SC09-2425, and *C/VSE Users Guide*, SC09-2424. These books are available on the Web at:

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

9.8 Using REXX on z/VSE

The Restructured Extended Executor (REXX) language is a procedural language that allows programs and algorithms to be written in a clear and structural way. It is an interpreted and compiled language. An interpreted language is different from other programming languages, such as COBOL, because it is not necessary to compile a REXX command list before executing it. On z/VSE there is no REXX/VSE Compiler available. However, a REXX program compiled under either z/OS or z/VM can be executed on z/VSE.

The REXX language is implemented in z/VSE through:

- The REXX/VSE interpreter
- The Library for REXX/370 in REXX/VSE

The interpreter is also called the language processor. The Library for REXX/370 in REXX/VSE is also called a compiler's runtime processor.

REXX/VSE is a partial implementation of Level 2 Systems Application Architecture (SAA) REXX on the z/VSE system. The purpose of SAA REXXX is to define a consistent set of language elements that can be used on several operating systems.

The REXX/VSE programming language is typically used for:

- Performing routine tasks, such as entering z/VSE commands
- Invoking other REXX execs
- Invoking applications written in other languages (application front ends)
- Prototyping
- One-time quick solutions to problems
- System programming, personal computing
- Wherever we can use another HLL compiled language

The structure of a REXX program is simple. It provides a conventional selection of control constructs. For example, these include IF... THEN... ELSE... for simple conditional processing, SELECT... WHEN... OTHERWISE... END for selecting from a number of alternatives, and several varieties of DO... END for grouping and repetitions, and a number of useful built-in-functions. No GOTO instruction is included, but a SIGNAL instruction is provided for abnormal transfer of control such as error exits and computed branching.

Environment

The system under which REXX programs run is assumed to include at least one command environment for processing commands. An environment is selected by default on entry to a REXX program. The environment for processing host commands is known as the host command environment. The underlying operating system defines environments external to the REXX program.

A host command (z/VSE command) is a command for the surrounding system to act upon. Issuing host commands from within a REXX program is an integral part of the REXX language.

REXX/VSE commands and ADDRESS POWER commands can be used in a REXX program. You can also link to programs and issue JCL commands. 9.8.2, "Host commands and host command environments" on page 235 describes the different environments for using host services.

The relationship between JCL and program files is the same for REXX as it is for COBOL and other HLLs. See Figure 9-3 on page 227 and Example 9-5 on page 228.

9.8.1 Compiling and executing REXX command lists

A REXX program compiled under z/OS can run under z/VM. Similarly, a REXX program compiled under z/VM can run under z/OS. A REXX program compiled under z/OS or z/VM can run under z/VSE if REXX/VSE is installed. REXX programs cannot be compiled under z/VSE.

The process of compiling a REXX source program and then link-editing the object deck into a load module is basically the same as it is for COBOL. See Figure 9-2 on page 223; 10.3.4, “How is a linkage editor used?” on page 257; and Figure 9-3 on page 227 to see this process.

There are three main components of the REXX language when using a compiler:

- IBM Compiler for REXX on System z

  The compiler translates REXX source programs into compiled programs (no compiler for REXX/VSE).

- IBM Library for REXX on System z

  The library contains routines that are called by compiled programs at runtime.

- Alternate Library

  The Alternate Library contains a language processor that transforms the compiled programs and runs them with the interpreter. It can be used by z/OS and z/VM users who do not have the IBM Library for REXX on System z to run compiled programs.

The compiler and library run on z/OS systems with TSO/E, and under CMS on z/VM systems. The IBM Library for REXX in REXX/VSE runs under z/VSE.

The compiler can produce output in the following forms:

- Compiled EXECs

  These behave exactly like interpreted REXX programs. They are invoked the same way by the system's EXEC handler, and the search sequence is the same. The easiest way of replacing interpreted programs with compiled programs is by producing compiled EXECs. Users need not know whether the REXX programs they use are compiled EXECs or interpretable programs. Compiled EXECs can be sent to z/VSE to be run there.

- Object decks under z/OS or TEXT files under z/VM

  A TEXT file is an object code file whose external references have not been resolved (this term is used on z/VM only). These must be transformed into executable form (load modules) before they can be used. Load modules and MODULE files are invoked the same way as load modules derived from other compilers, and the same search sequence applies. However, the search sequence is different from that of interpreted REXX programs and compiled EXECs. These load modules can be used as commands and as parts of REXX function packages. Object decks or MODULE files can be sent to z/VSE to build phases.

- IEXEC output

  This output contains the expanded source of the REXX program being compiled. Expanded means that the main program and all the parts included at compilation time by means of the %INCLUDE directive are contained in the IEXEC output. Only the text within the specified margins is contained in the IEXEC output. Note, however, that the default setting of MARGINS includes the entire text in the input records.

9.8.2 Host commands and host command environments

A host command is a command for the surrounding environment to act upon. Host commands can be issued from a REXX program. When the language processor processes a clause that it does not recognize as an assignment or other REXX instruction, the language processor treats the clause as a host command and routes the command to the host command environment. The host command environment processes the command and then returns control to the language processor.

REXX/VSE provides six host command environments:

- VSE
- POWER
- JCL
- LINK
- LINKPGM
- CONSOLE

The VSE host command environment

The default host command environment is VSE.

The VSE host command environment is used to invoke REXX/VSE commands and services. Another REXX program can be called by using the EXEC command. In the VSE environment, all REXX/VSE commands can be used, but you cannot use POWER, JCL, or Console commands. The following example assumes that the current host command environment is not VSE.:

```
ADDRESS VSE “EXEC programname  p1  p2  …”
ADDRESS VSE “programname  p1  p2 …”     (implicit EXEC Command)
```

The POWER host command environment

The POWER host command environment is for VSE/POWER spool-access services requests GET, PUT, and CTL. In the POWER host command environment both REXX/VSE and POWER commands can be used. The POWER host command environment lets you:

- Use the PUTQE command to put elements on a POWER queue and the GETQE command to retrieve POWER queue elements.
- Send a CTL service request to POWER. Use ADDRESS POWER to send commands like PALTER, PDISPLAY, PSTART, PSTOP, and so on.
- Use the QUERYMSG command to return job completion messages into the stem specified by OUTTRAP.
- Execute REXX/VSE commands.

The JCL host command environment

Use the JCL host command environment to issue a VSE JCL command in a much simpler way than with the conditional job control language. The host command environment is invoked via the command ADDRESS JCL.

The LINK and LINKPGM host command environments

Loading and calling a program is called linking. REXX/VSE provides the LINK and LINKPGM host command environments to let you load and call non-REXX programs. These programs must be phases from the active VSE PHASE search chain. LINK and LINKPGM offer different ways to provide parameters.

The LINK environment offers an alternative to the job control statement EXEC PGM. The parameter list is the same as if you specify the PARM parameter in the EXEC PGM Statement. For the LINK environment, you can specify only a single character string to pass to the program. The LINK environment does not evaluate the character string and does not perform variable substitution. It simply passes the string to the called program. The program can use the character string it receives. However, the program cannot return an updated string to the REXX program.

When you use the LINK environment, enclose the name of the program and the character string in single or double quotation marks. For example:

```
ADDRESS LINK “TESTPGMA varid”
ADDRESS LINK “TESTMODA this is a parameter string”
```

For the LINKPGM environment, you can pass multiple parameters to the called program. The environment performs variable substitution on the parameter you specify. That is, the environment determines the value of each variable. When the environment calls a program, it passes the value of each variable to the program. The program can update the parameters it receives and return the updated values to the REXX program.

To load and call a program, specify the name of the program, followed by any parameters you want to pass to the program. For example:

```
ADDRESS LINKPGM “program p1 p2 ….pn”
```

The CONSOLE host command environment

The Console host command environment calls to activate one or more z/VSE console sessions. Having activated a z/VSE console session, z/VSE console commands can be imbedded into a REXX program. A GETMSG function receives command responses and console messages.

Related reading: You can find more information about REXX in the following publications:

- *Procedures Language Reference (Level 1)*, C26-4358 SAA CPI
- *REXX/VSE Reference V6R70,* SC33-6642
- *REXX/VSE V6R10 User’s Guide,* SC33-6641
- *The REXX Language*, 2nd Ed., Cowlishaw, ZB35-5100

Also, visit the following Web sites:

```
http://www.ibm.com/software/awdtools/REXX/language/REXXlinks.html
http://www.ibm.com/servers/eserver/zseries/zvse/support/rexx.html
```

9.9 Compiled versus interpreted languages

During the design of an application, you might need to decide between using a compiled language or an interpreted language for the application source code. Both types of languages have their strengths and weaknesses. Usually, the decision to use an interpreted language is based on time restrictions on development or for ease of future changes to the program. A trade-off is made when using an interpreted language. You trade speed of development for higher execution costs. Because each line of an interpreted program must be translated each time it is executed, there is a higher overhead. Thus, an interpreted language is generally more suited to ad hoc requests versus non-ad hoc.

9.9.1 Advantages of compiled languages

Assembler, COBOL, PL/I, and C are all translated by running the source code through a compiler. This results in very efficient code that can be executed any number of times. The overhead for the translation is incurred just once, when the source is compiled. Thereafter, it need only be loaded and executed.

Interpreted languages, in contrast, must be parsed, interpreted, and executed each time the program is run, thereby greatly adding to the cost of running the program. For this reason, interpreted programs are usually less efficient than compiled programs.

Some programming languages, such as REXX, can be either interpreted or compiled.

9.9.2 Advantages of interpreted languages

In the previous section we discussed the reasons for using languages that are compiled. In 9.8, "Using REXX on z/VSE" on page 232 we discussed the strong points of interpreted languages. There is no simple answer as to which language is *better*—it depends on the application. Even within an application we could end up using many different languages. For example, one of the strengths of a language like CLIST is that it is easy to code, test, and change. However, it is not very efficient. The trade-off is machine resources for programmer time.

Keeping this in mind, we can see that it would make sense to use a compiled language for the intensive parts of an application (heavy resource usage), whereas interfaces (invoking the application) and less-intensive parts could be written in an interpreted language. An interpreted language might also be suited for ad hoc requests or even for prototyping an application.

One of the jobs of a designer is to weigh the strengths and weaknesses of each language and then decide which part of an application is best suited for a particular language.

9.10 What is Language Environment for z/VSE?

As we mentioned in Chapter 8, "Designing and developing applications for z/VSE" on page 195, an application is a collection of one or more programs cooperating to achieve particular objectives, such as inventory control or payroll. The goals of application development include modularizing and sharing code, and developing applications on a workstation-based front end.

On z/VSE, the Language Environment for z/VSE (LE/VSE) product provides a common environment for all conforming high-level language (HLL) products. An HLL is a programming language above the level of Assembler language and below that of program generators and query languages. Language Environment for z/VSE establishes a common language development and execution environment for application programmers on z/VSE. Whereas functions were previously provided in individual language products, Language Environment eliminates the need to maintain separate language libraries.

In the past, programming languages had limited ability to call each other and behave consistently across different operating systems. These characteristic constrained programs wanted to use several languages in an application. Programming languages had different rules for implementing data structures and condition handling, and for interfacing with system services and library routines.

With Language Environment and its ability to call one language from another, z/VSE application programmers can exploit the functions and features in each language.

9.10.1 How Language Environment is used

Language Environment establishes a common run-time environment for all participating HLLs. It combines essential run-time services, such as routines for run-time message handling, condition handling, and storage management. These services are available through a set of interfaces that are consistent across programming languages. The application program can either call these interfaces directly or use language-specific services that call the interfaces.

With Language Environment, you can use one run-time environment for your applications, regardless of the application's programming language or system resource needs.

Figure 9-5 shows the components in the Language Environment for z/VSE, including:

- Basic routines that support starting and stopping programs, allocating storage, communicating with programs written in different languages, and indicating and handling conditions.
- Common library services, such as math or date and time services, that are commonly needed by programs running on the system. These functions are supported through a library of callable services.
- Language-specific portions of the run-time library.

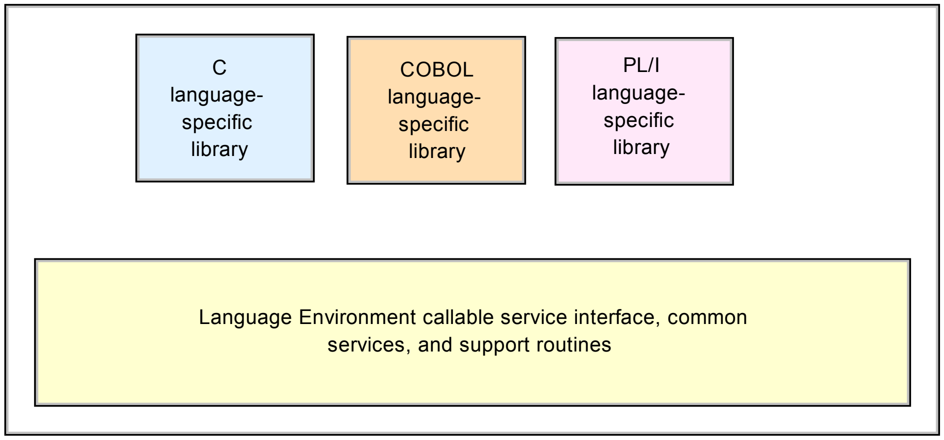

C language-specific library
COBOL language-specific library
PL/I language-specific library
Language Environment callable service interface, common services, and support routines

Figure 9-5 Language Environment components for z/VSE

Language Environment is the prerequisite run-time environment for applications generated with the following IBM compiler products:

- C for VSE/ESA
- COBOL for VSE/ESA
- PL/I for VSE/ESA

In many cases, you can run compiled code generated from the previous versions of the above compilers. A set of Assembler macros is also provided to allow Assembler routines to run with Language Environment.

9.10.2 A closer look at Language Environment

The language-specific portions of Language Environment provide language interfaces and specific services that are supported for each individual language, and that can be called through a common callable interface. In this section we discuss some of these interfaces and services in more detail.

Figure 9-6 shows a common run-time environment established through Language Environment.

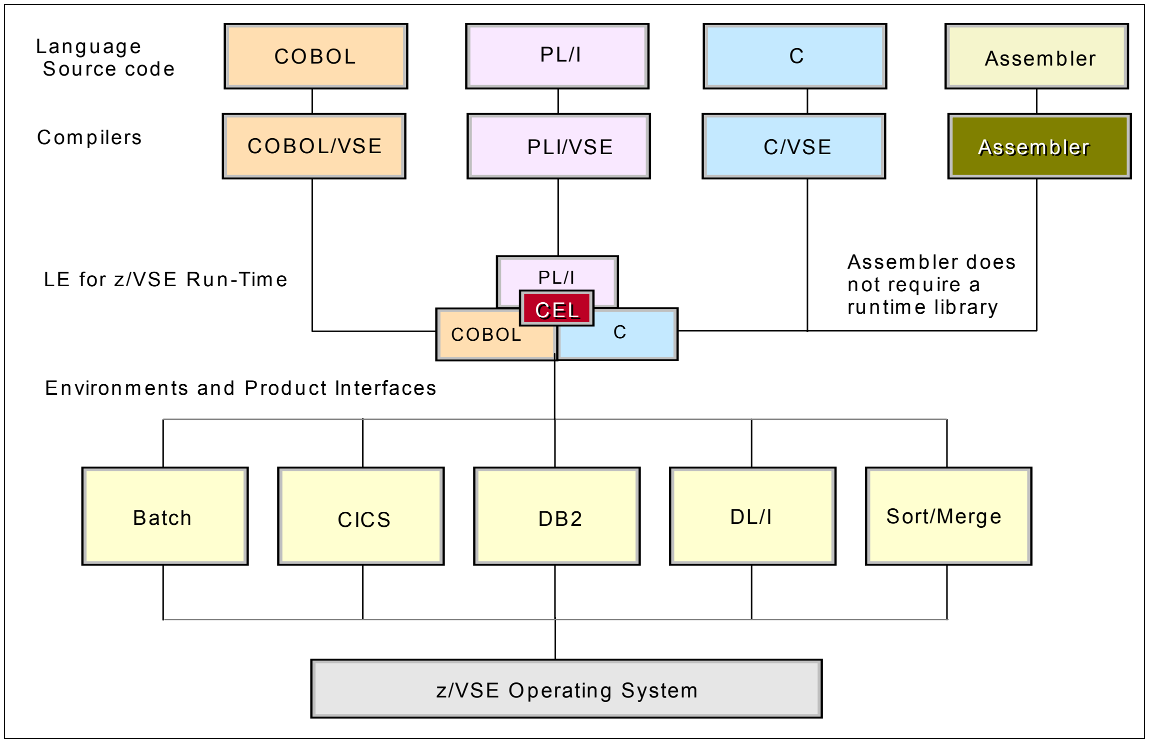


Figure 9-6 Language Environment's common run-time environment

The Language Environment architecture is built from models for the following:

- Program management
- Condition handling
- Message services
- Storage management

Program management model

The Language Environment program management model provides a framework within which an application runs. It is the foundation of all of the component models (condition handling, run-time message handling, and storage management) that comprise the Language Environment architecture.

The program management model defines the effects of programming language semantics in mixed-language applications, and integrates transaction processing.

Some terms used to describe the program management model are common programming terms. Other terms are described differently in other languages. It is important that you understand the meaning of the terminology in a Language Environment context as compared to other contexts.

Program management

Program management defines the program execution constructs of an application, and the semantics associated with the integration of various components' management of such constructs.

Three entities, *process*, *enclave*, and *thread*, are at the core of the Language Environment program management model.

Processes

The highest level component of the Language Environment program model is the process. A process consists of at least one enclave and is logically separate from other processes. Language Environment generally does not allow language file sharing across enclaves, nor does it provide the ability to access collections of externally stored data.

Enclaves

A key feature of the program management model is the enclave, a collection of the routines that make up an application. The enclave is the equivalent of any of the following:

- A run unit, in COBOL
- A program, consisting of a main C function and its sub-functions, in C
- A main procedure and all of its subroutines, in PL/I

In Language Environment, the term *environment* is usually a reference to the run-time environment of HLLs at the enclave level. The enclave consists of one main routine and zero or more subroutines. The main routine is the first to execute in an enclave. All subsequent routines are named as subroutines.

Threads

Each enclave consists of at least one *thread*, the basic instance of a particular routine. A thread is created during enclave initialization with its own run-time stack, which keeps track of the thread's execution, as well as a unique instruction counter, registers, and condition-handling mechanisms. Each thread represents an independent instance of a routine running under an enclave's resources.

Condition-handling model

For single-language and mixed-language applications, the Language Environment run-time library provides a consistent and predictable condition-handling facility. It does not replace current HLL condition handling, but instead allows each language to respond to its own unique environment as well as to a mixed-language environment.

Language Environment condition management gives you the flexibility to respond directly to conditions by providing callable services to signal conditions and to interrogate information about those conditions. It also provides functions for error diagnosis, reporting, and recovery.

Message-handling model and national language support

A set of common message handling services that create and send run-time informational and diagnostic messages is provided by Language Environment.

With the message handling services, you can use the condition token that is returned from a callable service or from some other signaled condition, format it into a message, and deliver it to a defined output device or to a buffer.

National language support callable services allow you to set a national language that affects the language of the error messages and the names of the day, week, and month. It also allows you to change the country setting, which affects the default date format, time format, currency symbol, decimal separator character, and thousands separator.

Storage management model

Common storage management services are provided for all Language Environment-conforming programming languages. Language Environment controls stack and heap storage used at run time. It allows single-language and mixed-language applications to access a central set of storage management facilities, and offers a multiple-heap storage model to languages that do not now provide one. The common storage model removes the need for each language to maintain a unique storage manager, and avoids the incompatibilities between different storage mechanisms.

9.10.3 Running your program with Language Environment

After compiling your program you can do the following:

- Link-edit and run an existing object deck and accept the default Language Environment run-time options.
  - Batch default run-time options are supplied via member CEEDOPT.A (skeleton CEEWDOPT in ICCF library 62).
  - LE/CICS default run-time options are supplied via member CEECOPT.A (skeleton CEEWCOPT in ICCF library 62).

- Link-edit and run an existing object deck and specify new Language Environment run-time options.

  Supplied member CEEUOPT.A can suit as a base for tailoring application specific run-time option overrides (skeleton CEEWUOPT in ICCF library 62).

  Note: If you plan to introduce application-specific run-time option overrides, we recommend carefully checking applicability for the target environment. Only those run-time options that need to be overridden should be coded in a CEEUOPT.OBJ generation. This keeps the object size small and avoids potential overrides that do not match with the target environment.

- Call a Language Environment service.

Run-time library services

The Language Environment shipment library PRD2.SCEEBASE contains the run-time library routines needed during execution of applications written in C, PL/I, and COBOL

Important: Language Environment library routines are divided into two categories: resident routines and dynamic routines. The resident routines are linked with the application and include such things as initialization/termination routines and pointers to callable services. The dynamic routines are not part of the application and are dynamically loaded during run time.

There are certain considerations that you must be aware of before link-editing and running applications under Language Environment.

Language Environment callable services

There is a common set of callable services designed to supplement the programmer’s language intrinsic capability. For example, COBOL application developers will find Language Environment's consistent condition handling services especially useful. For all languages the same occurs with common math services, as well as the date and time services.

Language Environment callable services are divided into the following groups:

- Condition handling services
- Date and time services
- Dynamic storage services
- General callable services
- Initialization and termination services
- Locale callable services

- Math services
- Message handling services
- National language support services

Related reading: The callable services are described in the IBM publication *Language Environment for z/VSE Programming Reference*, SC33-6685.

Language Environment calling conventions

Language Environment services can be invoked by HLL library routines, other Language Environment services, and user-written HLL calls. In many cases, services will be invoked by HLL library routines as a result of a user-specified function, such as a COBOL intrinsic function. Following is an example of the invocation of a callable math service from the COBOL language.

Example 9-8 shows how a COBOL program invokes the math callable services CEESDLG1 for log base 10.

Example 9-8 Sample invocation of a math callable service from a COBOL program

```
       77   ARG1RL  COMP-2.
       77   FBCODE  PIC X(12).
       77   RESLTRL COMP-2.
            CALL "CEESDLG1" USING ARG1RL , FBCODE ,
            RESLTRL.
```

Assembler - HLL communication via Language Environment

Assembler and High Level Language (HLL) routines can interact in LE/VSE run time. Attached are some samples outlining recommended programming techniques:

- LE/VSE-conforming Assembler (main or sub) routine calls a high-level language (HLL) routine such as COBOL/VSE, PLI/VSE, and C/VSE. This is a typical situation where the use of CEEENTRY/TERM macros is required.
- A non-LE/VSE-conforming Assembler driver calls HLL routines. Here CEEPIPI pre-initialization is the appropriate coding method.
- A C/VSE program calls an Assembler routine bridging to another C/VSE subroutine. This Assembler routine is a good candidate for EDPRLG/EPIL macros usage.

An overview is given in Figure 9-7.

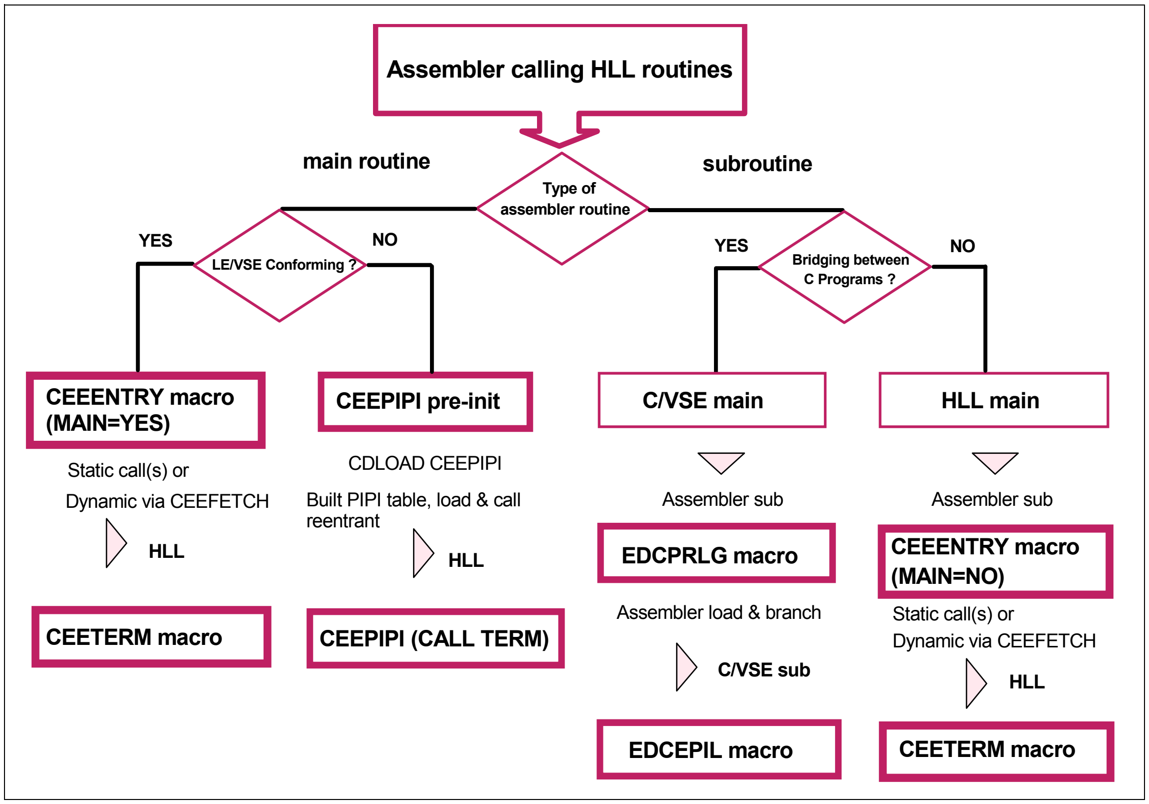


Figure 9-7 Overview of Assembler calling HLL routines

9.11 Using Java to access z/VSE resources

There is no Java Virtual Machine (JVM™) available for z/VSE, but Java can be used to access VSE functions and data via the Java-based connector, which consists of a client and a server part. The VSE Connector Client provides a Java class library that can be used to develop all kinds of Java programs on any platform where a JVM is available. This includes Linux on System z. VSE data and functions are then accessed via a TCP/IP connection to the VSE Connector Server running on VSE.

Figure 9-8 shows this scenario.

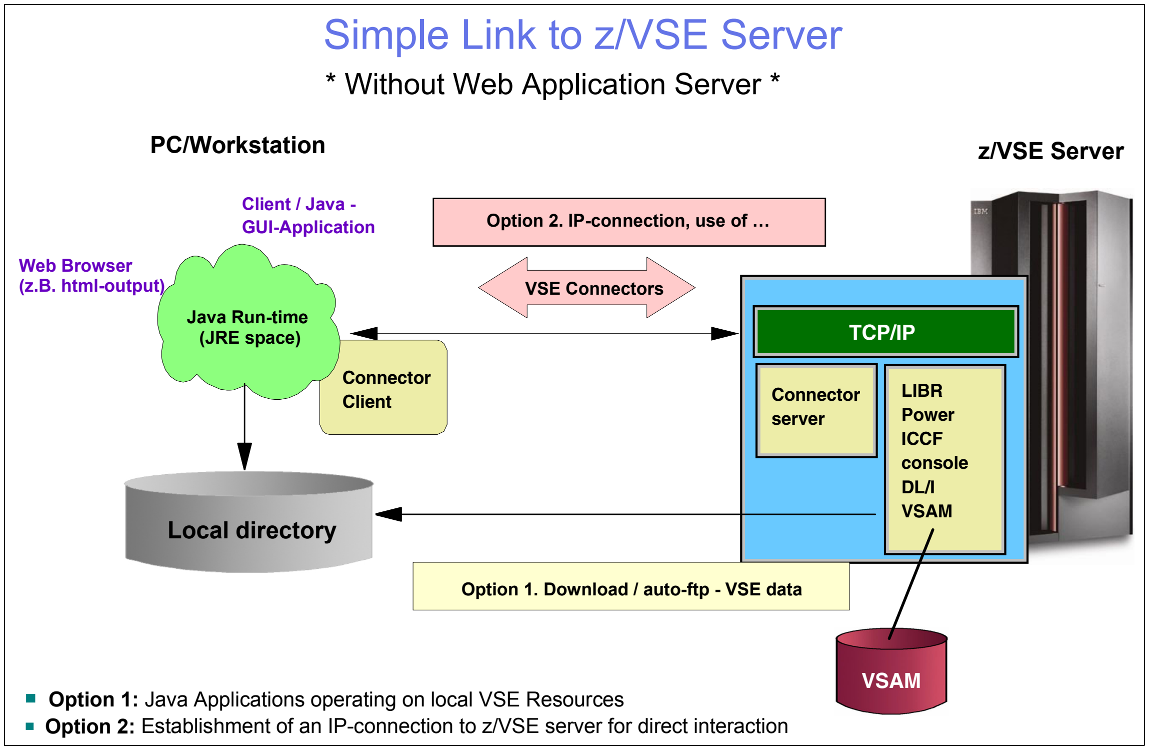

Simple Link to z/VSE Server
* Without Web Application Server *
PC/Workstation
z/VSE Server
Client / Java - GUI-Application
Option 2. IP-connection, use of …
Web Browser (z.B. html-output)
VSE Connectors
Java Run-time (JRE space)
TCP/IP
Connector Client
Connector server
LIBR
Power
ICCF
console
DL/I
VSAM
Local directory
Option 1. Download / auto-ftp - VSE data
VSAM
Option 1: Java Applications operating on local VSE Resources
Option 2: Establishment of an IP-connection to z/VSE server for direct interaction

Figure 9-8 Simple link to z/VSE server

For more information about the Java-based connector refer to Chapter 13, “z/VSE connectors” on page 333.

9.12 Traditional languages and Web orientation in z/VSE

In addition to the traditional characteristics provided by the COBOL language, the *VisualAge Generator EGL plug-in VSE* has triggered new Application Development (AD) opportunities for the VSE environment.

Based on IBM Rational Application Developer (RAD), an integrated development environment, it is possible to generate COBOL/VSE applications from a fourth generation language (4GL) that involves Web architectures.

The interaction of workstation and VSE components allows the generation and deployment of COBOL/VSE back-end programs that are called from an Enterprise Generation Language (EGL) front-end (Java applications that are located on a Web Application Server and invoked from a Web browser). This way COBOL for VSE/ESA and Java applications can interoperate in the e-business world.

- Java2 platform, Enterprise Edition (J2EE) connection architecture (J2C/JCA).
- Java Server Pages (JSP™) - The Web Application Server (WAS) is aware of the application.

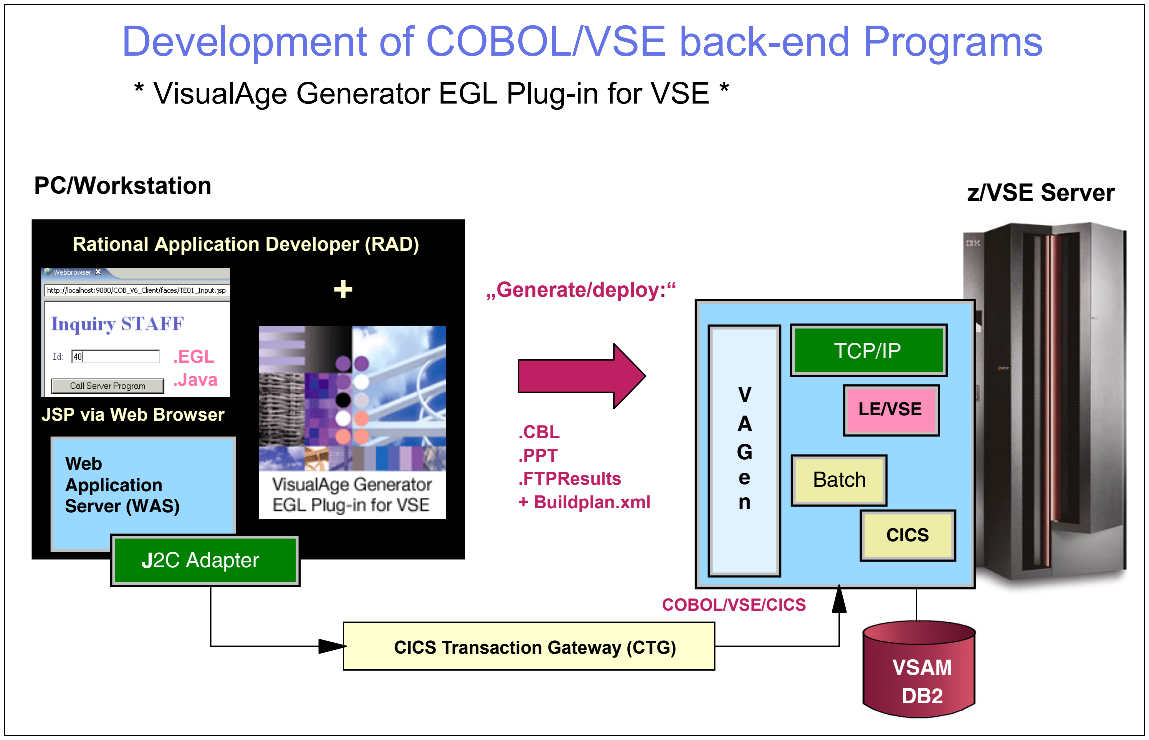

Development of COBOL/VSE back-end Programs
* VisualAge Generator EGL Plug-in for VSE *
PC/Workstation
z/VSE Server
Rational Application Developer (RAD)
Inquiry STAFF
Call Server Program
.EGL
.Java
JSP via Web Browser
Web Application Server (WAS)
J2C Adapter
VisualAge Generator EGL Plug-in for VSE
„Generate/deploy:“
.CBL
.PPT
.FTPResults
+ Buildplan.xml
VAGen
TCP/IP
LE/VSE
Batch
CICS
COBOL/VSE/CICS
CICS Transaction Gateway (CTG)
VSAM
DB2

Figure 9-9 Development of COBOL/VSE back-end programs

After development, the components for the z/VSE back-end execution are as shown in Figure 9-10.

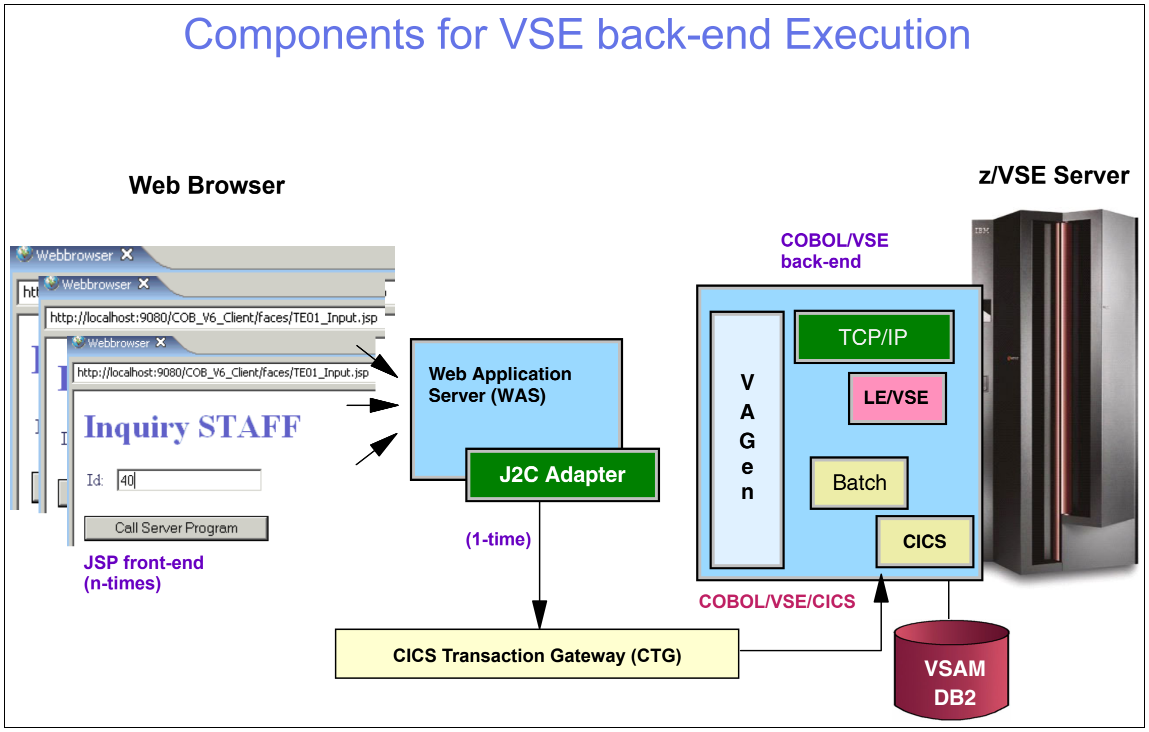


Figure 9-10 Components for z/VSE back-end execution

Related reading: You can find more information about multi-platform development via *Rational Application Developer and VisualAge Generator EGL Plug-in VSE* at the z/VSE Home (solution page):

`http://www.ibm.com/servers/eserver/zseries/zvse/solutions/egl.html`

9.13 Summary

This chapter outlined help for decisions you might need to make when designing and developing applications to run on z/VSE. Selecting a programming language to use is one important step in the design phase of an application. The application designer must be aware of the strengths as well as the weaknesses of each language to make the best choice, based on the particular requirements of the application.

A critical factor in choosing a language is determining which one is most used at a given installation. For example, if COBOL is used for most of the applications in an installation, it will likely be the language of choice for the installation's new applications as well.

When a choice for a primary language is made, it does not mean that you are locked into that choice for all programs within the application. There might be cases for using multiple languages, to take advantage of the strengths of a particular language for only certain parts of the application. Here, it might be best to write frequently invoked subroutines in Assembler language to make the application as efficient as possible, even when the rest of the application is written in COBOL or another high-level language.

Many z/VSE sites maintain a library of subroutines that are shared across the business. The library might include, for example, date conversion routines. As long as these subroutines are written using standard linkage conventions, they can be called from other languages, regardless of the language in which the subroutines are written.

Each language has its inherent strengths, and designers should exploit these strengths. If a given application merits the added complication of writing it in multiple languages, the designer should take advantage of the particular features of each language. Keep in mind, however, that when it is time to update the application, other people must be able to program these languages as well. This is a cardinal rule of programming. The original programmer might be long gone, but the application will live on and on. Thus, complexity in design must always be weighed against ease of maintenance.

There is no doubt about z/VSE focusing on traditional languages. However, as shown in the above sections, the z/VSE operating system is open in many ways for Web orientation. It is therefore at the user's choice to implement applications in a traditional or multi-platform oriented way.

Key terms in this chapter		
Assembler	Linkage editor	Compiler
Debugging	Dynamic link library	Generation
I/O (input/output)	Interpreter	Load modules
Preprocessor	Programming language	Variable

9.14 Questions for review

To help test your understanding of the material in this chapter, complete the following questions:

1. Why might a program be written in Assembler language?
2. Do companies continue to enhance the compilers for COBOL and PL/I?
3. Why is REXX called an interpreted language?
4. What are the main areas of suitability for REXX?
5. Which interpreted language can also be compiled?
6. Is the use of Language Environment mandatory in z/VSE application development?
7. Which of the data file organizations are appropriate for online applications? Which are appropriate for batch applications?
8. What is an HLL? What are some of the advantages of writing in an HLL versus Assembler language?
9. Assume that program PROG1 is invoked twice using the JCL below:

   ```
   // JOB JNM=VSEJOB,...
   *  STEP 01
   // DLBL INPUT1,'MY.data set',...
   // DLBL OUTPUT1,'X1.Y1'...
   // EXEC PROG1
   *  STEP 02
   // DLBL INPUT1,'A1.B1',...
   // DLBL OUTPUT1,'X1.Y1'...
   // EXEC PROG1
   ```

   If the // DLBL INPUT1 card were changed to use the data set A1.B1, would we be able to use the same program to process it? Assume that the new data set has the same characteristics as the old data set.
10. What is a 4GL like Enterprise Generation Language (EGL)? What are some of the advantages of writing in a 4GL versus a high-level language?
11. How can EGL be used to create new or extend legacy CICS Applications as Web and services-based applications?

10

Compiling and link-editing a program on z/VSE

Objective: As your company's newest z/VSE application programmer, you will be asked to create new programs to run on z/VSE. Doing so will require you to know how to compile, link, and execute a program.

After completing this chapter, you will be able to:

- Explain the purpose of a compiler.
- Compile a source program.
- Create executable code from a compiled program.
- Explain the difference between an object deck and a phase.
- Run a program on z/VSE.

10.1 Source modules, object decks, and phases

A program can be divided into logical units that perform specific functions. A logical unit of code that performs a function or several related functions is a *module*. Separate functions should be programmed into separate modules, a process called modular programming. Each module can be written in the symbolic language that best suits the function to be performed.

Source module
The input to a language translator (compiler)

Object deck
The output from a language translator

Each module is assembled or compiled by one of the language translators. The input to a language translator is a *source module*. The output from a language translator is an *object deck*. Before an object deck can be executed, it must be processed by the linkage editor. The output of the linkage editor is a *phase*. See Figure 10-1.

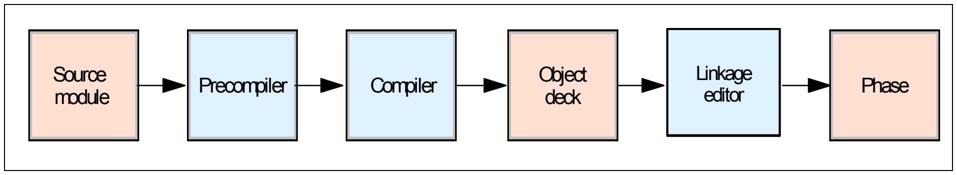


Figure 10-1 Source module, object deck, and phase

Depending on the status of the module (whatever it is—source, object, or phase), it can be stored in a VSE library. A VSE library is a VTOC-controlled or VSAM data set on direct access storage. A VSE library is divided into sublibraries that contain the members. In a library, each member can contain a source module, an object deck, or a phase.

10.2 What are source members?

A source member (or *source code*) is a set of statements written in an computer language, as discussed in Chapter 9, “Using programming languages on z/VSE” on page 217. Source programs, once they are error-free, are stored in a sublibrary *member*. A source member is identified by a one-character, alphameric member type. Source members contain the source code to be submitted for a compilation process, or to be retrieved for modification by an application programmer.

Copybook
A source member in which programmers store commonly used program segments

A *copybook* is a source member containing pre-written text. It is used to copy text into a source program at compile time, as a shortcut to avoid having to code the same set of statements over and over again. It is stored in a sublibrary that may be defined in the job control LIBDEF SOURCE statement. It should not be confused with a subroutine or a program. A copybook member is just text; it might not be actual programming language statements.

A *subroutine* is a commonly called routine that performs a predefined function. The purposes behind a copybook member and a subroutine are essentially the same: to avoid having to code something that has previously been done. However, a subroutine is a small program (compiled, link-edited, and executable) that is called and returns a result, based on the information that it was passed. A copybook member is just text that will be included in a source program on its way to becoming an executable program. The term *copybook* is a COBOL term, but the same concept is used in most programming languages.

If you use copybooks in the program that you are compiling, you can retrieve them from the LIBDEF SOURCE chain by supplying the job control LIBDEF SOURCE statement. In Example 10-1 we insert the text in member INPUTRCD from the library DEPT88.COBLIB into the source program that is to be compiled.

Example 10-1 Copybook in COBOL source code

```
//LIBDEF SOURCE,SEARCH=DEPT88.COBLIB
     . . .
      IDENTIFICATION DIVISION.
     . . .
      COPY INPUTRCD
      . . .
```

Libraries must reside on direct access storage devices (DASD).

10.3 Compiling programs on z/VSE

The function of a compiler is to translate source code into an object deck, which must then be processed by the linkage editor before it is executed. During the compilation of a source module, the compiler assigns relative addresses to all instructions, data elements, and labels, starting from zero.

Linkage editor
Converts object decks into executable phases

The addresses are in the form of a base address plus a displacement. This allows programs to be relocated (that is, they do not have to be loaded into the same location in storage each time that they are executed). See 10.4, “Creating phases for executable programs” on page 257 for more information about relocatable programs. Any references to external programs or subroutines are left as unresolved. These references will either be resolved when the object deck is linked, or dynamically resolved when the program is executed.

To compile programs on z/VSE, you can use a batch job. For compiling through a batch job, z/VSE includes a set of compile skeletons that can help you avoid some of the JCL coding you would otherwise need to do. If none of the compile skeletons meet your needs, you will need to write all of the JCL for the compilation.

As part of the compilation step, you need to define the LIBDEF statements needed for the compilation and specify any compiler options necessary for your program and the desired output.

10.3.1 What is a precompiler?

Some compilers have a precompiler or preprocessor to process statements that are not part of the computer programming language. If your source program contains EXEC CICS statements or EXEC SQL statements, then it must first be pre-processed to convert these statements into COBOL, PL/I, or Assembler language statements, depending on the language in which your program is written.

10.3.2 What is an object deck?

An *object deck* is a collection of one or more compilation units produced by an Assembler, compiler, or other language translator, and used as input to the linkage editor.

An object deck is in relocatable format with machine code that is not executable. A phase is also relocatable, but with executable machine code. A phase is in a format that can be loaded into virtual storage and relocated by supervisor fetch/load, a supervisor component that prepares phases for execution by loading them at specific storage locations.

Object decks and phases share the same logical structure consisting of:

- Control dictionaries, containing information to resolve symbolic cross-references between control sections of different modules, and to relocate address constants
- Text, containing the instructions and data of the program
- An end-of-module indication, which is an END statement in an object deck, or an end-of-module indicator in a phase

Object decks are stored in a file identified by a // DLBL IJSYSLN statement, which is input to the linkage editor job step.

10.3.3 Where are object decks stored?

You can use any z/VSE sublibrary to catalog object decks with a member type of .OBJ. The object decks to be link-edited are retrieved from the sublibraries defined by job control statement // LIBDEF OBJ,SEARCH= and transformed into an executable program. The executable program is cataloged by the linkage editor in the sublibrary defined by job control statement // LIBDEF PHASE,CATALOG=. The membertype of an executable program is .PHASE.

When using the OBJECT compiler option, you can punch the object deck on SYSPCH or SYSLNK. The job control statement // OPTION DECK or CATAL/LINK indicates whether the object deck is to be:

- Punched to SYSPCH
- Punched to SYSLNK

10.3.4 How is a linkage editor used?

Linkage editor processing follows the source program assembly or compilation of any problem program.

Linkage editor and supervisor fetch/load prepare the output of language translators for execution. The linkage editor prepares a phase that is to be brought into storage for execution by supervisor fetch/load. Primary input for the linkage editor consists of object decks and linkage editor control statements.

Output of the linkage editor is of two types:

- A phase cataloged into a z/VSE sublibrary as a member with membertype .PHASE
- Diagnostic output to SYSLST

Supervisor fetch/load prepares the executable program in storage and passes control to it directly.

10.3.5 How a phase is created

In processing object decks, the linkage editor assigns consecutive relative virtual storage addresses to control sections, and resolves references between control sections. Object decks produced by several different language translators can be used to form one phase.

A phase is composed of all input object decks processed by the linkage editor.

10.4 Creating phases for executable programs

A *phase* is an executable program stored in a VSE library. Creating a phase requires that you use the linkage editor. The phase is relocatable, which means that it can be located at any address in virtual storage within the confines of the residency mode (RMODE).

Once a program is loaded, control is passed to it, with a value in the base register. This gives the program its starting address, where it was loaded, so that all addresses can be resolved as the sum of the base plus the offset. Relocatable programs allow an identical copy of a program to be loaded in many different address spaces, each being loaded at a different starting address.

10.5 Overview of compilation to execution

In Figure 10-2 we see the relationship between the object decks and the phase stored in a library and then loaded into central memory for execution.

We start with two programs, A and B, which are compiled into two object decks. Then the two object decks are linked into one phase call MYPROG, which is stored in a library on direct access storage. The phase MYPROG is then loaded into central storage by the supervisor fetch/load, and control is transferred to it to for execution.

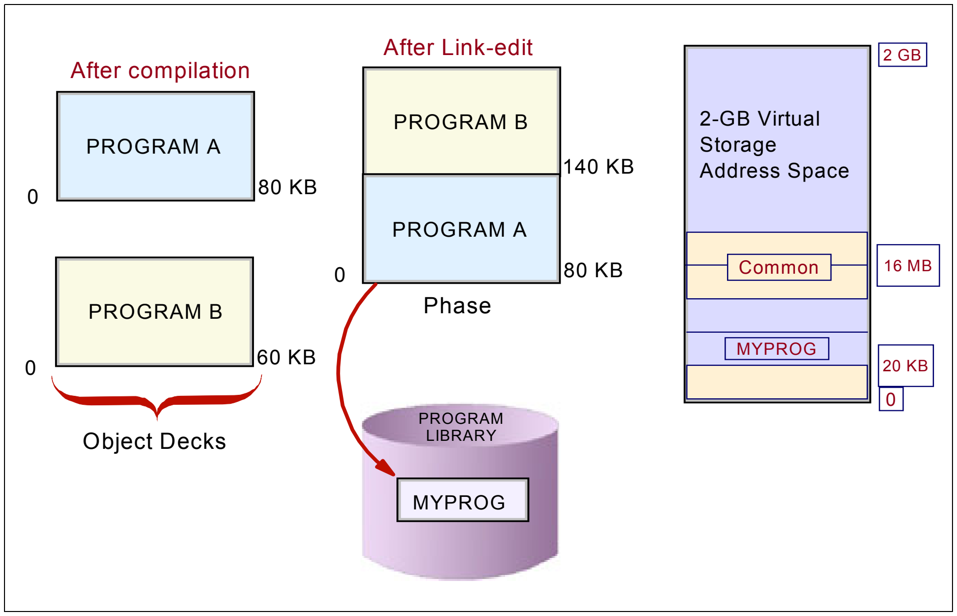


Figure 10-2 Program compile, link-edit, and execution

10.6 Using procedures

To save time and prevent errors, you can prepare sets of job control statements and place them in a sublibrary. This can be used, for example, to compile, assemble, link-edit, and execute a program. For a more in-depth discussion on JCL procedures see 6.5, “Using JCL procedures” on page 170.

To execute a procedure, a job control LIBDEF PROC,SEARCH= statement must be specified to define the sublibrary search chain.

10.7 Summary

This chapter described the process for translating a source program into an executable phase, and executing the phase. The basic steps for this translation are to compile and link-edit, although there might be a third step to pre-process the source prior to compiling it. The pre-processing step would be required if your source program issues CICS command language calls (SQL calls). The output of the pre-processing step is then fed into the compile step.

The purpose of the compile step is to validate and translate source code into relocatable machine language, in the form of object code. Although the object code is machine language, it is not yet executable. It must first be processed by a linkage-editor before it can be executed.

The linkage-editor takes as input object code and then produces an executable phase. This process resolves any unresolved references within the object code and ensures that everything that is required for this program to execute is included within the final phase. The phase is now ready for execution.

To execute a phase, it must be loaded into central storage. The supervisor fetch/load loads the phase into storage and then transfers control to it to begin execution. Part of transferring control to the module is to supply it with the address of the start of the program in storage. Because the program’s instructions and data are addressed using a base address and a displacement from the base, this starting address gives addressability to the instructions and data within the limits of the range of displacement[1].

Key terms in this chapter		
Source module	Copybook	Linkage editor
Phase	Object deck	Object-oriented code
Procedure	LIBDEF chain	Relocatable

[1] The maximum displacement for each base register is 4096 (4K). Any program bigger than 4K must have more than one base register in order to have addressability to the entire program.

10.8 Questions for review

1. What steps are needed to be able to execute a source program?
2. Why must a source program containing EXEC CICS or EXEC SQL statements be processed by the precompiler?
3. How is the output (on SYSPCH) of a compiler or the Assembler called?
4. How is the output (on SYSPCH) of the Linkage editor called?

Part 3

Online workloads for z/VSE

In this part we examine the major categories of online or interactive workloads performed on z/VSE, such as transaction processing, database management, and Web connecting. The chapters that follow guide you through discussions of network communications and several popular middleware products, including DB2, CICS, and z/VSE connectors.

11

Transaction management systems on z/VSE

Objective: To expand your knowledge of mainframe workloads, you must understand the role of mainframes in today's online world. This chapter 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 chapter we focus on the most widely used transaction management product for z/VSE: CICS TS.

After completing this chapter, you will be able to:

- Describe the role of mainframe systems in a typical online business.
- List the attributes common to most transaction systems.
- Explain how databases are used in a typical online business.
- Explain the role of CICS in online transaction processing.
- Describe CICS programs, CICS transactions, and CICS tasks.
- Explain what conversational and pseudo-conversational programming is.

11.1 Online processing on the mainframe

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

Also, 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). Later in this chapter 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.

11.2 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 began, its IT staff designed 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 handled through periodic batch updates.

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 11-1).

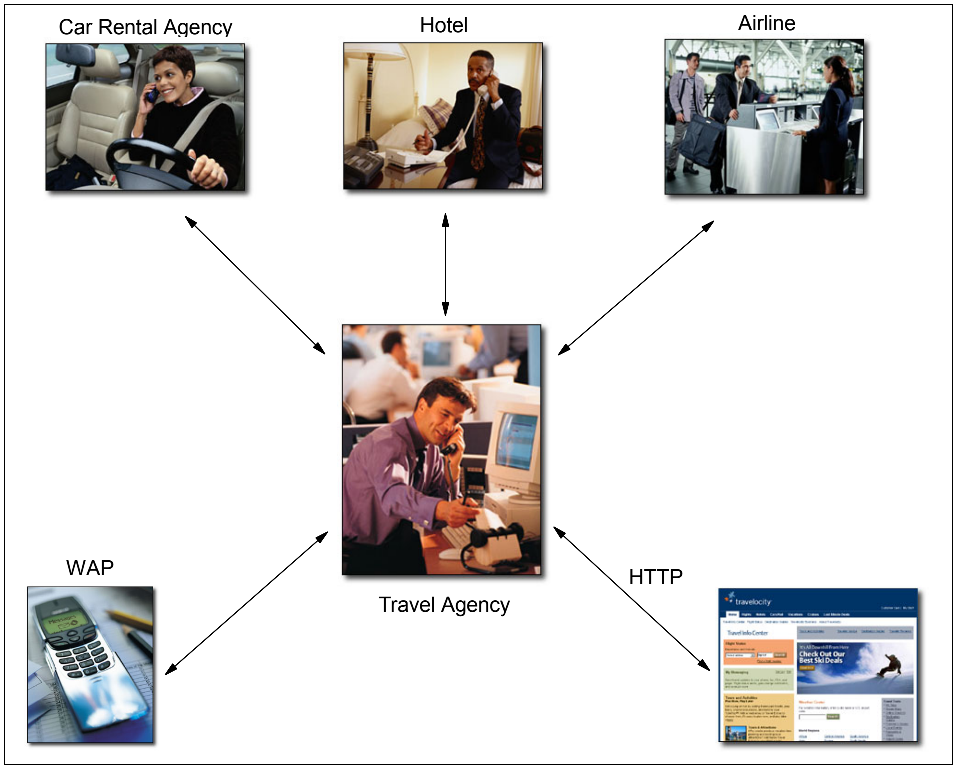


Figure 11-1 A practical example

Therefore, the agency IT staff created 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.

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

11.3 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:

- A small amount of data is processed and transferred per transaction.
- There are large numbers of users.
- Transactions are executed in large numbers.

11.3.1 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 IBM CICS product) rely on the *multitasking* and *multithreading* capabilities of z/VSE to allow more than one task to be processed at the same time, 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
A single copy of an application can be processed by several transactions concurrently.

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 a working storage section being obtained each time the program is invoked.

11.3.2 What is a transaction system?

Figure 11-2 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.

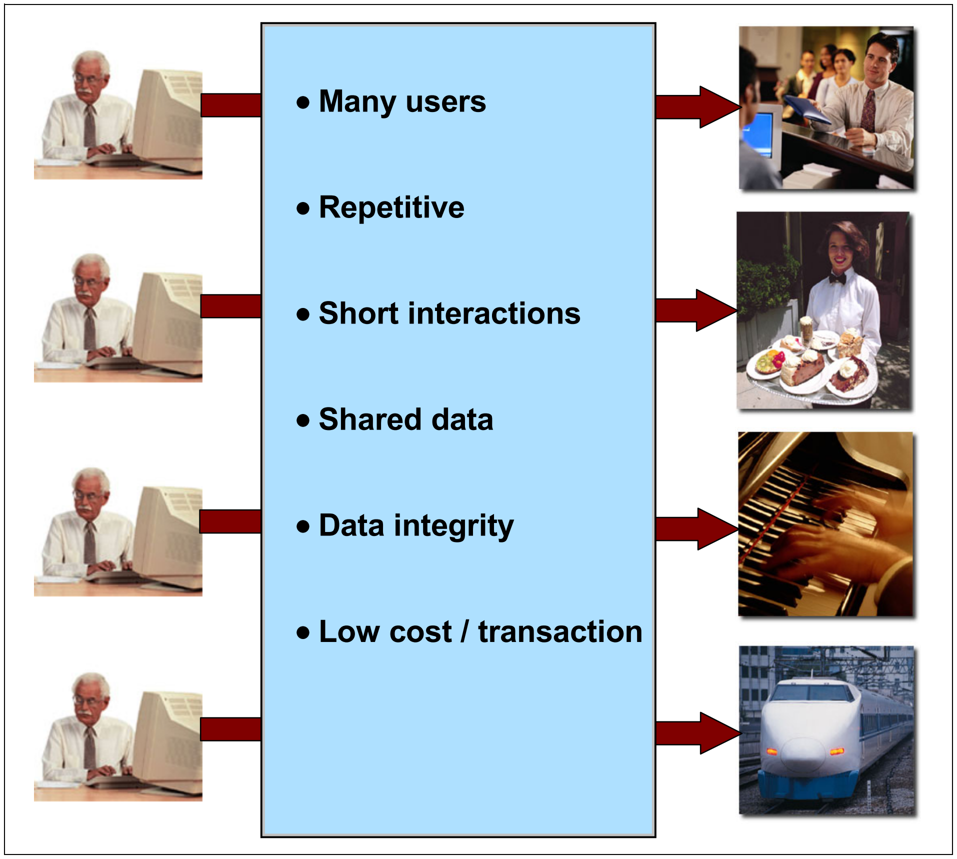


Figure 11-2 Characteristics of a transaction system

Transaction
A unit of work performed by one or more transaction programs, involving a specific set of input data and initiating a specific process or job

Transaction systems must be able to support an unpredictable 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 short. 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

11.3.3 What are the typical requirements of a transaction system?

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

- *A*tomicity. The processes performed by the transaction are done as a whole or not at all.
- *C*onsistency. The transaction must work only with consistent information.
- *I*solation. The processes coming from two or more transactions must be isolated from one another.
- *D*urability. 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

11.3.4 What is commit and roll back?

In transaction systems, *commit* and *roll back* refer 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.

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.

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 11-3 illustrates the two-phase commit.

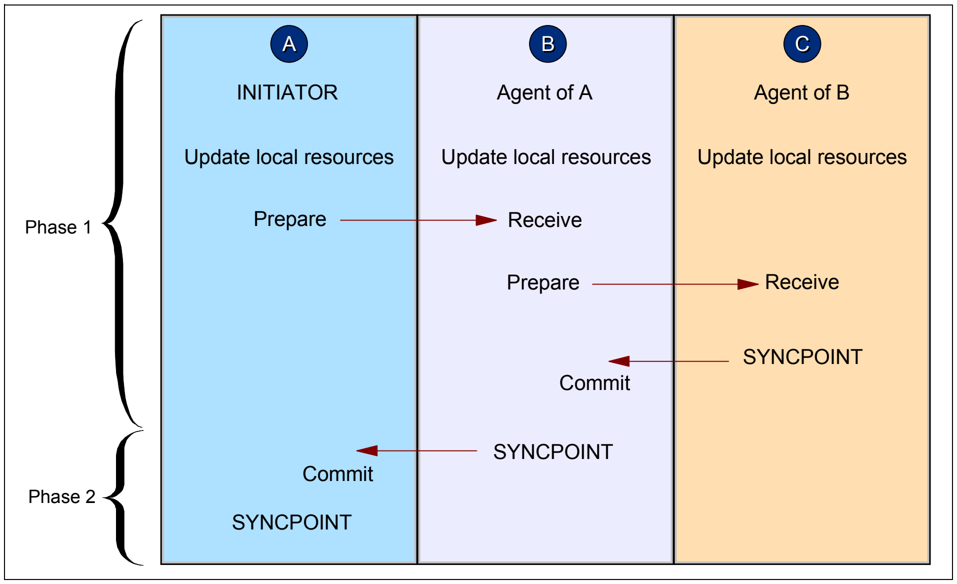


Figure 11-3 Two-phase commit

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

11.4 What is CICS?

CICS stands for *Customer Information Control System*[1]. It is a general-purpose transaction processing subsystem for the z/VSE and z/OS operating systems. CICS provides services for running an application online, by request, at the same time that 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,

[1] The current version of CICS in z/VSE is *CICS Transaction Server for VSE/ESA* (CICS TS). There might be VSE systems that still use *CICS/VSE*, the predecessor of CICS TS. If not already specified, we use the term *CICS* for CICS TS in this book.

and performs many of the same functions as the z/VSE and z/OS operating systems.

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.

11.4.1 CICS in a z/VSE system

In a z/VSE 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 does not 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.

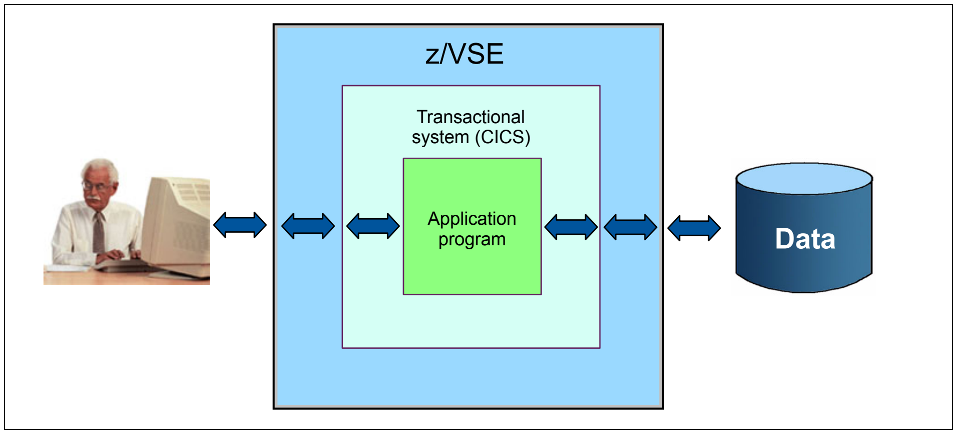


Figure 11-4 Transactional system and the operating system

A z/VSE system might have multiple copies of CICS running at one time. Each CICS starts as a separate z/VSE partition, also named CICS region. CICS provides an option called multi-region operation (MRO), which enables the separation of different CICS functions into different CICS partitions, so a specific CICS partition (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.

11.4.2 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 four-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

Unit of work
A transaction; a complete operation that is recoverable

  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.

11.4.3 Using programming languages

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

Most of the time, you use the CICS command-level programming interface, EXEC CICS. This is the case for all four supported program languages. These commands are defined in detail in the *CICS Application Programming Reference*.

11.4.4 Conversational and pseudo-conversational programming

Conversational transaction
A program conducts a conversation with a user.

In CICS, when the programs being executed enter into a conversation with the user, it is called a *conversational transaction* (also see Figure 11-5). A non-conversational transaction (see Figure 11-6 on page 276), 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.

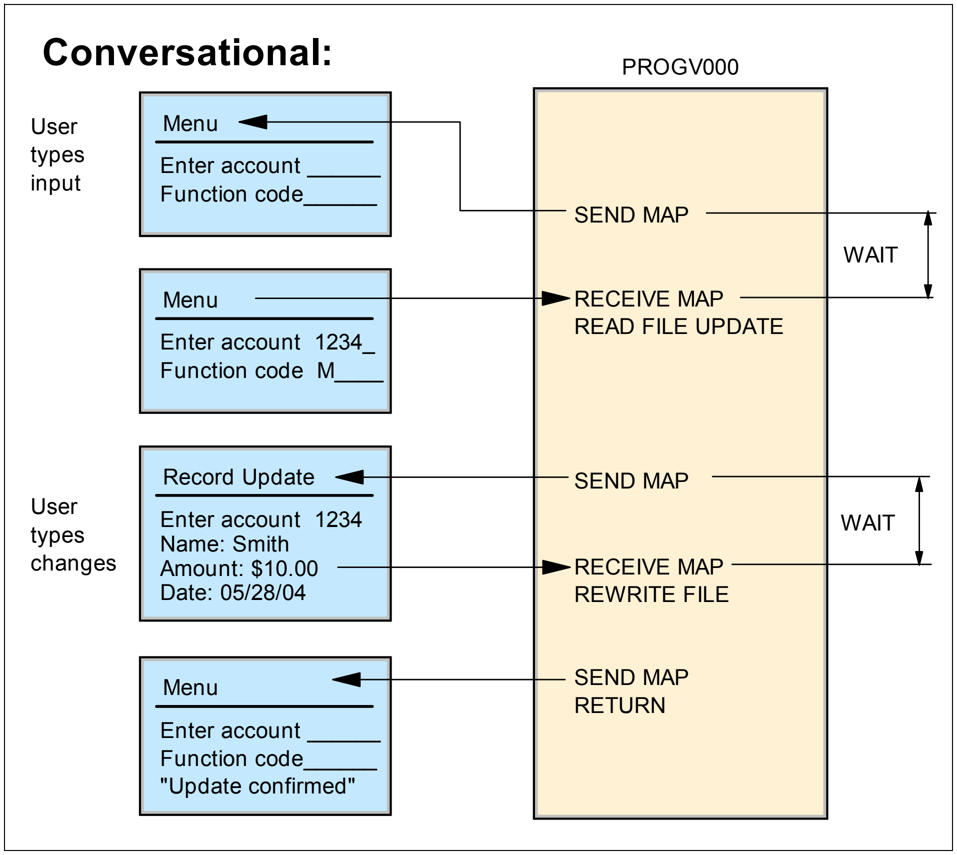


Figure 11-5 Example of a conversational transaction

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 reads the input when the user sends it. Figure 11-5 on page 275 and Figure 11-6 show different types of conversation in an example of a record update in a banking account.

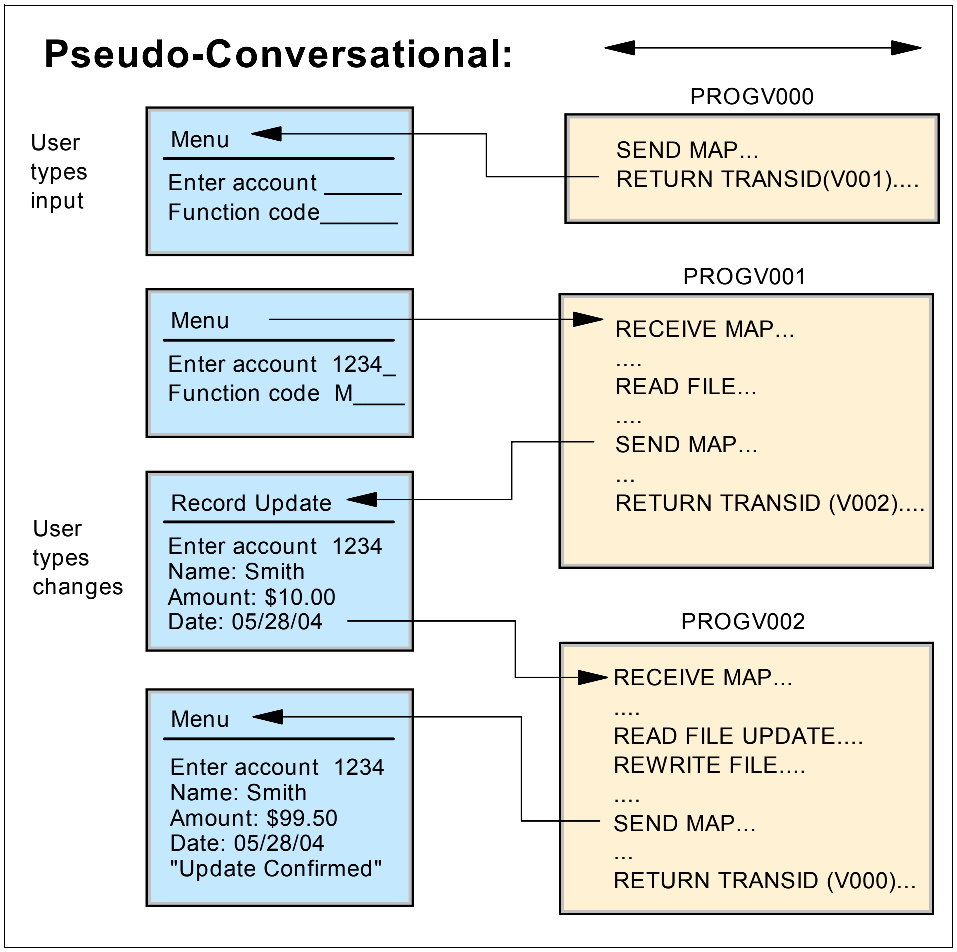


Figure 11-6 Example of a pseudo-conversational transaction

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

This pseudo-conversational processing technique is the preferred one from a performance point of view because of freeing all task-related storage between interactions. Another advantage is that all locked resources for a task are freed/committed after sending a response to the user. In case of any failure, the standard recovery/restart facility of CICS can take place to ensure data integrity.

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 11-6 on page 276).

More information can be found in the *CICS Application Programming Guide.*

11.4.5 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), just include a CICS command in your code. In COBOL, for example, CICS commands look like this:

```
EXEC CICS function option1 option2 ... END-EXEC.
```

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

An *option* is some specification that is associated with the function. Options are expressed as keyboards. 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 does not 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 Example 11-1.

Example 11-1 CICS command example

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

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 can be altered by the EXEC interface modules.

11.4.6 How a CICS transaction flows

To begin an online session with CICS, users usually begin by signing on, the process that 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 is not 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 is not 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 for a response from a terminal, CICS is able to pass 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 11-7, Figure 11-8 on page 280, and Figure 11-9 on page 280 should help you understand what goes on.

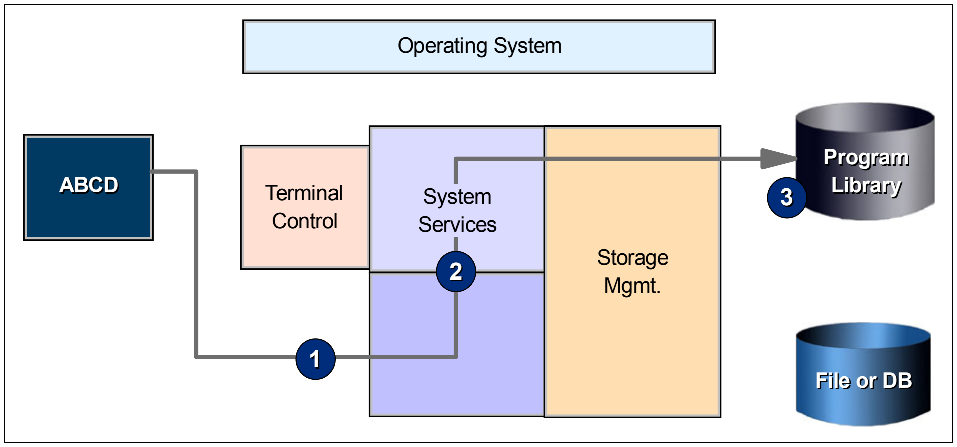


Figure 11-7 CICS transaction flow (part 1)

The flow of control during a transaction (code ABCD) is shown by the sequence of numbers 1 to 8. (We are only using this transaction to show some of the stages that 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.

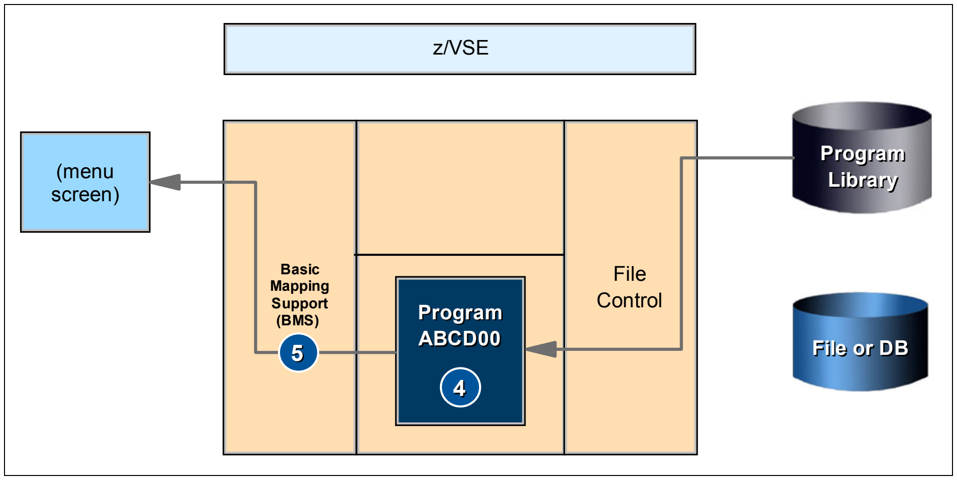


Figure 11-8 CICS transaction flow (part 2)

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.

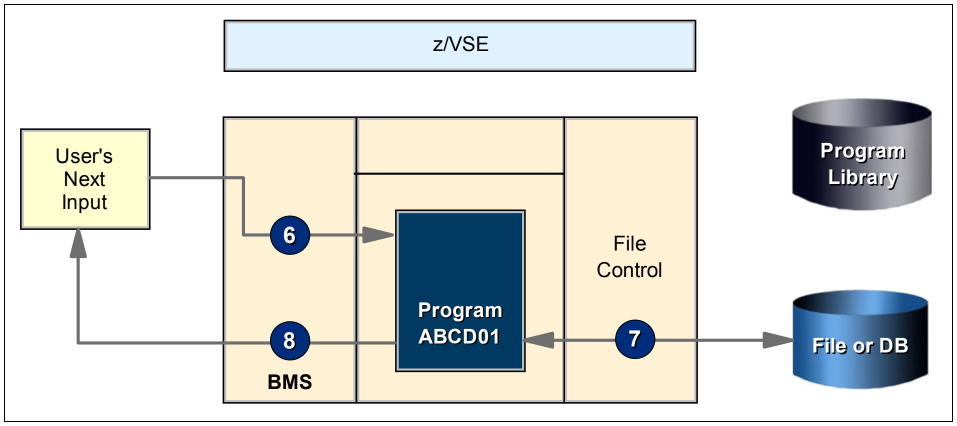


Figure 11-9 CICS transaction flow (part 3)

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.

11.4.7 CICS services for application programs

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

Application Programming Interface

Use the *application programming interface* (API) to access CICS services from the application program. You write a CICS program in much the same way in which 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 is not easy to deal with *terminal control services* in a direct way. *Basic Mapping Support* (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 CICS data management services:

- *CICS file control* offers you access to data sets that are managed by either the Virtual Storage Access Method (VSAM) or the Direct Access Method (DAM). CICS file control lets you read, update, add, and browse data in VSAM and DAM data sets and delete data from VSAM data sets.
- *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

Other CICS services are:

- *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 (DASD). 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, request notification when a specified time has expired, and so on.
- *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 or a system dump in case of a more severe failure within CICS. The CICS trace is a debugging aid for application programmers that produces trace entries of the sequence of CICS operations.

11.4.8 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 program definitions, finds out where it is, and loads it if it is not 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. 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.

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 11-10 illustrates this concept.

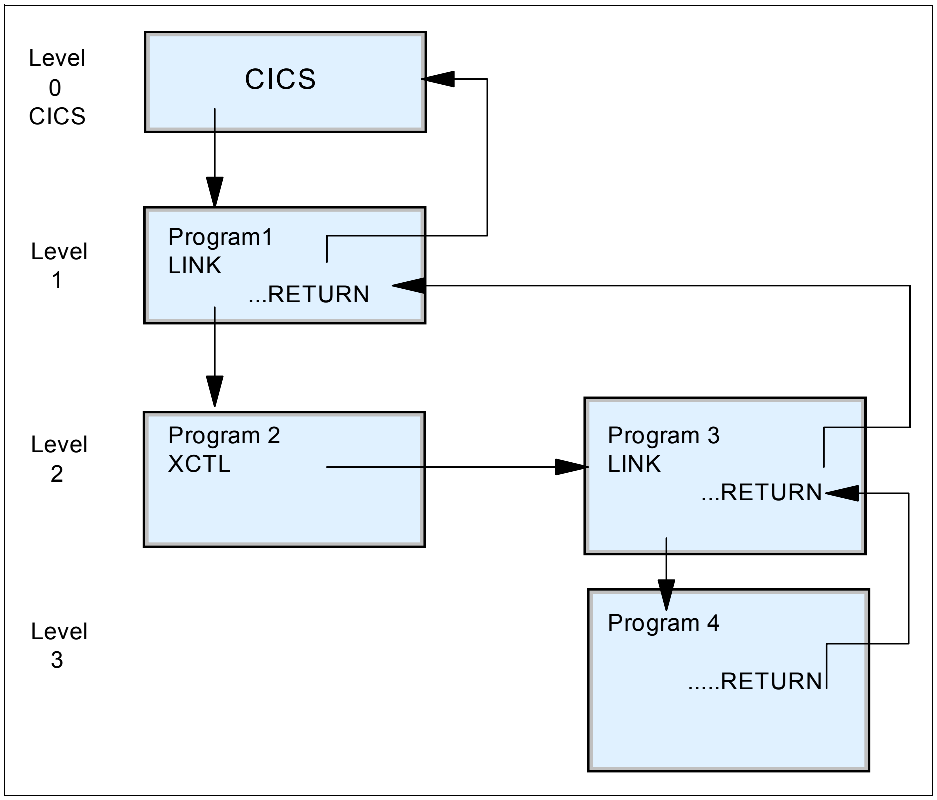


Figure 11-10 Transferring control between programs (normal returns)

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 or the area to which results are to be returned. The COMMAREA interface is also an option to invoked 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.

11.4.9 CICS programming roadmap

Typical steps for developing a CICS application that uses the EXEC CICS command level programming interface are as follows:

1. Design the application, identifying the CICS resources and services you will use. See the chapter on Application design in the *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.

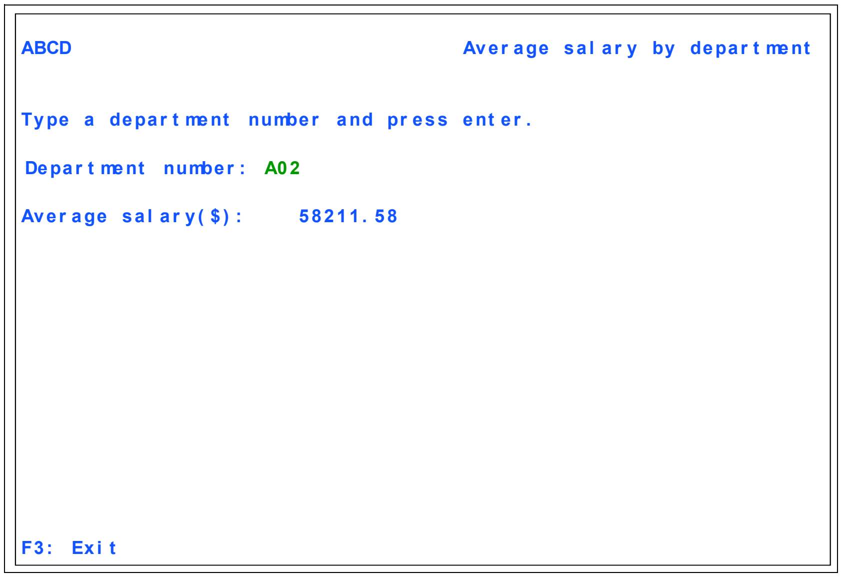
   One of the needed components for online transactions is the screen definition, that is, the layout of what is displayed on the screen (such as a Web page). In CICS we call this a *map*.
3. Depending on the compiler, you might only need to compile the program and install it in CICS, or you might need 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*.

11.4.10 Our online example

When we look back to our travel agency example, examples of CICS transactions might be:

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

Figure 11-11 shows the possibility to 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 11-11 CICS application user screen

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

11.5 Summary

In this chapter we learned that transaction applications keep changing, depending on the needs of the organization, its customers, and suppliers. At other times, changes are implemented through new technologies, but the dependable, solid application remains unchanged. Interaction with the computer happens online through the help of a transaction manager. Many transaction managers and database managers exist, but their principles are the same.

CICS is a transactional processing subsystem. That means that 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.

You write a CICS program in much the same way that you write any other program. You can use COBOL, C, PL/I, or Assembler language to write CICS application programs. Most of the processing logic is expressed in standard language statements, but you use *CICS commands*. The CICS commands are grouped according to their function, terminal interaction, access to files, or program linking. Most of the CICS resources may be defined and altered online through CICS-supplied transactions. Some resources like transient data have to be defined via control tables. Other supplied transactions allow us to monitor the CICS system.

Key terms in this chapter		
CICS TS	Conversational	Pseudo-conversational
CICS command	Transaction	Region
Multi-threading	Unit of work	Basic mapping support (BMS)

11.6 Questions for review

To help test your understanding of the material in this chapter, complete the following questions:

1. Describe the main phases in the CICS programming road map.
2. How might the meaning of *business transaction* differ from *CICS transaction*?
3. How do you define resources in CICS?

11.7 Exercise: Create a CICS program

Here is an exercise to try.

During this exercise, you might find it helpful to consult the *CICS Application Programming Guide*.

Analyze and update the class program

- Think of a possible use of the COMMAREA.

  Think of passing data between programs called with LINK or XCTL. A generic program for error processing may be developed. All of the invocations to it may be done passing the required error data through the COMMAREA. Also, the COMMAREA option of the return command is designed for passing data between successive transactions in a pseudo-conversational sequence.

The state of a resource may be passed by the first transaction through COMMAREA in order to be compared to its current state by the second transaction. It may be necessary to know if this has changed since the last interaction before allowing an update. In Web applications, the business logic in a CICS application can be invoked using the COMMAREA interface.

- Several simple updates to the class program transaction may be done quite easily:
  - Include one additional output field in the screen. The maximum value of employee commissions could be an example.

    A new field has to be defined in the map source. Perhaps some literals have to be changed. Assemble the map and generate the new copy file. Modify the program to have another column in the SQL statement and move its content after retrieval to the corresponding new output field in the map. Execute the preparation job for the user program. New copies for program and map are required in the CICS session.
  - Create a transaction that could be like a main menu. One of the options would start the current program.

    Only two variable fields are required in the map for this transaction: the option field and the message line. Only one option has to be initially included, the one for the current ABCD transaction. The same mapset may be used to include the new map. The ABCD transaction has to be modified to do the RETURN TRANSID to the new transaction. Only the following resources have to be added to the CICS system: the new transaction and programs (user program and map).
  - Learn about the CICS HANDLE CONDITION statement and find out where it may be used.

    Try to add error control to the RECEIVE CICS command. The MAPFAIL condition occurs when no usable data is transmitted from the terminal after a RECEIVE command.

Business transaction

Analyze a typical business transaction. Think of different CICS programs and transactions that could be needed to accomplish this. Draw a diagram to show the flow of the process.

The example that is developed in the manual *CICS Application Programming Primer* could be appropriate. A department store with credit customers keeps a master file of its customers' accounts. The application performs the following actions:

- Displays customer account records
- Adds new account records
- Modifies or deletes existing account records
- Prints a single copy of a customer account record
- Accesses records by name

12

Database management systems on z/VSE

Objective: You will need a good working understanding of the major types of system software used to process online workloads on the mainframe. In this chapter we focus on two of the most widely used database management system (DBMS) products for z/VSE: DB2 and DL/I.

After completing this chapter, you will be able to:

- Explain how databases are used in a typical online business.
- Explain the role of DB2 in online transaction processing.
- List common DB2 data structure.
- Give an overview of application programming with DB2.
- Explain what the DL/I components are.
- Describe the structure of the DL/I subsystem.

12.1 Database management systems for the mainframe

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

12.2 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 number, 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 12-1 illustrates an entity relationship mode.

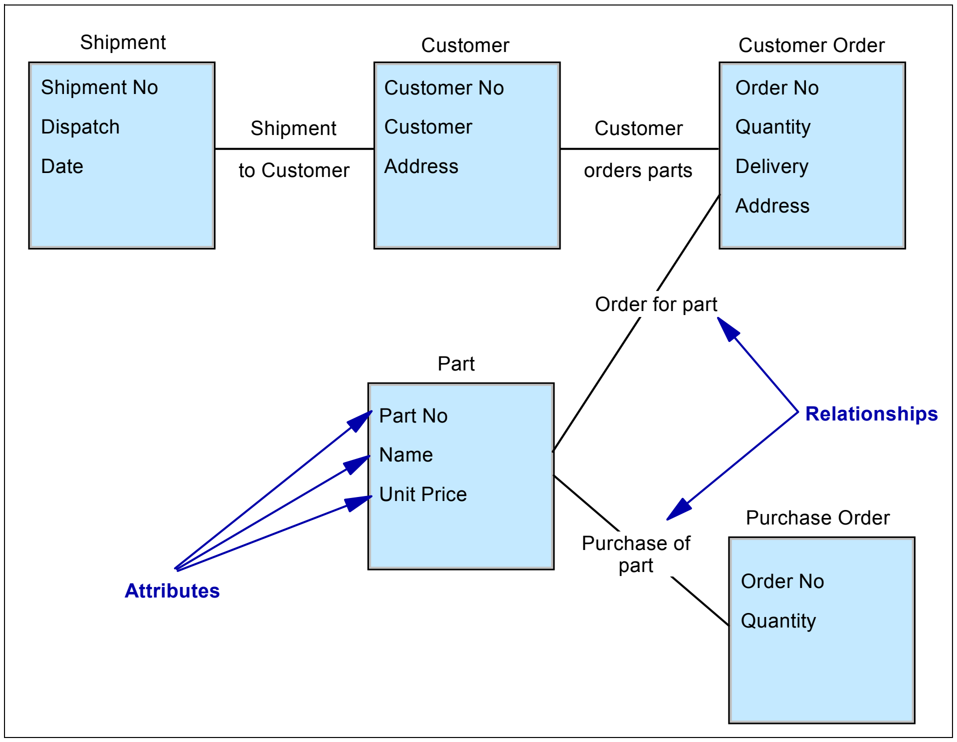


Figure 12-1 Entities, attributes, and relationships

A database management system (DBMS), such as the DB2 product, provides a method of storing and using the business data in the database.

12.3 Why use a database?

DBMS
Database management system - provides a method of storing and using data in a database.

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. But a properly designed and implemented DBMS provides many advantages over a flat file system like VSAM:

- 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 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 caused 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 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 table (in DB2), while in a 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 file be reassembled or recompiled, even if it does not need the added or changed fields. A properly designed database 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.

12.4 Who is the database administrator?

Database administrators (DBAs) are primarily responsible for specific databases in the subsystem (DBMS). In some companies, DBAs are given the system administrator authorization besides the database administration rights. This 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 data objects, beginning with the database, then 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 tables and indexes, and assigning storage.

There are many tools that can assist the DBA in these tasks. 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 person or 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.

12.5 How is a database designed?

The process of database design, in its simplest form, can be described as the structuring of the data elements for the various applications, in such an order that:

- Each data element is readily available by the various applications, now and in the foreseeable future.
- The data elements are efficiently stored.
- 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.

12.5.1 Entities

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

- Can be uniquely defined
- We may 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, and so on. 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.

12.5.2 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).

12.5.3 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 (for example, fastener) may consist of several other parts (bolt, nut, and washer).

12.5.4 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. 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.

12.5.5 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.

12.6 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 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/VSE: VSAM, 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.

Root
The top level of a hierarchy

Both database systems offer the benefits listed in 12.3, "Why use a database?" on page 293. 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.

12.6.1 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 tables and indexes. 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:

SQL
Structured Query Language - a language used to interrogate and process data in a relational database

– A base table that is created and holds persistent data

– A temporary table that stores intermediate SQL results

– A results table that is returned when you use an SQL statement to query tables

Table 12-1 Example of a DB2 table (department table)

Dept No	Dept Name	Mngr No	Adma Dept
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
B01	PLANNING	0000020	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

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.

- 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:

    - It must have a value (that is, it cannot be null).
    - 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.

  - 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.

12.7 What is DB2?

DB2 is a relational database management system (RDBMS). The general concepts of a relational database management system are discussed in Chapter 11, “Transaction management systems on z/VSE” on page 263, and in 12.6, “What is a database management system?” on page 299.

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*. A *schema* is a logical grouping of these objects.

12.7.1 Data structures in DB2

Earlier in this chapter we discussed most of the basic structures common to DBMSs. Now let us look at several structures that are specific to DB2 for VSE.

Table and dbspace

A *table* is a logical construct. It is kept in a dbspace, which is a logical space definition limiting the maximum size of the tables and indexes within that space.

Dbspaces are structures that can contain one or more tables. They are assigned to storage pools, which are built up of physical data sets called dbextents. DB2 for VSE uses VSAM data sets to store the data. This means that each *dbextent* is a VSAM data set.

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 have a department table and a home address table. You can create a view on these tables to only let users have access to one particular department in order to update salary information and home address information of these employees. You do not want them to see the salaries or home addresses in other departments. You create a view of the tables that only lets the users see one department together with the related home addresses, 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.

Index

An *index* is another logical construct that is stored similar to tables, in a dbspace. Indexes are used for faster access to data. The purpose of non-unique indexes is to provide efficient access to a group of data. Unique indexes have the additional purpose of ensuring that key values are unique. Even when present, the index is not always used. The database manager selects an access path to the data based on a combination of factors.

Storage pool

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

12.7.2 Storage concept

A DB2 Server for VSE & VM database is a collection of tables and indexes (user data objects and supporting information) maintained by the database manager. The supporting information includes control information such as how each data table is formatted and where each is located, and data recovery information.

The physical implementation of the database consists of:

- The directory

  In VSE it is a VSAM data set. It includes mappings of the dbspace pages (4 KB per page) to their addresses on the DASD (that is, it relates the logical database image to the physical storage used).

- One, two, or four logs

  In VSE, these are VSAM data sets. These contain information about the changes made to the data. If any changes must be *undone* or *redone*, logs can be used to restore the data to its proper state.

- One or more storage pools

  In VSE these are collections of VSAM data sets. Each is called a database extent or dbextent. A dbextent is an allocation of actual DASD space. Storage pools are composed of one or more dbextents. The size of the storage pool can be increased or reduced by:

  – Adding more dbextents
  – Deleting existing dbextents

  Note: In z/VSE, each dbextent is the primary allocation of a VSAM data set (CLUSTER).

When a table is created, it must be assigned to a logical allocation of storage called a dbspace.

The table creator (that is, the user that creates the database) can either do this assignment explicitly, or let the database manager use a default assignment.

Any indexes created on that table will be stored in the same dbspace.

Figure 12-2 shows how tables are stored in the database. It includes two tables and their indexes in dbspace A, two tables and their indexes in dbspace B, and one table with three indexes in dbspace C.

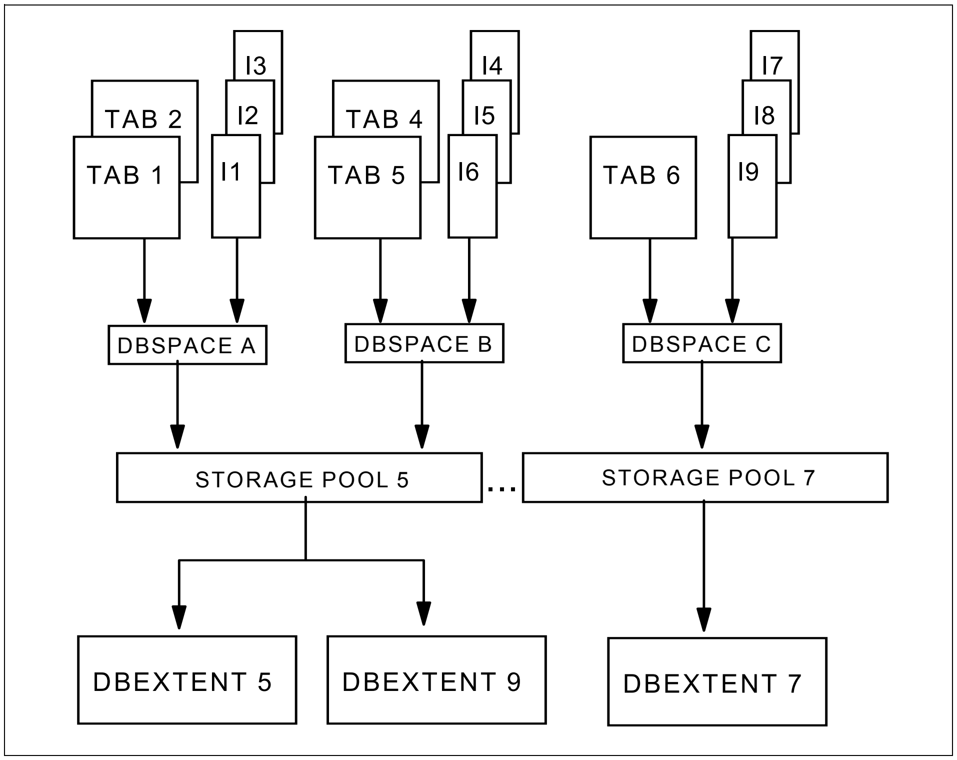


Figure 12-2 Tables in a database

12.7.3 Schema structures

Stored procedure concepts

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 it is defined to DB2 and managed by the DB2 subsystem.

The stored procedure server

A *stored procedure server* is an application requester that is local to the database manager and is used to execute the stored procedure. It runs in its own partition.

In DB2 Server for VSE & VM, all stored procedures are fenced, which means that they are separated from the database manager with respect to execution and memory usage. This is necessary to ensure that a stored procedure does not:

- Inadvertently use storage that is allocated to the database manager.
- Monopolize processing in the database machine or partition.

A fenced implementation is achieved through the use of stored procedure servers.

12.7.4 System structures

Catalog and directory

DB2 itself maintains a set of tables that contain metadata or data about all of the DB2 objects in the database. The catalog keeps information about all the objects, such as the tables, views, indexes, dbspaces, and so on, while the directory keeps information about the database storage and the mapping of physical to logical definitions. 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, and its dbspace in the catalog and puts this information in the catalog table called SYSTEM.SYSCATALOG. All of the columns defined in the table are automatically recorded in the SYSTEM.SYSCOLUMNS table.

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

Page buffers

The number of *pages buffers* defines the area of virtual storage in which DB2 temporarily stores pages of tables and 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 page buffer. If the needed data is already in a buffer, expensive I/O access to the disk can be avoided.

Current and alternate 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 *current log*.

When the current log is full, DB2 wants to archive the contents to a disk or tape data set called the logarchive. If the log cannot be archived immediately, DB2 can switch to an alternate log data set to continue processing. The now inactive log has to be archived before DB2 can switch back to the inactive log data set when the now current log has to be archived. DB2 uses the stored information in recovery scenarios, for system restarts, or for any activity that requires reading the log.

12.7.5 Integration of the basic DB2 components in z/VSE

The previous text showed the structure of DB2. In Figure 12-3 we see how the basic components of DB2 work together in z/VSE.

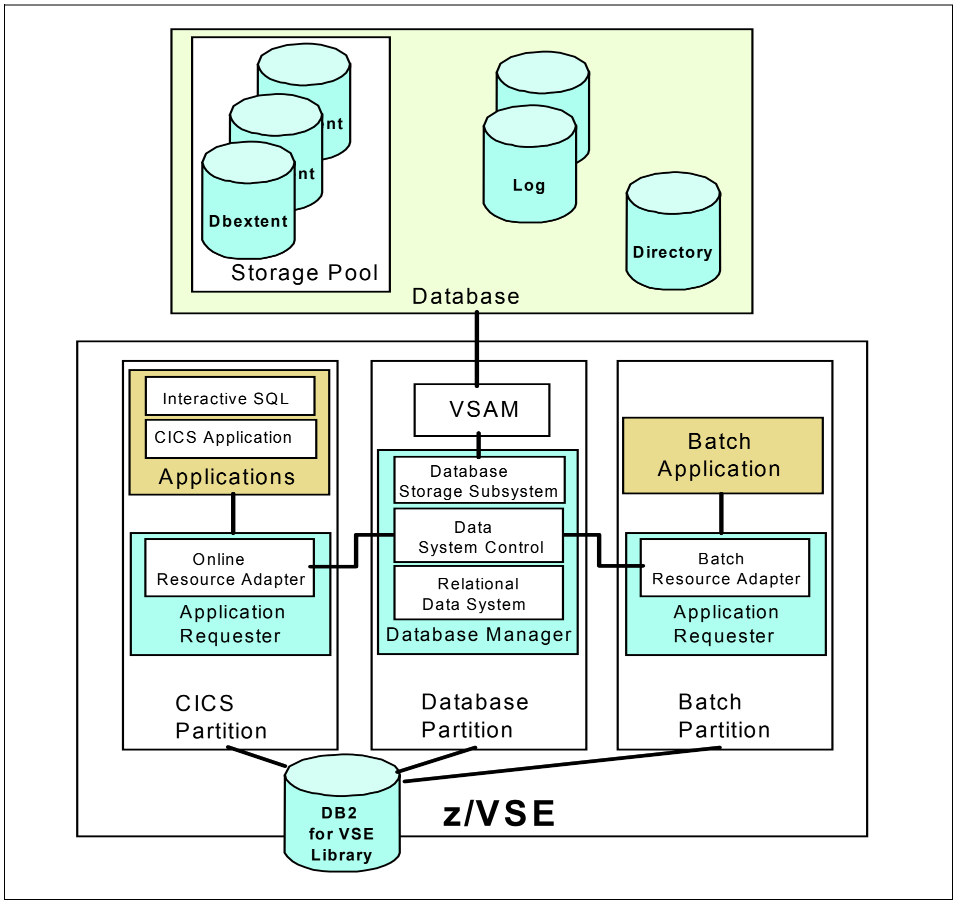


Figure 12-3 Overview of DB2 components in z/VSE

The basic DB2 components in z/VSE are:

- Database

  This is composed of:

  – A collection of data contained in one or more storage pools, each of which in turn is composed of one or more database extents (dbextents). A dbextent is a VSE VSAM data set.

  – A directory that identifies data locations in the storage pools. There is only one directory per database.

  - A log that contains a record of operations performed on the database. A database can have one, two (dual or alternate), or four (dual and alternate) logs.

- Database manager

  This is the program that provides access to the data in the database. The database manager is running in its own partition in a VSE environment.

  The application server of database manager is the component that receives and processes requests issued by the application requester. It is the facility that responds to requests for information from and updates to the database. It is composed of the database and the database manager.

- Application requester

  This is the component that accepts a request from an application and passes it to an application server. It is the facility that transforms a request from an application into a form suitable for communication with an application server.

- Online resource adapter

  This incorporates the application requester component for use with CICS transactions. It runs as a background task working as a CICS external resource manager getting called for every SQL statement issued by a CICS application.

- Batch resource adapter

  This implements the application requester component for VSE batch applications. It is loaded together with the application in the same VSE partition (getvis area) and gets called for every SQL statement to be processed on behalf of the application.

- Interactive SQL (ISQL)

  This is a utility running as a CICS application (transactionID 'ISQL') providing an interactive interface to the user to enter SQL statements and some operator commands.

- CICS application

  This is a program prepared and compiled to run under control of CICS. It is started on user request using the assigned transaction ID (four-letter symbolic name) with or without required parameters. Normally such applications are designed to communicate interactively with the user using maps and panels displayed on the terminal.

  CICS Applications can make use of all resources and data sources available in CICS.

- Batch application

  This is a program prepared and compiled to run in a batch partition. So it is executed in an own address space together with the batch resource adapter. Any SQL statements issued by this application are routed to the DBMS.

  The purpose of a batch application is to execute process logic that does not need interactive communication with the user such as bulk data updates or generating reports from the database.

12.7.6 Database administration

Database administrators are primarily responsible for specific databases in the subsystem. In companies, DBAs are given the special authorization *DBA*, which gives them the ability to do almost everything in the DB2 subsystem, and gives them jurisdiction over the whole database.

Creation and management of DB2 objects

The DBA creates the hierarchy of DB2 objects, beginning with the database, then storage pools, dbspaces, 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 called dbextents, which make up the storage pools containing the tables and indexes in logical units (so-called dbspaces_.

If objects increase in size, the DBA is able to alter certain objects, and add or remove definitions to make the appropriate changes.

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

Using DB2 utilities

The DBA maintains the database objects by using a set of commands, which can be submitted using JCL jobs. Usually a company will have a data set library for these jobs that DBAs can copy and use.

The utility that helps the DBAs to do their job and process the administrative commands in jobs is the Database Services Utility (DBSU). There are different tasks to use this utility:

- Data organization

  Once tables are created, the DBA uses the DATALOAD function of the utility to populate them. There is the UNLOAD function to save single tables or a set of tables (dbspace), including their definitions. It is possible to keep the data in a certain order by using the utility UNLOAD/RELOAD functions for reorganization. Subsequent insertions and loads can disturb this order, and the DBA must schedule subsequent REORGs based on information derived from the statistics and performance information for the tables, which are maintained in the system catalog tables. To maintain these statistics, the UPDATE STATISTICS command is used. The statistics include the system catalog tables, as these are DB2 tables too.

- Backup and recovery

  It is vital that a DBA take image copies of the data and the indexes with the DBSU functions (UNLOAD or DATAUNLOAD) in order to recover data. But these functions require a working DBMS and an available database.

In order to provide recovery of a media failure or other damage, which prevents the database from starting, a full database backup has to be restored. The database offers two different kinds of backups to be taken. The ARCHIVE saves just the active data pages including the database directory, where the USERARCHIVE is an image of all the physical database files saved with operating system functions or other utilities.

The database changes are recorded on separate logfiles, which are to archived too, to get to a most current point in time during recovery.

Additional products and utilities are available to maintain and schedule backup tasks (control center) and to enhance backup and recovery functionality (data restore feature).

Using the data restore feature (DRF), a DBA can make a full copy and an incremental image copy of the database (userarchive). Since recovery requires a complete database restore, a full copy is always needed as the starting point. From the incremental copies taken after the full one, just the last one would be needed to recover to the latest backup, as the incremental copy always contains the delta to the latest full backup. The DRF product provides additional functions for a point-in-time recovery of single database objects (dbspaces/tables). The data image is extracted from the backup and then information from the archived 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 CREATE INDEX.

Using DB2 commands

Both the system administrator and the DBA use DB2 commands to monitor the subsystem. The operator command interface, DBSU, or Control Center provide you with a means to easily enter these commands. Operator commands can be entered from the system console or via ISQL interactively. DBSU provides the capability to issue commands from a batch job. Control Center allows you to schedule commands and certain actions to execute these while the system is unattended.

Several display commands provide the status of the database and processing. For example, with 'SHOW ACTIVE' you can display all active tasks currently under processing. Other commands provide lock status or log information. A combination of several SHOW commands is the 'SHOW SYSTEM' display command.

In some shops, DBAs are responsible for binds, although these are usually done by programmers as part of the compile job. The BIND process is needed to create access packages. The packages created for each application within the database using bind or preprocess can be saved, restored, or moved between databases using the DBSU UNLOAD / RELOAD PACKAGE commands.

12.8 What is SQL?

Structured Query Language (SQL) is not a full programming language, but it is necessary to access and manipulate data in a DB2 database. SQL is a 4GL non-procedural language that was developed in the mid-1970s to use with DB2. It is used to specify what information a user needs without having to know how to retrieve it. DB2 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 result table.

12.8.1 Using SQL and ISQL on VSE

SQL can either be used dynamically with an interpretive program like ISQL, or it can be imbedded and compiled or assembled in a host language (such as COBOL, PL/I, or Assembler).

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

12.8.2 Connecting to the application server in different environments

IBM DB2 server for VSE & VM is a relational database management system that supports both production and interactive environments.

DB2 Server for VSE users can connect to the application server in the following environments:

- CICS online environment

  In a CICS online environment, online users can connect to the application server implicitly and explicitly. If online users do not explicitly issue a CONNECT statement specifying the authorization ID and the password, then the first time they try to process an SQL statement, the CICS user is connected to the default application server implicitly.

- Batch/interactive environment

  In a VSE batch or VSE/ICCF environment, an explicit CONNECT must be the first statement entered by the batch user to access the application server.

- ISQL environment

  When CICS users start ISQL, they are prompted for a user ID, password, and target database. If the user enters the user ID and password only, ISQL does an explicit CONNECT to the default target database for the user. If the user does not enter a user ID, password, or target database, ISQL does a CONNECT to the default target database as a default user ID for the user. This defaulting is called an implicit CONNECT. If the ISQL user enters a target database only, a CONNECT would be made to that target database using a default user ID. If the user enters the user ID, password, and target database, ISQL does an explicit CONNECT to the target database.

  In the ISQL environment, you can access any of the application servers connected with the online resource adapter. If the online resource adapter is not connected to an application server, you cannot access the ISQL environment.

Switching to another application server

After connecting to an application server, a VSE user can switch to another one by issuing an SQL CONNECT statement. The switch occurs between logical units of work.

12.9 Application programming with DB2 Server for VSE

Applications are using SQL to interface with a DBMS. SQL can either be used dynamically with an interpretive program like ISQL, or it can be imbedded and compiled or assembled in a host language (such COBOL, PL/I, or Assembler).

Application development usage of the DB2 Server for VSE system includes a large amount of data design, application coding, and testing, as shown in Figure 12-4. The program products required to support application development on the DB2 Server for VSE system vary depending on the type of application being developed and the DB2 Server for VSE facilities to be used.

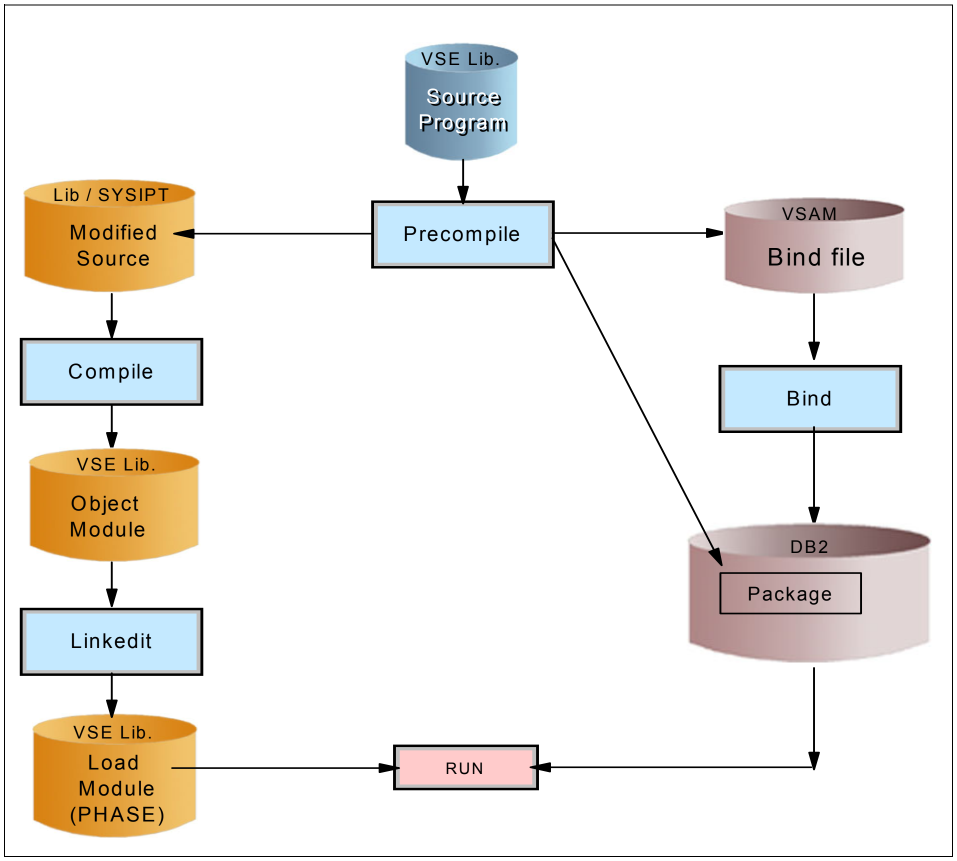


Figure 12-4 Program preparation flow

DB2 in VSE can be accessed from VSE subsystems or self-written applications, using:

- Programming languages

  For development of programmed applications you would need one or more of the PL/I, COBOL, C, Fortran, or Assembler language products.

- VSE/ICCF

  You can install VSE/ICCF (or an equivalent) to support an interactive application development capability.

- CICS subsystem

  The CICS subsystem is required to support development of CICS transactions or use of the ISQL facilities for application development.

- VSE/POWER

  The VSE/POWER program is required for the system printer or remote workstation printer report-writing support in ISQL. It is not required for report writing to CICS terminal printers. Only the VSE/POWER program provides multiple copy capability.

12.9.1 How do you write a DB2 application program?

To do this, SQL is embedded in the source code of a programming language. SQL can be used with the following programming languages: REXX, 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 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 ISQL. ISQL is actually an application program that accepts dynamic SQL statements. These are the SQL statements that you enter in the input file or at the console. Each time you use ISQL, the SQL can change.

The application development steps, as described in Chapter 8, “Designing and developing applications for z/VSE” on page 195, are used also for applications accessing DBMS. The difference is the type of access to the data.

Compared to other resources (for example, VSAM files), the database is accessed and manipulated using SQL. These static or dynamic SQL statements are not understood by the language compilers. This is why you need to precompile or preprocess the program.

The pre-processor has two major tasks:

- It replaces the SQL statements with a call or branch to a part of code that is linked to the program from the DB2 library during the linkedit step. These calls or branch statements are now understood by the language compiler at compile time. And it puts the SQL statement in a data structure that is to be handled and referenced during program execution time.
- It passes the data structures containing all of the SQL statements to the database server to let the server check the syntax of the statements and evaluate whether the referenced resources (tables, views, and so on) are available in the database. The database server then parses the statements and evaluates the best access path to be used at execution time (using an index or a dbspace scan). The statement itself and the access path are stored in an internal structure with the database server as a so-called package or access module.

Due to the first point, there is a pre-processor available for each programming language.

The precompilation, compilation, and linkedit steps to generate the executable form of a program are executed in a batch process.

First, the batch job calls the pre-processor and puts the source deck on the reader (SYSIN). The source code is processed and punched to an intermediate location (data set). From there the compiler reads the modified source when it is called in the next step. The output of the compile step is written to the library or intermediate storage as object code (program_name.OBJ). The finale step to generate the executable program (PHASE) is the linkage editor. The linkedit process combines your object deck with other required objects from their libraries (such as DB2 code), resolves symbolic references, and either stores the phase into the library or holds it in storage for a single execution (such as a test).

In case of CICS applications, there is one additional precompile step necessary to resolve the CICS-specific commands, which cannot be understood by the language compiler directly, the same as the SQL statements.

Now the program is ready to be used and can be called as a CICS transaction (if compiled for use with CICS) or in a batch job. Or it may be used by other applications.

12.10 What is DL/I?

Data Language/I (DL/I) has been developed from Information Management System (IMS), which is the hierarchical database management system on the z/OS platform.

Created in 1969, IMS was implemented on the basis of a hierarchical database model. This concept introduced the idea to separate application programs from the physical structure and storage of the data. It reduces data redundancy by maintaining only one copy of the data.

The term *DL/I* is used for the hierarchical structure of the database, the access method (language interface), and (on the z/VSE platform) for the product itself. It is also to a large extent compatible to IMS, but does not offer the same range of functions. One could see DL/I as a subset of the Database Manager component of IMS. For online transaction processing DL/I applications can be run on the CICS platform.

The DL/I databases are organized internally using different DL/I access methods, for example, Hierarchical Direct Access Method (HDAM) or Hierarchical Indexed Direct Access Method (HIDAM). The database data is usually stored in a VSAM data set, though there are database organizations that can also be implemented as sequential disk or tape file. DL/I data can be accessed directly or sequentially, through primary or secondary indexes, and it is possible to combine DL/I databases through logical relationships.

You write a DL/I program in much the same way that you write any other program. You can use COBOL, PL/I, C, Assembler language, or (on a workstation) Java to write DL/I application programs. DL/I offers two types of programming interfaces: a standard call-level interface and a High Level Programming Interface (HLPI), which can be compared to the CICS command-level interface.

More information about how to write a DL/I application in Java can be found in the IBM publication *z/VSE e-business Connectors User's Guide*.

12.10.1 The DL/I hierarchical database model

DL/I uses a hierarchical model as the basic method for storing 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, with segments at lower levels depending on segments at higher levels for their context. A segment usually consists of a group of related fields, and a field is a single piece of data containing an attribute. It can be used as a key for ordering the segments or as a qualifier for searching.

The hierarchical structure is determined by the designer of the database, based on the relationships between the segments and the access paths required by the applications. The different segments within a hierarchy make up a database record, and a DL/I database is the entire stored collection of all occurrences of one type of database record.

Figure 12-5 shows an example of a DL/I database with three levels.

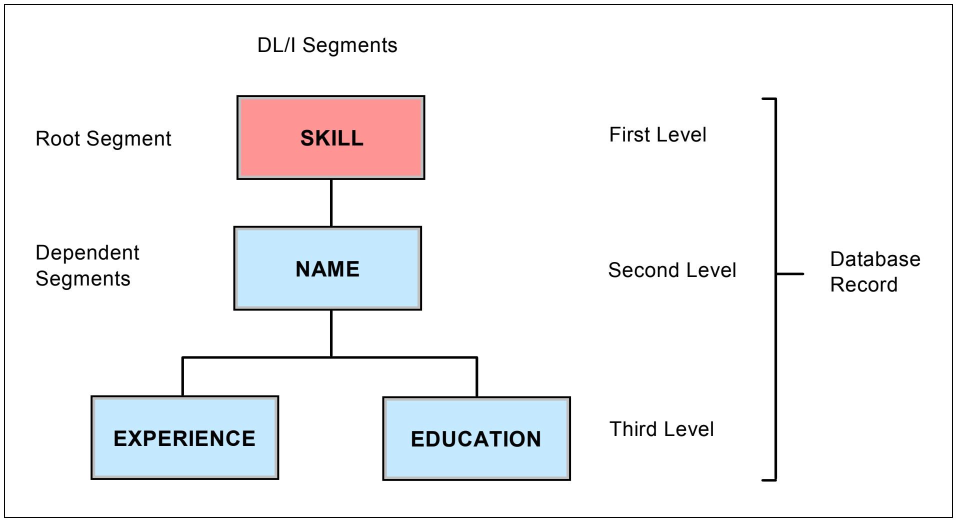


Figure 12-5 DL/I hierarchical structure - Sample Skills Inventory database

DL/I allows a maximum of 255 segment types and 15 segment levels to 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.

Segment relationship

The segment on the first level of the hierarchical data structure is called the root segment. One occurrence of a root segment could be considered as the representation of one database record.

Within a hierarchy the basic building element is the parent/child relationship between segments. Each parent can have many children, but each child only has one parent. Segments of the same type belonging to the same occurrence of a parent segment (and also the different occurrences of the root segment itself) are called twins. In Figure 12-6, the segments Adams, Jones, and Smith are twins, and they are children of Artist, their parent segment.

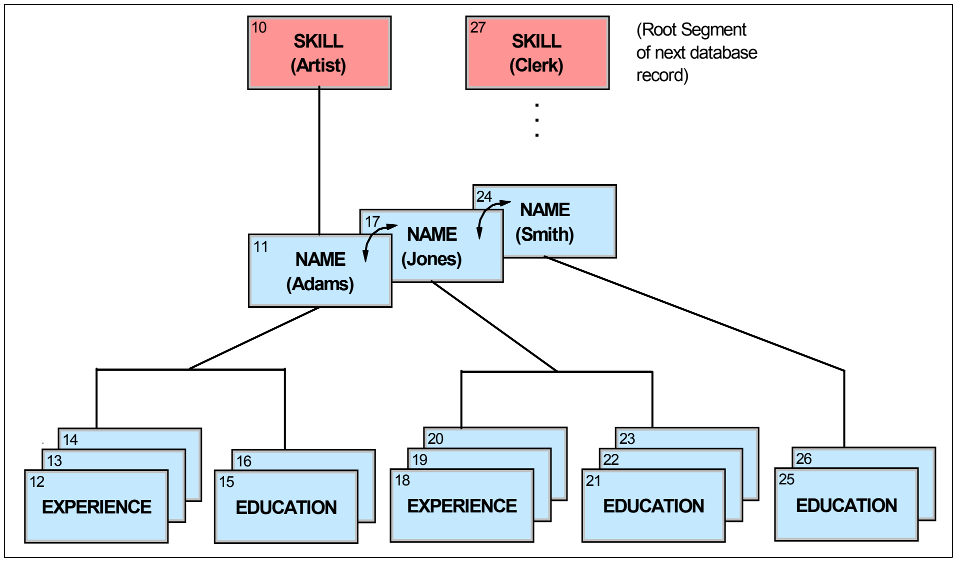


Figure 12-6 Expanded database record

The sequence of traversing the hierarchy is top to bottom, left to right, front to back (for twins including the root segments). When sequentially processing a database record all segments are included, so twins do have a place in hierarchic sequences. As already mentioned, key fields can be used to determine the order in which segments of the same type are stored and processed. Figure 12-6 illustrates the sequence of segments within a database record of the Skills Inventory database. It is important to know that, although shown in this example, a database record does not have to contain occurrences of all of the segment types that are possible in that database record.

Figure 12-6 describes the data structure of one database record as seen by the application program. It does not represent the physical storage of the data, which is of no concern to the application. The association between related segments within (and, for indexed or logically related databases, also between) the physical VSAM data sets of the databases are established through 4-byte pointers called Relative Byte Addresses (RBAs). For example, the link between segments Adams and Jones and Jones and Smith is implemented through a Physical Twin Forward (PTF) and Physical Twin Backward (PTB) pointer.

12.10.2 Defining a database

When the design of a database has been completed, it must be specified to DL/I by these steps. A description of the single steps follows:

- Database description generation (DBDGEN)
- Program Specification Block Generation (PSBGEN)
- Application control blocks generation (ACBGEN)
- Database space allocation

Database description generation (DBDGEN)

The DBDGEN process implements the hierarchical structure of the data by describing a real physical or (virtually constructed) logical database (physical or logical DBD) in a formal definition language. It applies to the system view of a database, in contrast to the application view, which is defined during PSBGEN and that may comprise only a part of the system view. *Physical* in this context means in a DL/I sense that we are considering a real database (compared to a logical database) and does not refer to the way that data are physically kept on a storage device.

A physical DBD describes the organizational characteristics of a database (for example, HDAM or HIDAM), all of its segments with their hierarchical relationships, and the properties of the database data set. It also allows the definition of secondary indexes when a different access path to the database record other than via the root key is wanted. A logical DBD (and the resulting logical database) can be built by combining physical DBDs, or parts of them, to a new hierarchical data structure without altering the original position of the segments involved. To application programs, this data structure appears as if it were a physical database.

A DBD consists of a set of DL/I macro instructions that are assembled and link-edited to a phase. Example 12-1 represents a physical DBD generation for the skills inventory database.

Example 12-1 DBD generation

```
// JOB DBDGEN - GENERATE DBD FOR SAMPLE DATABASE SKILINV
// OPTION CATAL,NODECK
// EXEC ASSEMBLY,SIZE=256K
         PRINT NOGEN                 no macro expansion printing
         DBD                                                          x
               NAME=SKILINV,         Data Base Description name      x
               ACCESS=HDAM,          Hierarchical Direct             x
               RMNAME=(DLZHDC10,     Randomizing routine phasename   x
               3,                    Root Anchor Points per block    x
               100,                  Root Addr. Area Hi relative blk x
               300)                  insert bytes limit for RAA
```

```
data set                                                   x
      DD1=SKHISAM,            DLBL file name              x
      DEVICE=CKD,             disk device type            x
      BLOCK=(2048),           VSAM Control Interval size  x
      SCAN=2                  # cylinders scan for isrt space
SEGM                                                      x
      NAME=SKILL,             segment name for skill      x
      PARENT=0,               it is a root segment        x
      BYTES=31                data length
FIELD                                                     x
      NAME=(TYPE,SEQ,U),      unique key field - skill type  x
      BYTES=21,               field length                x
      START=1,                where it starts in segment  x
      TYPE=C                  alphameric data
FIELD                                                     x
      NAME=STDCODE,           skill code                  x
      BYTES=10,               field length                x
      START=22,               where it starts in segment  x
      TYPE=C                  alphameric data
SEGM                                                      x
      NAME=NAME,              segment name for person     x
      PARENT=SKILL,           parent is skill segment     x
      BYTES=20
FIELD                                                     x
      NAME=(STDCLEVL,SEQ,U),  unique key field - skill level  x
      BYTES=20,                                           x
      START=1,                                            x
      TYPE=C
SEGM                                                      x
      NAME=EXPR,              segment for experience of personx
      PARENT=NAME,            parent is person segment    x
      BYTES=20
FIELD NAME=PREVJOB,           previous occupation         x
      BYTES=10,                                           x
      START=1,                                            x
      TYPE=C
FIELD NAME=CLASSIF,           classification              x
      BYTES=10,                                           x
      START=11,                                           x
      TYPE=C
SEGM                                                      x
      NAME=EDUC,              segment for education of person x
      PARENT=NAME,            parent is person segment    x
      BYTES=75
FIELD NAME=GRADLEVL,          grade level                 x
      BYTES=10,                                           x
      START=1,                                            x
      TYPE=C
FIELD NAME=SCHOOL,            school                      x
```

```
              BYTES=65,                                            x
              START=11,                                            x
              TYPE=C
        DBDGEN                   required to mark DBD end
        FINISH                   for source compat with IMS
        END
/*
// EXEC LNKEDT
/*
/&
```

Program specification block generation (PSBGEN)

The PSBGEN process applies to the application view of the data as required by a particular application program and is based on the previously defined system view from DBDGEN. The application view of a database is represented by a so-called Program Communication Block (PCB). The PCB describes for an underlying physical or logical DBD (if the database may be updated or ready only) which of its segments and fields are accessible and whether, for example, the database is to be processed via a secondary index. All PCBs an application may use comprise a PSB, which is the unit that is passed to the application program. Sometimes the application view is also referred to as *logical* view, independently from the terms *logical DBD* or *logical database*.

Like a DBD, each PCB included in a PSB is defined in a formal definition language through a set of DL/I macro instructions. The PSB is created by assembling the PCB definitions as one source and link-editing them to a phase. Example 12-2 represents a PSB generation with one PCB for the skills inventory database.

Example 12-2 PSB generation

```
// JOB PSBGEN - GENERATE PSB FOR SAMPLE DATABASE SKILINV
// OPTION CATAL,NODECK
// EXEC ASSEMBLY,SIZE=256K
        PRINT NOGEN              no macro expansion printing
        PCB                                                        x
              TYPE=DB,           required                          x
              DBDNAME=SKILINV,   from DBD macro in DBD assembly    x
              PROCOPT=AP,        read and write access             x
              KEYLEN=50,         concatenated key length           x
              POS=S              single positioning (default)
*                                the longest concatenated key in
*                                the Skills Database is 41.
*                                50 leaves room for expansion.
        SENSEG                                                     x
              NAME=SKILL,        using the same names as found     x
              PARENT=0           in the SEGM macro in the DBD
```

```
         SENSEG                                                    x
               NAME=NAME,                                          x
               PARENT=SKILL
         SENSEG                                                    x
               NAME=EXPR,                                          x
               PARENT=NAME
         SENFLD NAME=PREVJOB          access to this field only
         SENSEG                                                    x
               NAME=EDUC,                                          x
               PARENT=NAME
         PSBGEN                                                    x
               LANG=COBOL,            application prog is COBOL    x
               PSBNAME=SKILPSB        Program Specification Blk name
         END
/*
// EXEC LNKEDT
/*
/&
```

Application control blocks generation (ACBGEN)

The system and application views from DBDGEN and PSBGEN with their associated phases are of an intermediate format only. They have to be tied together during the ACBGEN process to create the Application Control Block (ACB) phase image for the DBDs and PSBs. This is the final format necessary for program execution.

Example 12-3 shows how the ACB format for the DBD and PSB of the skills inventory database is built by the application control blocks creation and maintenance utility. After having been processed through this utility, a DBD is usually called DMB (data management block).

Example 12-3 ACB generation

```
// JOB ACBGEN - GENERATE ACBs FOR DBD SKILINV with PSB SKILPSB
// OPTION CATAL,NODECK
// EXEC DLZUACBO,SIZE=200K
 BUILD PSB=(SKILPSB),                                              x
               OUT=LINK,              destination is syslnk        x
               DMB=YES                create also DMBs
/*
// EXEC LNKEDT
/*
/&
```

Figure 12-7 summarizes the complete process of the stepwise DL/I DBD and PSB generation.

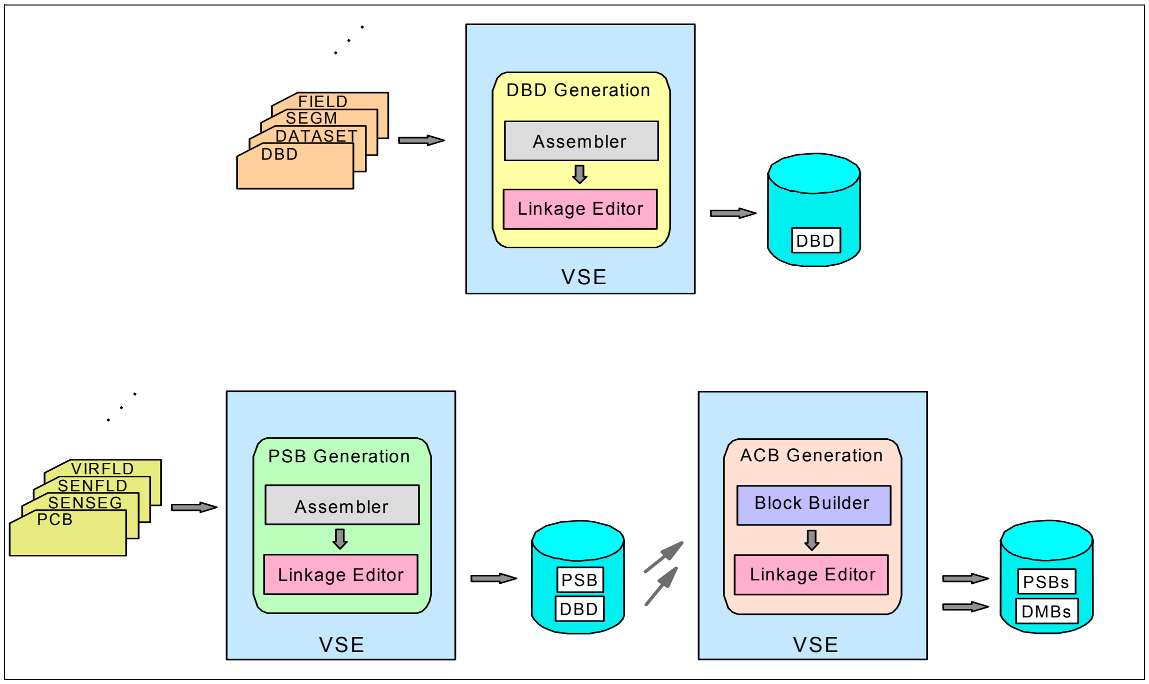


Figure 12-7 DL/I control block generation - summary

Database space allocation

Before a database can be loaded and processed, it must be defined to VSAM using the Access Method Services utility functions. Example 12-4 shows how the data set for the skills inventory database can be allocated.

Example 12-4 Define cluster

```
// JOB DEFCLUS - DEFINE VSAM data set FOR DATABASE SKILINV
// EXEC IDCAMS,SIZE=AUTO
         DEFINE CLUSTER (                           -
                        NAME(SAMPLE.SKILLS)         -
                        NONINDEXED  )               -
                  DATA  (                           -
                        NAME(SKILL)                 -
                        VOLUMES(SYSWRK)             -
                        CYL(1 1)                    -
                        CNVSZ(2048)                 -
                        RECSZ(2038 2038))
/*
/&
```

Online nucleus generation

If you want to run DL/I applications in an online environment, you also have to make definitions to the CICS subsystem. This comprises a CICS and a DL/I part. The CICS part essentially means that you have to adapt different CICS tables and the CICS CSD file to allow the addition of DL/I. These tasks are not further described here. For the DL/I part you have to generate a so-called DL/I online nucleus.

The DL/I online nucleus controls DL/I operation in a CICS subsystem. It consists of the particular DL/I online routines and control blocks, a PSB directory, the definition of the buffer pool and all DL/I online application programs, plus various other online-related information. For each DL/I online program there is a specification of which PSBs it is authorized to use (schedule).

You generate the DL/I online nucleus by assembling different types of the DLZACT macro and link-editing them to the nucleus phase. Example 12-5 shows a nucleus generation that includes an entry for a program processing the skills inventory database and the allocation of a buffer subpool for that database.

Example 12-5 Online nucleus generation

```
// JOB NUCGEN - GENERATE DL/I ONLINE NUCLEUS '1B'
// OPTION CATAL,NODECK
// EXEC ASSEMBLY,SIZE=256K
       PRINT NOGEN
       DLZACT TYPE=INITIAL,SUFFIX=1B
```

```
*
       DLZACT TYPE=CONFIG,MAXTASK=5,CMAXTSK=5,BFRPOOL=1,PI=YES,        x
               HSMODE=ANY,PSBLOC=ANY,SVA=YES
*
       DLZACT TYPE=PROGRAM,PGMNAME=SKILPROG,PSBNAME=SKILPSB
*
       DLZACT TYPE=BUFFER,HDBFR=(20,SKILINV)
*
       DLZACT TYPE=FINAL
       END
/*
       ENTRY DLZNUC
// EXEC LNKEDT
/*
/&
```

12.10.3 Processing a database

Once all implementation steps are accomplished, the database is ready for use. You can now load and then process it by an application program through one of the two types of programming interfaces available with DL/I. Example 12-6 shows how this may look when the DL/I HLPI interface is used in a COBOL online program. The represented syntax samples and their results are based on the skills inventory database and the expanded database record of Figure 12-6 on page 318.

Example 12-6 DL/I calls in HLPI style - COBOL

```
           EXEC DLI SCHEDULE PSB(SKILPSB) END-EXEC.
...
           MOVE 'ARTIST' TO SKILL-TYPE.
           EXEC DLI GET UNIQUE USING PCB (1)
                    SEGMENT (SKILL) WHERE (TYPE=SKILL-TYPE)
                    FIELDLENGTH (21)
                    SEGMENT (NAME) SEGLENGTH (20) INTO (IOAREA)
                    END-EXEC.
...
           EXEC DLI GET NEXT USING PCB (1)
                    INTO (IOAREA) END-EXEC.
```

Before a program can access a DL/I database it must first establish a connection to it. For online programs (that is, online tasks) this is achieved through a scheduling call, which is also the first DL/I call shown in Example 12-6. The scheduling call builds a link from the application program to the requested PSB and the PCBs contained in it, and also to all required DMBs. On successful completion of a scheduling call the online program can submit regular DL/I database calls. For batch programs the connection to the PSB is built in another way, but there is no difference to online programs in respect to the database calls.

In the second example a Get Unique call retrieves segment Adams into a field named ioarea. The Get Unique call-type allows it to access DL/I segments directly.

The subsequent Get Next call is not qualified by a segment type. Therefore DL/I follows the hierarchical path and returns the EXPR (experience) segment with number 12 in the ioarea field. The Get Next call-type can be used to read a DL/I database sequentially.

12.10.4 DL/I environments

DL/I applications can be run in different environments. Initially (and this is still the most common practice), you could run them as a VSE batch job, an MPS batch job, or as an online transaction under the control of the CICS subsystem. Later, in order to meet new business requirements, network-based options have been added. These are new programming services offered in many cases by other products, but also from DL/I itself, allowing you to make DL/I data centrally stored on the VSE host also available to the user on the Internet or at a remote workstation.

Normal batch

In this environment the DL/I batch initialization phase (invoked as a VSE batch job step) loads all DL/I resources like processing routines, PSB and DMB control blocks, the buffer pool, and the DL/I batch application into a VSE batch partition and then passes control to the application. The DL/I databases are accessed in the VSE batch partition. This implies that—because of the required VSAM Shareoptions—if you are processing a database through a normal VSE batch job in update mode, no other partition (for example, no other CICS/DLI online partition) will be allowed to access this database in update mode at the same time, while multiple concurrent read access from other partitions would be granted. And vice-versa: When a CICS/DLI online partition is processing a database in update mode, only read access will be possible for normal DL/I batch jobs running in other partitions.

CICS online

The CICS Transaction management system is the central online platform of the z/VSE and also the most important environment to access DL/I data. It is described in more detail in Chapter 11, “Transaction management systems on z/VSE” on page 263. In the CICS environment the users normally start an application by submitting a transaction request from a 3270-type terminal and then work interactively with the application. But there are now programming services allowing the use of CICS applications also, for example, from batch programs, remote PC workstations, or the Internet. This is further described in “Network based” on page 327.

During the CICS/DLI initialization stage all DL/I resources (same types as in batch plus the DL/I online nucleus) are allocated in CICS storage. As DL/I online applications are basically CICS online applications, they are loaded, invoked, and executed under the control of the CICS subsystem in the same way as regular CICS applications. The DL/I calls are routed to the DL/I processing routines. Running DL/I applications in the CICS online environment allows concurrent access and update of DL/I data for multiple users and programs (batch, online, local, and remote) through CICS tasks, while preserving the integrity of that data.

MPS batch

In this environment the DL/I application is running in a VSE batch partition while the database accesses are made in a CICS online partition. The DL/I MPS (Multiple Partition Support) facility (invoked as a VSE batch job step) establishes a connection to a CICS online system, where a particular MPS mirror program is started, loads the DL/I batch application, and passes control to it. After that all DL/I calls issued by the batch application are directed via the MPS facility to the online mirror program, which enters them to the DL/I online system and in turn sends back any data or result information to the MPS facility, which passes them on to the batch application.

MPS is transparent to the application (that is, a DL/I batch program needs not be changed to run in this environment). Using MPS, multiple batch and online applications running in different partitions can concurrently process the same DL/I databases in update mode, because all database accesses are made by a CICS/DLI online system.

Network based

You can now also access DL/I databases from a workstation platform, which communicates with the VSE host through a connectivity service on a 2-tier or 3-tier environment. This allows you to write new applications on a client workstation employing the new e-business and Web technologies while keeping and accessing the DL/I data centralized on the z/VSE server system. This means that DL/I data can also be made available to users on the Internet.

In this environment you can access the DL/I host data in two different ways:

- A Java client program can use a new Java DL/I interface, allowing it to submit regular DL/I calls. These calls are then routed across the network through the

VSE Java-Based Connector and executed by a CICS mirror program on the VSE host.

- The client makes a request for a resource (for example, a HTTP request from a Web browser), which, after having been processed by the components and programs of the network and routed to the VSE host, leads to the invocation of a CICS application that accesses the DL/I database. The middleware and the associated software on the host playing together and providing the connectivity service determine which interfaces can be used to transform the initial request and transmit it to the VSE system in order to invoke the CICS application. The following products and their interfaces are supported:
  - CICS Web support
  - CICS Transaction Gateway
  - DB2 Connect
  - MQSeries

In all scenarios, the DL/I data are retrieved or written using standard DL/I calls on the CICS subsystem of the VSE host. The obtained data and feedback information are then sent back to the client on the same connectivity path, on which the request had first been sent.

More information about the different z/VSE connectivity options and components involved can be found in Chapter 13, “z/VSE connectors” on page 333, and the IBM publication *z/VSE e-business Connectors User's Guide*.

Figure 12-8 illustrates the different environments that are supported by DL/I.

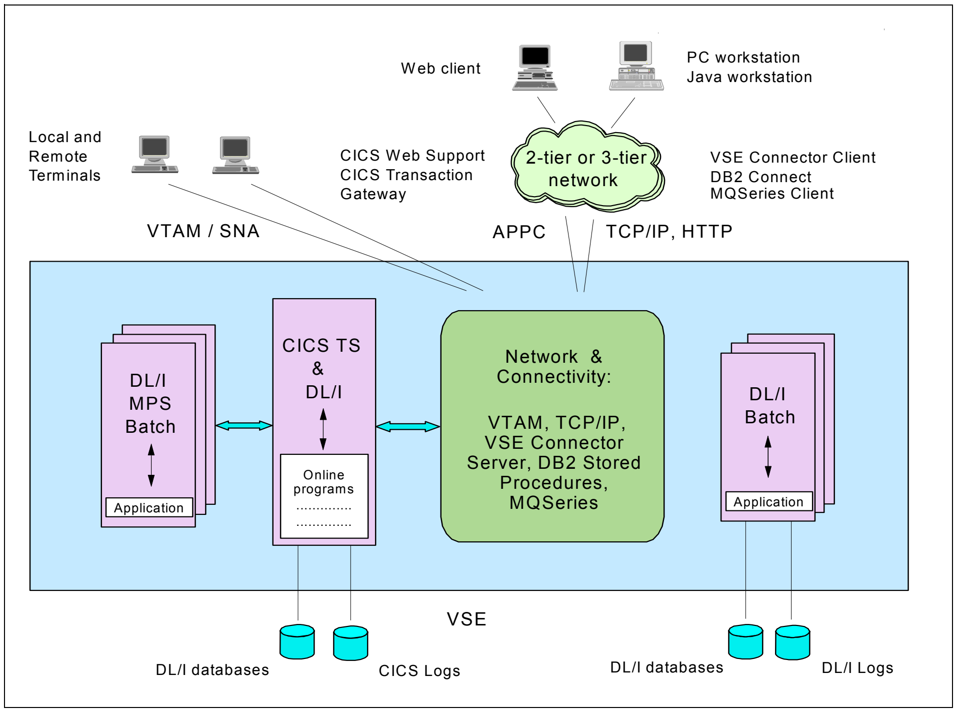


Figure 12-8 DL/I environments

Data integrity and recovery

Locking mechanisms are used to ensure data integrity in a CICS/DLI online environment when multiple tasks are concurrently accessing the same database.

All modifications to a DL/I database used in one of the environments described above can be recorded on either the log data set of the CICS subsystem or a DL/I log data set. The log data on these data sets allow for a rollback of uncommitted (incomplete) database updates, when an abnormal termination of an application or the system has occurred. If necessary, they can also be printed and used together with the DL/I recovery utilities to completely rebuild a database.

DL/I utilities

DL/I supplies utilities to implement the system design of a database and its application views. There are also utilities to resolve logical and secondary index relationships during database load as well as backup and recovery utilities for securing and maintaining the databases. And there are utilities for tuning the databases by reorganizing and restructuring them and facilities to provide diagnostic and statistic (performance) information.

12.11 Summary

Data can be stored in a flat file, but this usually results in lots of duplication, which may result in inconsistent data. Therefore, it is better to create central databases, which can be accessed (for reading and changing) from various places. The handling of consistency, security, and so on, is done by the database management system. The user/developer does not need to worry about it.

IBM DB2 implements the relational principles, where the fundamental structure is the table with columns and rows. The language to access the database is the Structured Query Language (SQL).

DL/I uses a hierarchical model as the basic method for storing data. 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.

Key terms in this chapter		
Database	DL/I	Dbspace
ISQL	View	DBMS
Root	Dbextent	Storage pool
Database administrator (DBA)	Sequence	DB2

12.12 Questions for review

To help test your understanding of the material in this chapter, complete the following questions:

1. What DB2 objects define a physical storage area?
2. Where is table placed?
3. What are some of the problems with the following SQL statement?

```
SELECT *
FROM PAYROLL;
```

4. What category of SQL would you use to define objects to DB2?
5. How does the precompiler find an SQL statement in a program?
6. How does a load module get put back together with the SQL statements?
7. What is a stored procedure?
8. What are some of the responsibilities of a system administrator?
9. What are some of the responsibilities of a database administrator (DBA)?
10. What are some of the ways that security is handled by DB2?
11. What is the database structure of DL/I? Describe it.

13

z/VSE connectors

Objective: Connectors are the building blocks to connect different applications on heterogeneous platforms and operating systems. They are the link from mainframe operating systems to open standards. This chapter mainly covers the native z/VSE connectors (that is, connectors that are unique to z/VSE.**)**

After completing this chapter, you will be able to:

- List the z/VSE connectors and their functionality.
- Describe scenarios where they are used.
- Explain the function of a resource adapter.

13.1 What are z/VSE connectors?

On today's computer systems many standard communication methods like Telnet, FTP, HTTP, SCP, SSH, SMTP, and so on are usually available. Although VSE has some of these communication methods, additional functionality is needed to handle VSE-specific data:

Telnet	VSE does not use standard line mode telnet (VT100). Instead it uses the screen-based Telnet mode tn3270.
FTP	FTP is usually available on VSE, but not all VSE resources can be transferred using a sequential FTP. Some VSE resources require special handling like variable record length, field-based translation, and so on.

Most modern computer systems have one or more file systems that store files. In that case, files are known as streams of bytes. Only the user of a file knows how to read the data. Text files use special characters (for example, CR LF) to separate the lines.

On VSE most files work on a record basis (for example, VSAM, SAM). This means that each file consists of records containing user data. Records can have a fixed or variable length. Multiple records can be combined into blocks. Applications usually read and write complete records, not single bytes.

Although it is possible to map record-based files to sequential byte stream oriented communication mechanisms, you probably loose some information when doing so (for example, the record length information). In VSAM it is very common to have textual data mixed with binary numbers, packed decimals, and other data types stored in a record. When transferring these records to another computer, you must translate the data into the receiver's data format and codepage (ASCII / EBCDIC). Because a record can contain both textual and binary data, you cannot translate the whole record to ASCII (as, for example, FTP does), this would destroy the binary data. However, if you transfer the data in binary, the receiver will not be able to read the textual data, because it is in EBCDIC.

Therefore the VSE connectors provide additional communication methods that know VSE specifics and are able to make them available to the communication partner. Besides others, they provide mechanisms to translate transferred VSAM data on a field basis (for details see 13.9, "Mapping VSE/VSAM data to a relational structure" on page 351).

As a consequence, VSE provides a framework of connectors for accessing VSE data from various client platforms over TCP/IP, as well as for accessing remote data from VSE applications.

These are the current z/VSE e-business connectors:

- The Java-based connector, which consists of a client-part (the VSE connector client) and a server-part (the VSE connector server)
- The VSAM redirector connector, which consists of a client-part (the VSAM redirector client) and a server-part (the VSAM redirector server)
- The VSE script connector, which consists of a client-part (any user-written Java or non-Java application called the VSE script client) and a server-part (IBM-supplied VSE script server, which is a Java application)
- The Simple Object Access Protocol (SOAP) connector
- The DB2-based connector, including its stored procedure support

The following chapter explains the z/VSE connectors in detail.

13.2 Overview of the Java-based connector

The Java-based connector consists of a client-part (the VSE connector client) and a server-part (the VSE connector server). It allows you to access VSE resources from a remote system. You can, for example, access VSAM data on a record basis from a Java-program that is running on a PC or workstation, as a standalone application, or (for example) as a servlet[1] in a Web application server.

13.2.1 Overview of the VSE connector client

You can install the VSE connector client on most Java-enabled platforms. It consists of these parts:

- A main file VSEConnector.jar that contains the VSE Java Beans class library and the VSAM JDBC™ driver. This provides a Java programming interface for communicating with VSE/VSAM, VSE/Librarian, VSE/POWER, VSE/ICCF, and operator console, on the z/VSE host.
- Three additional jar files (ibmjsse.jar, cci.jar, and ibmpkcs.jar) that provide support for a Web Application Server in a J2EE environment, like WebSphere and SSL.
- A set of samples, including Java source code, that show you how to write Java programs that are based upon the use of VSE Java Beans.

[1] Servlets are highly portable Java programs that run in the environment of a Web application server, like IBM WebSphere. Their purpose is to create dynamic Web pages, for example, Common Gateway Interface (CGI) programs. But unlike CGI programs, servlets do not run as a separate program. As they are written in Java, they have the advantage that they are not platform dependent. Servlets can run on any platform that supports a Web application server.

- Online documentation (a set of HTML pages) describing the various concepts and samples.

The VSE connector client provides two interfaces to access VSE resources:

VSE Java Beans	The VSE Java Beans provide a Java programming interface that allows you to work with VSE resources. It gives access to VSAM, LIBR, DL/1, POWER, Operator Console, and ICCF.
VSAM JDBC driver	The VSAM JDBC driver provides a standard interface that allows you to issue SQL statements against non-relational VSAM data.

Both interfaces can also be used from within a J2EE Web Application Server environment. The VSE Java Beans library can be deployed into an application server as J2C Resource Adapter. The VSAM JDBC Driver can be deployed as a JDBC Data Source.

Figure 13-1 shows how a Java application running on a workstation can access VSE data by using the VSE Java beans class library, which connects to the VSE Connector Server on VSE via TCP/IP. There is a proprietary communication protocol used internally by the VSE connector client and server.

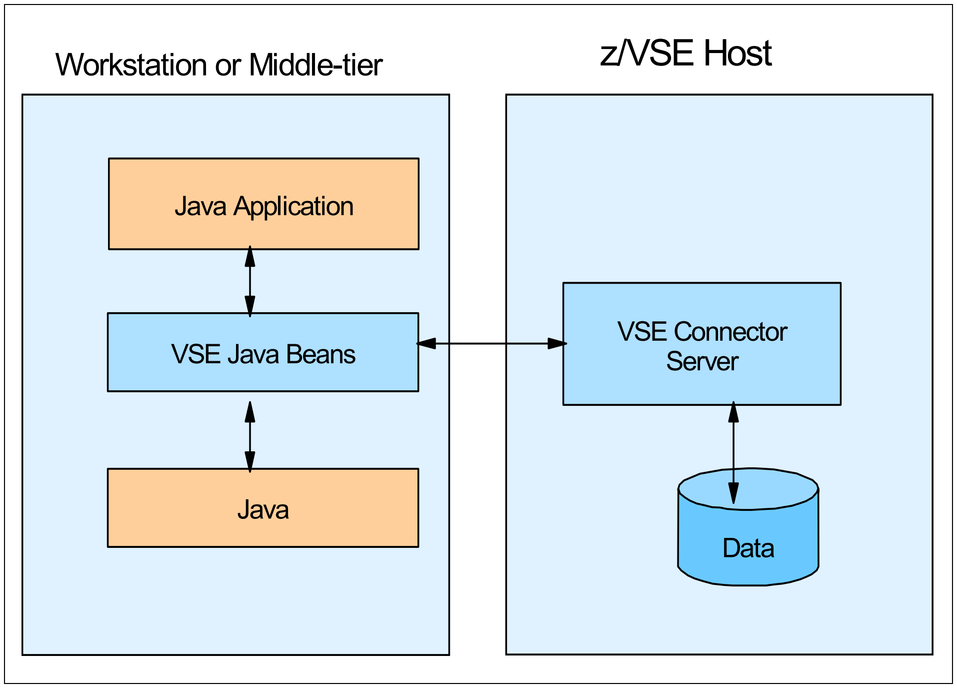


Figure 13-1 Overview of Java-based connector

13.2.2 Overview of the VSE connector server

The VSE connector server is part of the z/VSE host, (that is, it is already installed and preconfigured and it must simply be started in order to become operational). It then provides a TCP/IP socket listener that can handle multiple clients. The server is a VSE batch application that runs by default in dynamic class R.

Java programs running on Web clients or the middle-tier use the VSE Java Beans to build connections to the VSE connector server running on the z/VSE host.

To start the VSE connector server, you use the job STARTVCS, which is placed in the POWER reader queue during the installation of the z/VSE base operating system. When started, the VSE connector server listens to incoming TCP/IP traffic on port 2893 by default.

Although the VSE connector server is pre-configured and should require no major configuration effort by yourself, it is, however, possible to modify various configuration members to specify:

- The VSE/AF libraries that can be accessed by the VSE connector server. You may extend or restrict this list according to your needs.
- Plug-ins to be loaded at VSE connector server startup. You can extend the Java-based connector by specifying your own host-side plug-ins.
- Which users or groups of users are allowed to log on to the VSE connector server.
- Whether SSL will be used to communicate with clients.

13.3 Overview of the VSAM redirector connector

The VSAM redirector connector enables VSE programs to access data on remote systems in real-time. Using the VSAM Redirector connector:

- VSAM data can be migrated to other file systems or databases.
- VSAM data can be synchronized with data on remote systems.
- VSE programs can work transparently with data on other file systems or databases.

The VSAM redirector connector consists of:

- The VSAM redirector client (installed on your z/VSE host)
- A VSAM redirector server, which can be installed on most Java platforms

The VSAM redirector server provides an interface that allows users to develop their own handler. The handler is responsible for converting the VSAM requests into database or file system specific operations. IBM provides sample handlers that show how such a handler works.

On VSE you define in a configuration table which VSAM clusters are redirected and to which server. You also specify which handler is responsible for converting the requests.

Figure 13-2 gives an overview of the involved platforms and system components.

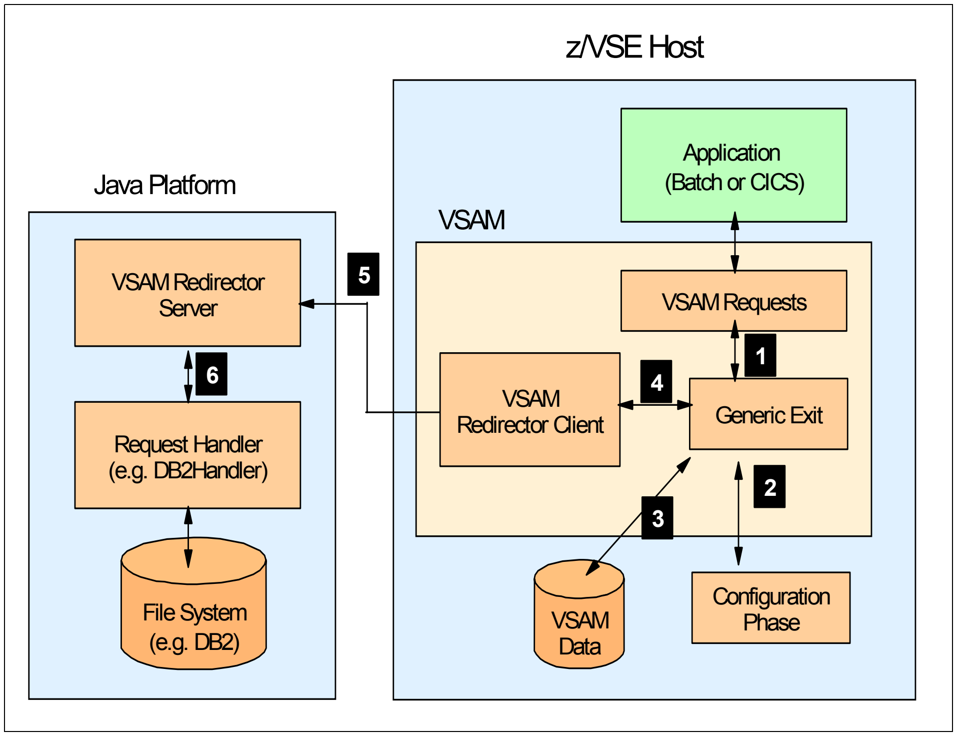


Figure 13-2 Overview of VSAM redirector connector

The general processing shown in Figure 13-2 is as follows:

1. An application running on the z/VSE host issues a VSAM command (for example, to open a VSAM file). The request is passed to the generic exit.
2. The generic exit checks whether the VSAM file has been set up to be redirected. To do so, it checks the configuration phase.
3. If the VSAM file has not been redirected to another Java platform (and is therefore still stored as a VSAM record on the z/VSE host), the generic exit returns and indicates that the VSAM file has not been redirected. It also

indicates that the generic exit should not be called again for any request against this VSAM file. Normal VSAM processing then continues.

4. If the VSAM file has been redirected to another Java platform, the generic exit calls the VSAM redirector client.
5. The VSAM redirector client establishes a connection to the VSAM redirector server running on the Java platform, and forwards the VSAM file request, together with any data (such as a VSAM record contents) to the VSAM redirector server.
6. The VSAM redirector server uses the request handler that is specified in the configuration phase, to perform access to the target file system or database. In Figure 13-2 on page 338, the specified request handler is DB2Handler (which is supplied by IBM during the installation of the VSAM).

A Java handler provides access to the specific file system or database on the remote system. For example, you can migrate your VSAM data into DB2 tables residing on a remote system, and your z/VSE host programs will then work with this data without requiring any changes.

The VSAM redirector connector handles requests to VSAM data sets and redirects them to a different:

- Java platform (for example, Linux and Windows)
- File system (for example, DB2 or flat files)

Your existing z/VSE host programs are:

- Written in any language (C, COBOL, PL/I, ASSEMBLER)
- Batch or CICS programs

The host programs can work with migrated VSAM data without the need to amend and recompile these z/VSE host programs.

13.4 Overview of the VSE script connector

As described in 13.2, “Overview of the Java-based connector” on page 335, VSE Java beans provide direct access to the z/VSE host from any kind of Java program. In addition, you can use the VSE script connector to access z/VSE host data from non-Java platforms. This is the main advantage of using the VSE script connector (although it can also be used to access z/VSE host data from most Java platforms).

The VSE Script connector uses the Java-based connector to access z/VSE resources. It consists of:

- A VSE Script client running on a Java or non-Java platform, and that can be:
  - A user-written Java application (for example, a Web service)
  - A user-written non-Java application (for example, a Windows C-program, a Windows CGI-program, or a COBOL application)
  - An office product, such as a word processing or spreadsheet program (for example, Lotus 1-2-3 or Microsoft® Excel®) where, for example, a Visual Basic® script is used to call a VSE Script
- The VSE script server running on the middle-tier of a 3-tier environment, which interprets and executes VSE Script files.
- Online documentation, including a programming reference manual.

The VSE script connector, as shown in Figure 13-3, works in this general way:

1. The VSE script client, which can be either an application running on a workstation or a VSE application, establishes a TCP/IP connection to the VSE script server on the middle tier. The VSE script client then calls a VSE script by sending the following to the VSE script server running on a middle-tier and Java-enabled platform:
   - The file name of the VSE script
   - The parameters belonging to the VSE script

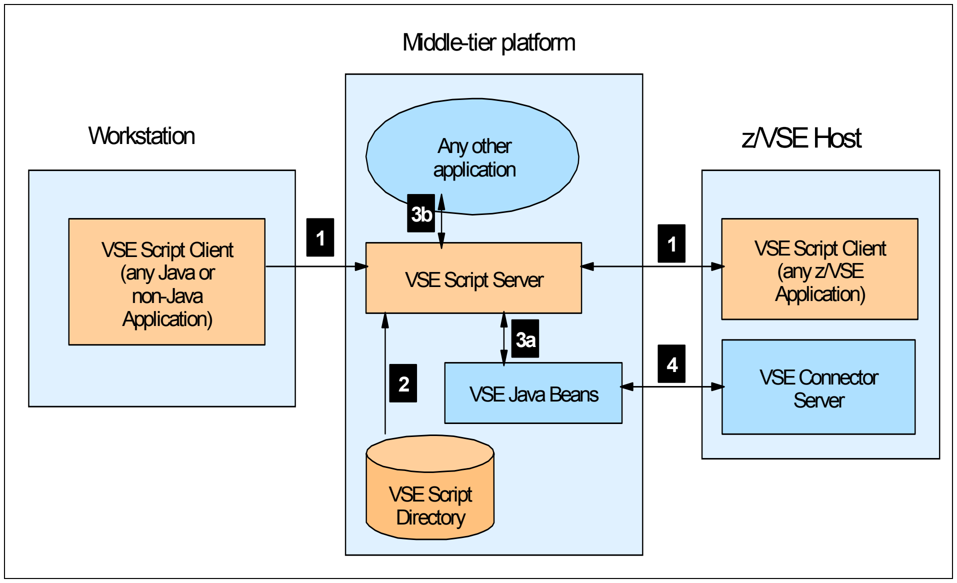


Figure 13-3 Overview of the VSE script connector

2. The VSE script server reads the VSE script file from the VSE script directory and starts to interpret and translate the VSE script file statements. Each VSE script file is written using the VSE script language.
3. This step has two options:
   a. If the VSE script file wishes to obtain data from the z/VSE host, the VSE Java Beans requests are executed by the VSE Java Beans.
   b. The VSE script file can also obtain data from any other application running on the middle tier by calling the application. This is especially useful if you plan to run the VSE script client on the z/VSE host.
4. The VSE Java Beans communicate with the VSE connector server running on the z/VSE host, which performs the request for functions and data. (This step only applies when step 3a was followed.) The data that was obtained is then converted to a format that the VSE script client can use, and is returned to the VSE script client

You can use the VSE script connector to execute remote applications. For example:

- If you follow step 3a, VSE script clients can execute jobs on the z/VSE host.
- If you follow step 3b, VSE batch jobs or programs can execute applications or shell scripts on the middle-tier.

13.5 z/VSE support for Web services and SOAP

SOAP is a standard, XML-based, industry-wide protocol that allows applications to exchange information over the Internet, for example, via HTTP. XML is a universal format that is used for structured documents and data on the Web. It is independent of both the Web client's operating-system platform and the programming language used. HTTP is supported by all Internet Web browsers and servers.

SOAP combines the benefits of both XML and HTTP into one standard application protocol. As a result, you can send and receive information to/from various platforms. Using Web browsers, you can view information contained on Web sites. However, using SOAP you can:

- Combine the contents of different Web sites and services.
- Generate a complete view of all the relevant information.

z/VSE supports the SOAP protocol and therefore allows you to implement Web services. An example of using SOAP might be when a travel agent requires a combined view of the Web services covering hotel reservation, flight booking, and car rental. After the travel agent has entered the required data, all three Web services from the three different providers would be processed in one transparent step. This is an example of how a *business-to-business* (B2B) relationship can be implemented.

13.5.1 Overview of the SOAP syntax

You do not usually need to concern yourself with the tagging described here, since it is automatically generated by either the:

- SOAP client, and converted to native data by the SOAP server
- SOAP server, and converted to native data by the SOAP client-processor

However, for debugging purposes you might require detailed knowledge of the SOAP tagging. The information below provides you with an overview only.

A SOAP message is a standard XML document containing these main parts:

- SOAP envelope, which defines the content of the message
- SOAP header (optional), which contains header information
- SOAP body, which contains call and reply information

The types of elements used for the SOAP message are:

Envelope	The root element of a SOAP message. Defines the XML document to be a SOAP message.
Header	Can be used to include additional, application-specific information about the SOAP message. The information is user defined, such as the language used.
Body	Used to define the message itself.
Fault	Can be optionally used within the body element to supply information about errors that might have occurred when the SOAP message was processed.

Example 13-1 shows a SOAP XML document used for requesting the IBM share price. It is, of course, much simplified.

Example 13-1 SOAP XML document

```
<soap:Envelope>
        <soap:Body>
          <GetStock>
              <Company>IBM</Company>
          </GetStock>
        </soap:Body>
      </soap:Envelope>
```

13.5.2 How the z/VSE host can act as the SOAP server

Figure 13-4 shows how SOAP can be used in a CICS environment, when the z/VSE host acts as the SOAP server that provides SOAP services (in the z/VSE environment, CICS User Transactions).

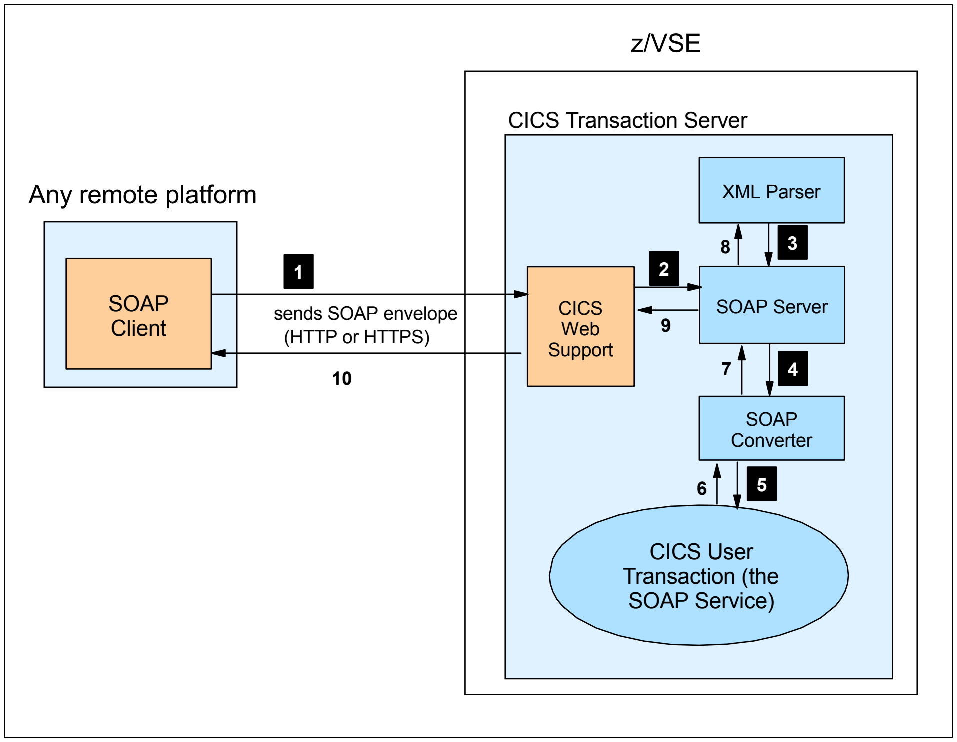


Figure 13-4 z/VSE acts as SOAP server

The steps in Figure 13-4 have the following meaning:

1. The SOAP client (for example, a platform using IBM WebSphere, Microsoft .NET, Apache SOAP, or AXIS) sends a SOAP envelope (in XML format) to the SOAP server running under CICS. The SOAP envelope is sent via the CICS Web Support (CWS) component of the CICS Transaction Server.
2. CWS forwards the SOAP envelope (in XML format) to the SOAP server running under CICS.
3. The SOAP server forwards the SOAP envelope to the XML parser, also running under CICS. The XML parser then parses the SOAP envelope from textual XML format into a tree-representation of the data. For example, if the data is to be processed by a C program, the SOAP envelope would be converted to a C program structure (with pointers) so that a C program

running on the z/VSE host could process the data, and returns this parsed XML tree to the SOAP server.

4. The SOAP server forwards the parsed XML tree to the SOAP converter running under CICS. The SOAP converter de-serializes (decodes) the parameter sub-tree contained in the parsed XML tree, and converts the parameter sub-tree into a binary representation.
5. The SOAP converter forwards the binary representation of the parameter sub-tree to the CICS user transaction running on the z/VSE host (the SOAP service) via the communication area (COMMAREA) of the CICS User Transaction. The CICS user transaction then processes the data.

The reply is then sent from the CICS user transaction (the SOAP service) back to the SOAP client, using the reverse of the above steps (that is, steps 6 to 10). The reply is sent via the communication area back to the SOAP converter, which serializes the parameters and returns them to the SOAP server. The SOAP server uses the XML parser to convert it from a tree-representation of the data to textual XML. The SOAP server then creates a SOAP envelope, which is then sent back (via HTTP or HTTPS) to the SOAP client. The SOAP client can then convert the SOAP envelope to its own native data format, and process the reply.

13.5.3 How the z/VSE host can act as the SOAP client

Figure 13-5 shows how SOAP can be used in a CICS environment, when the z/VSE host acts as the SOAP client.

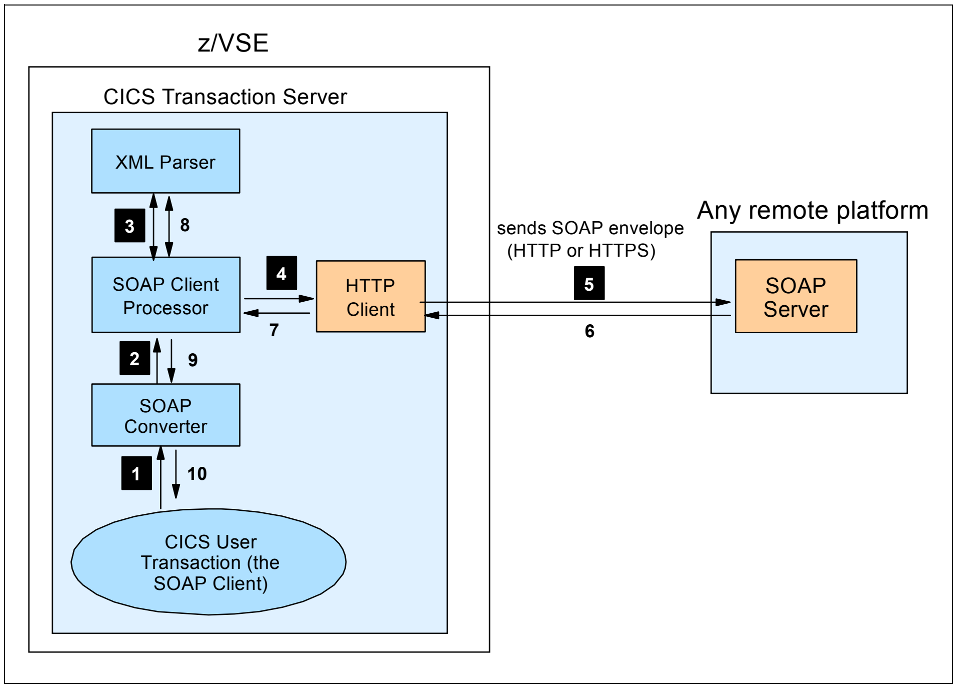


Figure 13-5 z/VSE acts as SOAP client

The steps in Figure 13-5 have following meaning:

1. The CICS User Transaction running on the z/VSE host (the SOAP client) sends the binary representation of the parameters to the SOAP converter. This is done via the communication area (COMMAREA) of the SOAP converter.
2. The encoder-part of the SOAP converter serializes (encodes) the binary representation of the parameters into an XML tree. The SOAP converter forwards the XML tree to the SOAP client-processor running under CICS.
3. The SOAP client-processor generates the SOAP envelope, and forwards it to the XML parser running under CICS. The XML parser then converts the XML tree into a textual XML format of the SOAP envelope. The XML parser returns the textual XML format to the SOAP client-processor.
4. The SOAP client-processor forwards the textual XML format to the HTTP client running under CICS.

5. The HTTP client sends the SOAP envelope (in textual XML format) to a SOAP server.

The reply is then sent from the SOAP server back to the CICS User Transaction (the SOAP client), using the reverse of the above steps (that is, steps 6 to 10). The SOAP server sends the reply back to the HTTP client, which forwards it to the SOAP client-processor. The SOAP client-processor calls the XML parser to parse the textual XML format into a tree representation. The tree is passed to the SOAP converter, which de-serializes the parameters into a binary representation, and forwards them to the CICS User Transaction. The CICS User Transaction then processes the reply.

13.5.4 XML parser

Part of the VSE SOAP Engine is the XML parser. It provides functionality to parse and generate XML documents with VSE programs.

The VSE XML Parser can be used in a CICS application as well as in a batch application. Since it is implemented in LE/C, it needs to be called from an LE conform program, or the LE environment must be set up prior to calling the XML Parser. In CICS you do not need to take care about that, since CICS does this for you (EXEC CICS LINK).

There are two types of XML Parser interfaces:

- The SAX (Simple API for XML, see `http://www.saxproject.org/`) like interface that uses callback functions for each XML element. Due to the callback mechanism this interface is designed to work best with C programming language applications, but it can also be used by any LE-enabled programming language.
- The DOM (Document Object Model, see `http://www.w3.org/DOM/`) like interface that builds a tree representation of the XML data in memory. When running in batch this interface can be used by a LE conforming program. When running in CICS it can be used by any kind of application since it uses an EXEC CICS LINK interface.

The XML Parser can also be used to create an XML data stream from a given tree representation in memory (DOM).

13.5.5 HTTP client

Part of the VSE SOAP Engine is the HTTP Client. It enables you to send HTTP requests to HTTP servers outside of VSE. This can be used to retrieve HTML documents from Web servers or call Common Gateway Interfaces (CGIs) or Servlets, among other functions.

The VSE HTTP Client can be used in a CICS application as well as in a batch application. Since it is implemented in LE/C, it needs to be called from a LE conforming program or the LE environment must be set up prior to calling the HTTP Client. In CICS you do not need to take care about that, since CICS does this for you (EXEC CICS LINK).

The HTTP Client supports HTTP 1.1 with methods GET and POST (refer to RFC 1945 and RFC 2616). Method GET is used to retrieve data from a HTTP server, for example, a HTML page or even binary content. Method POST is used to transfer data to the HTTP server and retrieve a response back. POST is typically used in HTML FORMs when calling CGIs or Servlets.

13.6 Overview of CICS connectivity

CICS connectivity in a 2-tier environment enables CICS applications to be accessed using the CICS Web Support and 3270 Bridge (functions delivered with CICS Transaction Server for VSE/ESA).

CICS connectivity in a 3-tier environment enables:

- A Java gateway application, which is usually stored on the middle-tier, to communicate with CICS applications running in the CICS TS through the External Call Interface (ECI) or External Presentation Interface (EPI) provided by the CICS Universal Client.
  - The ECI Interface enables a non-CICS Client application to call a CICS program synchronously or asynchronously as a subroutine.
  - The EPI Interface enables a non-CICS Client application to act as a logical 3270 terminal and so control a CICS 3270 application.
- The CICS Universal Client communicates with the CICS TS via the APPC protocol.
  - A CICS Java class library to be used for communication between the Java gateway application and a Java application (applet or servlet). The CICS Java class library also includes classes that provide an application programming interface (API).
  - Java programs can use the JavaGateway class to establish communication with the gateway process, and this class uses Java's sockets protocol:
    - ECIRequest class to specify the ECI calls that are flowed to the gateway.
    - EPIRequest class to specify EPI calls that are flowed to the gateway.

- A Web browser to be used as an emulator for a 3270 CICS application running on the CICS Transaction Server for VSE/ESA, via a Terminal Servlet.
- A set of Java EPI Beans to be used for creating Java front ends for existing CICS 3270 applications, without any programming effort.
- The Simple Object Access Protocol (SOAP) to be used to send and receive information between CICS programs and other modules, over the Internet.

13.7 Overview of the DB2-based connector

IBM DB2 database is also available for VSE. It is possible to access DB2 data on VSE from remote systems using DB2 Connect using Distributed Relational Database Architecture (DRDA). Vice versa, it is also possible to access remote DB2 data from a VSE application.

The DB2-based connector in VSE is an optional feature that allows you to use DRDA to access non-relational data such as VSE/VSAM and DL/I data. Your application programs use standard interfaces such as JDBC, ODBC, or Call Level Interface (CLI), provided by DB2 Connect, to access data.

The implementation of the DB2-based connector is based upon the use of DB2 Stored Procedures, which you can use with the DB2 Server for VSE & VM, Version 6 or later. DB2 Stored Procedures are application programs that you write and then compile and store on your z/VSE host.

You can write DB2 Stored Procedures in any Language Environment (LE) compliant language (COBOL, C, or PL/I). Local or remote DRDA applications can then invoke these DB2 Stored Procedures.

The DB2-based connector enables you to access VSE/VSAM and DL/I data from within the same DB2 Stored Procedure that you use to access DB2 data:

- To access VSE/VSAM data, use the VSAM Call Level Interface.
- To access DL/I data, use the AIBTDLI interface.

13.8 IBM WebSphere application server connectivity

In order to support WebSphere, the VSE connector class library implements the Common Client Interface (CCI). CCI is a standard API to access different systems and databases in the same way from a Web application server environment. The following sections give a brief overview of the J2EE connector architecture (J2C) and CCI.

13.8.1 J2EE connector architecture overview

As Java became the de facto language for Internet application development, many businesses have their enterprise applications based on it. Java 2 Platform Enterprise Edition (J2EE) is the specification for a standard Java platform to meet the requirements of the enterprise.

As the J2EE platform is designed for enterprise computing, it should be no surprise that applications need to connect to Enterprise Information Systems (EIS), such as:

- Enterprise resource planning (ERP) systems, such as SAP R/3®
- Mainframe transaction processing systems, such as CICS
- Legacy applications and non-relational database systems, such as IMS

In the past, most EIS vendors and application server vendors used vendor-specific architectures to provide EIS integration. This means that for each application server an EIS vendor wants to support, the EIS vendor needs to provide a specific connector (also called adapter). And for every connector an application server wants to support it will need to extend the server.

To overcome this problem the J2C defines a standard for connecting a compliant J2EE platform to EIS through the usage of a resource adapter and the common client interface.

13.8.2 Using the VSE connector class library as a resource adapter

If both EIS vendors and application server vendors follow this architecture, then only one resource adapter (RA) needs to be written for each EIS, which can plug into any J2EE-compliant application server.

Conversely, an application server only needs to be extended (or support the common client interface, CCI) to conform to the J2C, and this will ensure that any J2EE EIS connector will work with it.

The common client interface (CCI) defines a standard client API for application components to access the resource adapter. The intention of the CCI is not for application developers to use it directly, as it is a quite low-level API, but for enterprise application integration (EAI) frameworks to use, to generate EIS access code for the developer. The CCI is designed to be an EIS-independent API, such that an EAI can produce code for any J2EE-compliant resource adapter that also implements the CCI interface. This is shown in Figure 13-6.

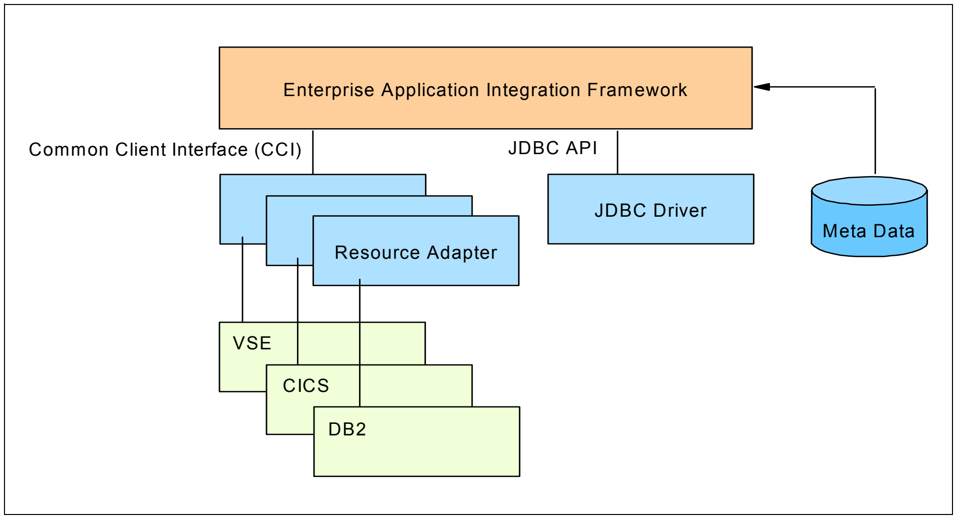

Enterprise Application Integration Framework
Common Client Interface (CCI)
JDBC API
JDBC Driver
Meta Data
Resource Adapter
VSE
CICS
DB2

Figure 13-6   Enterprise application integration framework

A particular WebSphere installation may include multiple resource adapters, each one for a specific EIS, (for example, z/VSE, CICS, or DB2). Each resource adapter has multiple data sources, where each data source gives access to a particular instance of an EIS (for example, a VSE system with given IP address, connector server port number, VSE user ID, and password for implicit logon). This makes it possible to write Web applications without referencing the properties of a specific backend system. Instead, some kind of URL, a JNDI name, is used for accessing the remote data source.

13.9 Mapping VSE/VSAM data to a relational structure

In the past, VSAM data was mainly accessed using application programs that understood the internal structure of the VSAM data. The record layout was represented by data structures within the application programs. The disadvantages of using this method are:

- There is no way to share a given record layout with other applications.
- If the record layout changes, the application program must also be changed.
- The data structure is dependent on the programming language (for example, COBOL, Assembler, or PL/I).
- Formatted data reports have to be created on the operating-system platform on which the application programs run.
- However, the development of e-business applications, as described in this book, requires:
  - A sharing of data representation across operating-system platforms.
  - Easy access to data representations from different applications.
  - That data representation and data display are independent of the operating-system platform and programming language.

To satisfy these requirements, VSE provides several means for creating so-called data maps and data views for given VSAM files. A data map consists of a set of data fields, where each data field has an offset within the VSAM record, a length, a data type (string, integer, signed integer, binary), and a name, the column name. A data view is a subset of a data map for the same VSAM file. Data maps and views are centrally stored on VSE in a special VSAM file, so that they can be accessed from any connector application.

There is a VSAM command RECMAP to define and modify maps and views, but there is also a Java-based utility program VSAM MapTool to manage them. As a third alternative, the VSE Java Beans class library provides functions for handling maps and views.

Figure 13-7 shows the relationship between a VSAM record, where VSAM only knows where the record begins, where it ends, and the location of the key, if any. But VSAM has no knowledge about the structure and the meaning of the data. A corresponding data map now represents the logical view on this record by identifying the data fields and providing knowledge about the meaning of single fields through field names.

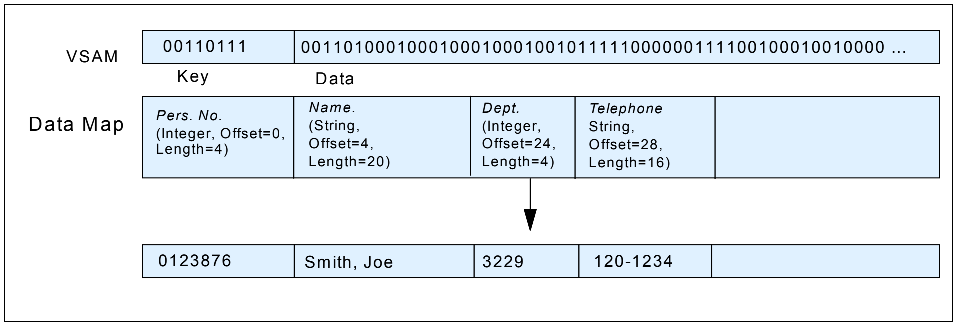

VSAM
00110111
0011010001000100010001001011111000000111100100010010000 ...
Key
Data
Data Map
Pers. No. (Integer, Offset=0, Length=4)
Name. (String, Offset=4, Length=20)
Dept. (Integer, Offset=24, Length=4)
Telephone String, Offset=28, Length=16)
0123876
Smith, Joe
3229
120-1234

Figure 13-7 Mapping of a VSAM record to a relational structure

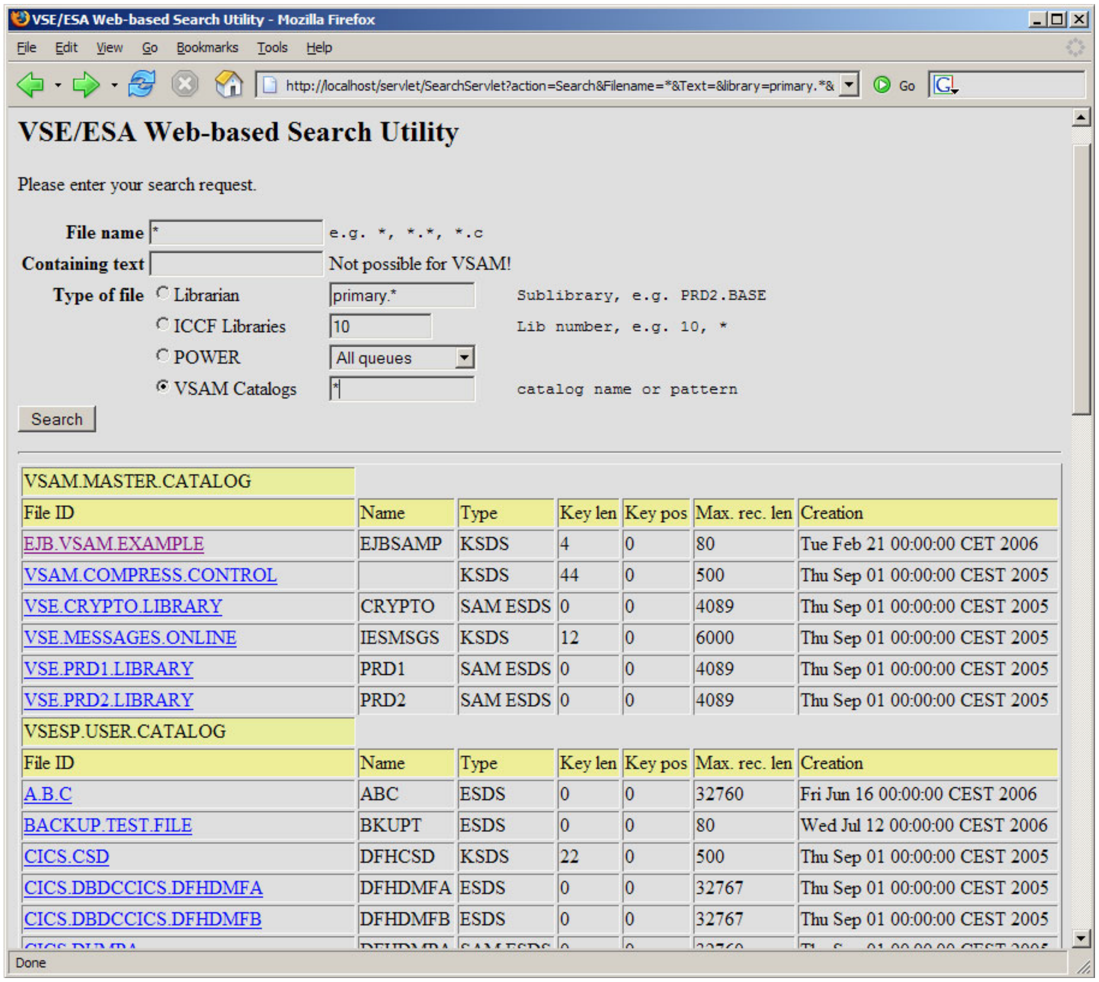
The mapping definitions of the VSAM record in Figure 13-7 have the following hierarchical structure.

Example 13-2 Mapping structure

```
CATALOG(MY.USER.CATALOG)
  CLUSTER(MY.DATA.CLUSTER)
    MAP(Map One)
      COLUMN("Part No.", Offset=0,Len=4,Type=Integer)
      COLUMN("Description", Offset=4, Len=20, Type=String)
      COLUMN("Supplier", Offset=24, Len=20, Type=String)
      …
    MAP(Map two)
    …
```

13.10 Using a servlet to display VSAM clusters

The VSE Connector client provides samples and documentation to demonstrate how the VSE Java Beans class library can be used as a resource adapter in WebSphere. The sample servlets demonstrate how VSE data can be accessed via Web browsers. Figure 13-8 shows how the SearchServlet can be used to display VSAM catalogs and clusters.


VSE/ESA Web-based Search Utility - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/servlet/SearchServlet?action=Search&Filename=*&Text=&library=primary.*&
Go
Done

VSE/ESA Web-based Search Utility

Please enter your search request.

File name	*	e.g. *, *.*, *.c	
Containing text		Not possible for VSAM!	
Type of file	○ Librarian	primary.*	Sublibrary, e.g. PRD2.BASE
	○ ICCF Libraries	10	Lib number, e.g. 10, *
	○ POWER	All queues	
	● VSAM Catalogs	*	catalog name or pattern

Search

VSAM.MASTER.CATALOG						
File ID	Name	Type	Key len	Key pos	Max. rec. len	Creation
EJB.VSAM.EXAMPLE	EJBSAMP	KSDS	4	0	80	Tue Feb 21 00:00:00 CET 2006
VSAM.COMPRESS.CONTROL		KSDS	44	0	500	Thu Sep 01 00:00:00 CEST 2005
VSE.CRYPTO.LIBRARY	CRYPTO	SAM ESDS	0	0	4089	Thu Sep 01 00:00:00 CEST 2005
VSE.MESSAGES.ONLINE	IESMSGS	KSDS	12	0	6000	Thu Sep 01 00:00:00 CEST 2005
VSE.PRD1.LIBRARY	PRD1	SAM ESDS	0	0	4089	Thu Sep 01 00:00:00 CEST 2005
VSE.PRD2.LIBRARY	PRD2	SAM ESDS	0	0	4089	Thu Sep 01 00:00:00 CEST 2005
VSESP.USER.CATALOG						
File ID	Name	Type	Key len	Key pos	Max. rec. len	Creation
A.B.C	ABC	ESDS	0	0	32760	Fri Jun 16 00:00:00 CEST 2006
BACKUP.TEST.FILE	BKUPT	ESDS	0	0	80	Wed Jul 12 00:00:00 CEST 2006
CICS.CSD	DFHCSD	KSDS	22	0	500	Thu Sep 01 00:00:00 CEST 2005
CICS.DBDCCICS.DFHDMFA	DFHDMFA	ESDS	0	0	32767	Thu Sep 01 00:00:00 CEST 2005
CICS.DBDCCICS.DFHDMFB	DFHDMFB	ESDS	0	0	32767	Thu Sep 01 00:00:00 CEST 2005

Figure 13-8 Using a servlet to display VSAM information with a Web browser

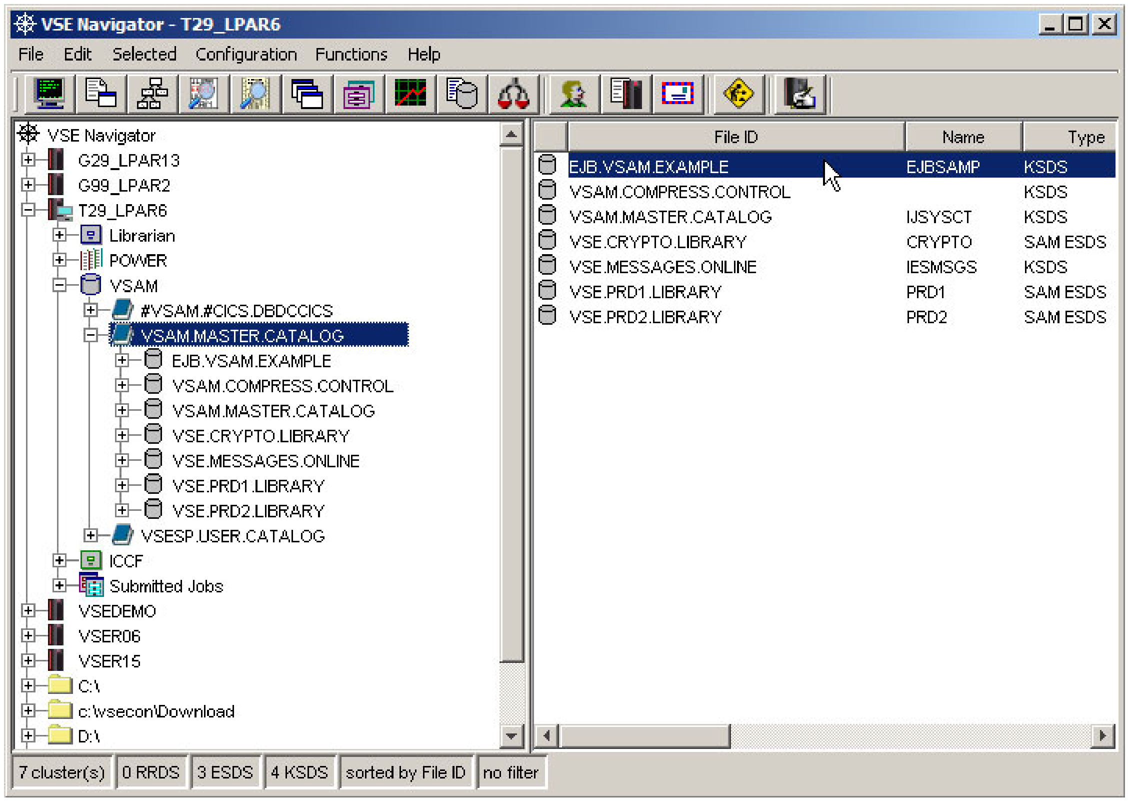
13.11 Using the VSE Navigator application

The VSE Navigator is a Java application that uses virtually all of the VSE Java Beans. It implements a graphical user interface (GUI) that has a similar appearance to many of the currently available file managers.

The client-part of the VSE Navigator, which communicates with the VSE Connector server, provides you with a variety of functions. You can, for example:

- Access VSE file systems (POWER, Librarian, ICCF, VSAM).
- Create and submit jobs, including generating jobs based on the skeletons stored in ICCF library 2.
- Work with the VSE operator console.
- Compare files and perform a full-text search in VSE-based file systems.
- Interactively insert and edit VSAM records.
- Display:
  - VSE hardware configuration, including the property dialogs for attached devices
  - VSE system activity (CPU usage, and so on)
  - Current VSE service level
  - System labels
  - System tasks
  - Used and free VSAM space
  - VSAM data, via maps and views

Figure 13-9 shows how to select a VSAM file from the VSE Navigator main window.



VSE Navigator
- G29_LPAR13
- G99_LPAR2
- T29_LPAR6
  - Librarian
  - POWER
  - VSAM
    - #VSAM.#CICS.DBDCCICS
    - VSAM.MASTER.CATALOG
      - EJB.VSAM.EXAMPLE
      - VSAM.COMPRESS.CONTROL
      - VSAM.MASTER.CATALOG
      - VSE.CRYPTO.LIBRARY
      - VSE.MESSAGES.ONLINE
      - VSE.PRD1.LIBRARY
      - VSE.PRD2.LIBRARY
    - VSESP.USER.CATALOG
  - ICCF
  - Submitted Jobs
- VSEDEMO
- VSER06
- VSER15
- C:\
- c:\vsecon\Download
- D:\

File ID	Name	Type
EJB.VSAM.EXAMPLE	EJBSAMP	KSDS
VSAM.COMPRESS.CONTROL		KSDS
VSAM.MASTER.CATALOG	IJSYSCT	KSDS
VSE.CRYPTO.LIBRARY	CRYPTO	SAM ESDS
VSE.MESSAGES.ONLINE	IESMSGS	KSDS
VSE.PRD1.LIBRARY	PRD1	SAM ESDS
VSE.PRD2.LIBRARY	PRD2	SAM ESDS

7 cluster(s) | 0 RRDS | 3 ESDS | 4 KSDS | sorted by File ID | no filter

Figure 13-9 Using the VSE Navigator application to display VSAM files

13.12 Summary

In a heterogeneous IT environment, connectors are the essential building blocks to tie different platforms and operating systems together. Connectors do not only provide file transfer between different platforms, but allow programs on different systems to talk directly to each others.

Some of the main functional benefits of the z/VSE connectors are:

- Access to VSE functions and data from any Java program on any Java-enabled platform via the Java-based connector. This includes WebSphere Application Server servlets, applets, EJBs, and JSPs.

- Redirection of VSAM data to relational databases or any file systems without the need for changing the legacy applications working on these data via the VSAM Redirector connector.
- Access to VSE functions and data from non-Java applications, like text processors or spreadsheet programs via the VSE Script connector.
- Access to WebServices in the company's intranet or the Internet from mainframe CICS applications.
- Possibility for providing WebServices on a z/VSE system as a CICS application.

Key terms in this chapter		
Connector	SOAP	VSAM data mapping
J2EE connector architecture	XML	WebSphere
Resource adapter	J2EE	Web application server
CCI	WebServices	Servlet

13.13 Questions for review

To help test your understanding of the material in this chapter, complete the following questions:

1. What is the main difference between the Java-based connector and the VSAM Redirector connector?
2. Why can the VSE SOAP connector be described as a 2-way connector?
3. Explain the function of a Resource Adapter.
4. What is the difference between a VSAM data map and a data view?
5. Describe the two working modes of the VSAM Redirector Connector.
6. What is the difference between a Web server (HTTP server) and a Web application server?

14

Messaging and queuing

Objective: As a mainframe professional, you should understand messaging and queuing. These functions are often used for communication between heterogeneous applications and platforms.

After completing this chapter, you will be able to:

- Explain why messaging and queuing is used.
- Describe the asynchronous flow of messages.
- Explain the function of a queue manager.

14.1 What is MQSeries (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 MQSeries (on other platforms it is called IBM WebSphere MQ) facilitates application integration by passing messages between applications. It is used on more than 35 hardware platforms and for point-to-point messaging from Java[1], C, C++, and COBOL applications.

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 MQSeries for VSE, 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.

[1] Java and C++ are not supported on z/VSE.

14.2 Synchronous communication

Figure 14-1 shows the basic mechanism of program-to-program communication using a synchronous communication model.

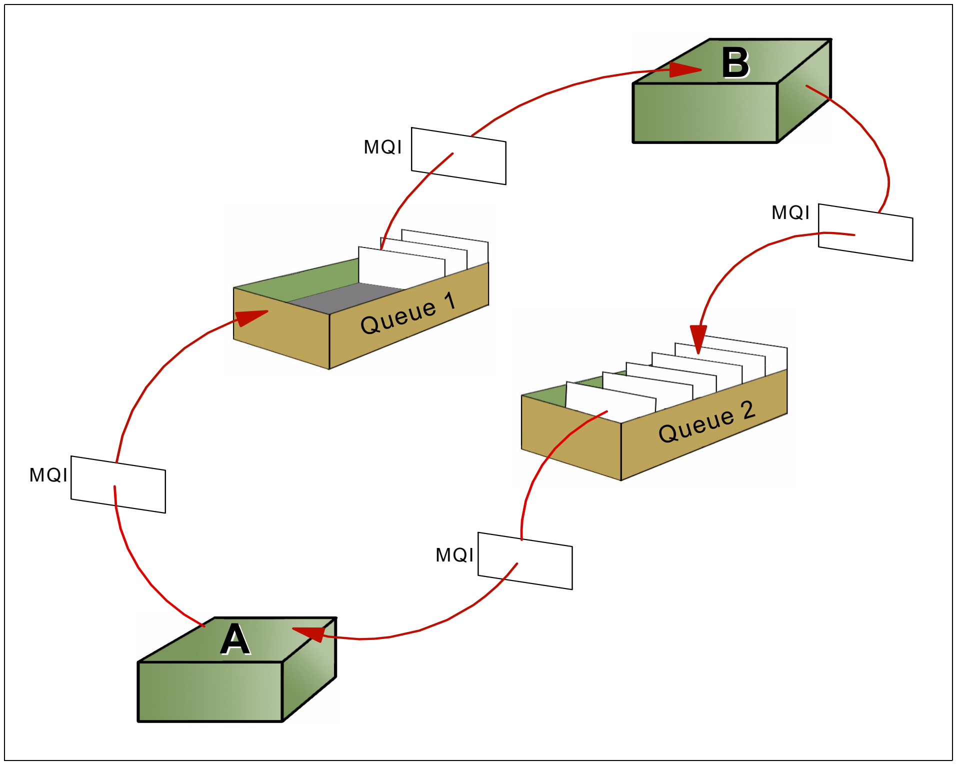


Figure 14-1 Synchronous application design model

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 MQSeries 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.

The synchronous communication is program controlled. 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 no longer as desirable. Asynchronous communication is designed to handle those situations.

14.3 Asynchronous communication

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, that 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 14-2.

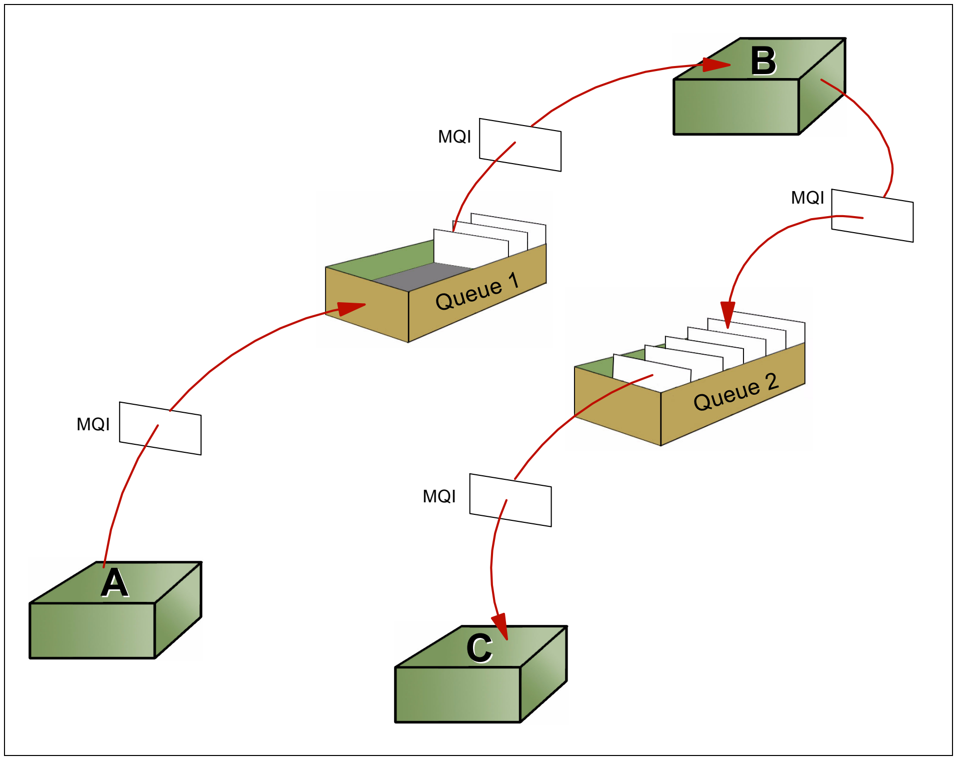


Figure 14-2 Asynchronous application design model

As previously stated, the *asynchronous model is natural for MQSeries*. 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 MQSeries, in which communicating applications do not have to be active at the same time, is known as *time independence*.

14.4 Messages

MQSeries 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.

MQSeries messages have two parts:

Application data	The content and structure of the application data is defined by the application programs that use them.
Message descriptor	The message descriptor identifies the message and contains other control information, such as the type of message and the priority assigned to the message by the sending application.

14.5 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
- recoverable
- 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 MQSeries.

14.5.1 What is a queue manager?

The component of software that owns and manages queues is called a queue manager (QM). In MQSeries, 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. Figure 14-4 on page 366 shows the flow of activity in queue managers and database managers.

14.5.2 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 does not 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 MQSeries 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.

14.6 What is a 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.

MQSeries 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 an MQSeries client to a queue manager. Clients do not have a queue manager and therefore no queues of their own. An MQI channel is bidirectional.

14.7 How transactional integrity is ensured

A business might require two or more distributed databases to be maintained. MQSeries offers a solution involving multiple units of work acting asynchronously, as shown in Figure 14-3.

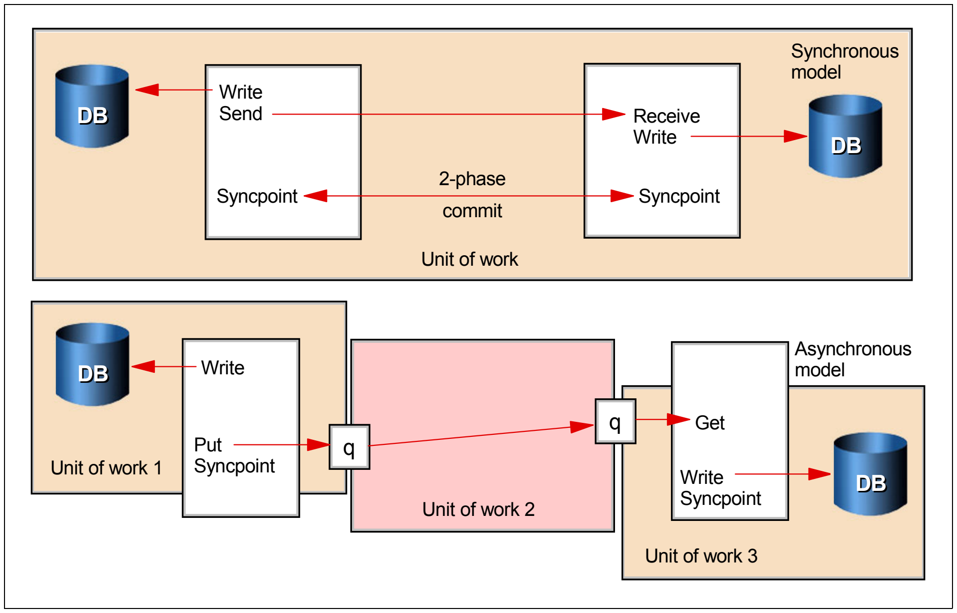


Figure 14-3 Data integrity

The top half of Figure 14-3 shows a two-phase commit structure, while the MQSeries 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 that 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 MQSeries. 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.

It is the transactional integrity of units of work 1 and 3, and the once and once only assured delivery property of MQSeries 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.

14.8 Example of messaging and queuing

Now let us 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 (Figure 14-4).

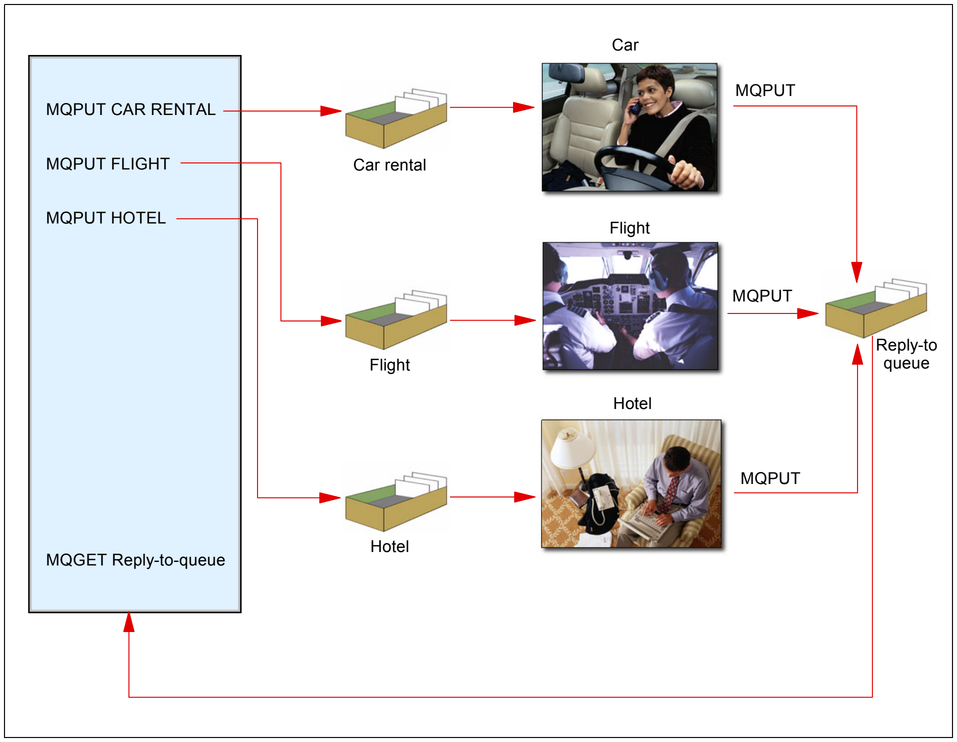


Figure 14-4 Parallel processing

With a message queue manager product such as MQSeries for VSE, 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.

14.9 IBM MQSeries for VSE

IBM MQSeries (WebSphere MQ) is available for a variety of platforms. On z/VSE messaging and queuing is supported by:

- IBM WebSphere MQ client for VSE

  An MQSeries client (WebSphere MQ client) is an MQSeries system that has no queue manager and of course no queues for its own. It directs MQI calls from the applications to a queue manager on an MQSeries server system to which it is connected.

- IBM MQSeries for VSE

  The product runs in a CICS partition as a CICS subsystem. Consequently, various features of the product are controlled by CICS itself.

  MQSeries for VSE provides an interface for batch programs. The *batch interface* is designed to standardize the programming style of CICS and batch programs.

  The *MQSeries-CICS Bridge* enables an application, not running in a CICS environment, to run a program or transaction on CICS and get a response back. This non-CICS application can be run from any environment that has access to an MQSeries network that encompasses MQSeries for VSE.

  MQSeries for VSE can function as an MQSeries server system to *MQSeries clients* (including Java clients, but not C++ clients) that can connect to the server.

14.10 Summary

In an online application environment, messaging and queuing enables communication between applications on different platforms. IBM MQSeries for VSE is an example of software that manages messaging and queuing in the mainframe and other environments. With messaging, programs communicate by through messages, rather than by calling each other directly. With queuing, messages are retained on queues in storage, so that programs can run independently of each other (asynchronously).

Here are some of the functional benefits of MQSeries:

- A common application programming interface, the MQI, which is consistent across the supported platforms.

- Data transfer data with assured delivery. Messages are not lost, even if a system fails. Nor is there duplicate delivery of messages.
- Asynchronous communication. That is, communicating applications need not be active at the same time.
- Message-driven processing as a style of application design. An application is divided into discrete functional modules that can run on different systems, be scheduled at different times, or act in parallel.
- Application programming is made faster when the programmer is shielded from the complexities of the network.

Key terms in this chapter		
Local queue	Channel	Message-driven
MQI	Asynchronous application	Dead-letter queue
QM	Remote queue	WebSphere MQ client

14.11 Questions for review

To help test your understanding of the material in this chapter, complete the following questions:

1. Why is messaging and queuing helpful for communication between heterogeneous applications and platforms?
2. Describe the asynchronous flow of messages.
3. Explain the function of a queue manager.
4. What is the purpose of MQI?
5. What is a dead-letter queue used for?

Part 4

System programming on z/VSE

In this part we reveal the inner workings of z/VSE with discussions of system libraries, security, and procedures for starting (IPLing) and stopping a z/VSE system. This part also includes chapters on hardware details and virtualization, and the network communications on z/VSE.

15

Overview of system programming

Objective: As a z/VSE system programmer, you need to know how z/VSE works.

After completing this chapter, you will be able to:

- Discuss the responsibilities of a z/VSE system programmer.
- Explain system libraries and their content.
- Describe the process of IPLing a system.
- Explain system installation and maintenance.

15.1 The role of the system programmer

The system programmer is responsible for managing the mainframe hardware configuration, and installing, customizing, and maintaining the mainframe operating system. Installations need to ensure that their system and its services are available and operating to meet service level agreements. Installations with 24-hour, 7-day operations need to plan for minimal disruption of their operation activities.

In this chapter we examine several areas of interest for the would-be z/VSE system programmer. While this text cannot cover every aspect of system programming, it is important to learn that the job of the z/VSE system programmer is very complex and requires skills in many aspects of the system, such as:

- Device I/O configurations
- Processor configurations
- Console definitions
- System libraries where the software is placed
- System data sets and their placement
- Customization parameters that are used to define your z/VSE configuration
- Installation and maintenance
- Security administration

As shown in Figure 15-1, the role of the system programmer usually includes some degree of involvement in all of the following aspects of system operation:

- Customizing the system
- Initializing the system
- Displaying system status
- Installing and maintaining the system

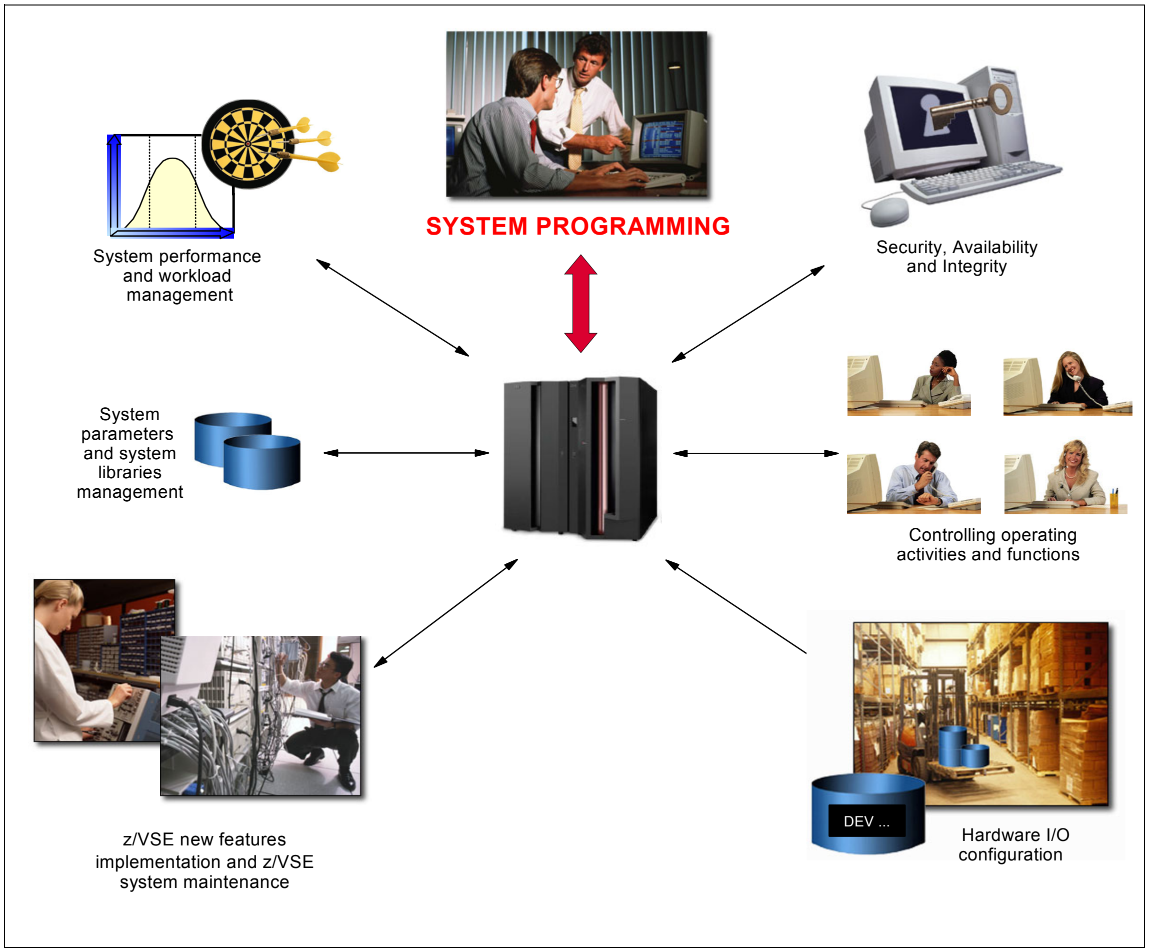

SYSTEM PROGRAMMING
System performance and workload management
Security, Availability and Integrity
System parameters and system libraries management
Controlling operating activities and functions
z/VSE new features implementation and z/VSE system maintenance
DEV ...
Hardware I/O configuration

Figure 15-1 Some areas in which the system programmer is involved

15.2 Customizing the system

Since z/VSE systems are used for different purposes and run very different application programs, the standard configuration parameters may not be optimal for each installation. Therefore, z/VSE allows the system to be configured to your needs. Configuration is required to make the best usage of system resources (for example, to optimize the system performance).

This section describes the following topics:

- System libraries where the software is located
- I/O device configuration
- Console definitions
- Customization parameters used to define the z/VSE configuration

15.2.1 z/VSE system libraries

First there is the z/VSE software as supplied by IBM. This is usually installed to two-disk volumes known as the system residence volumes (DOSRES) and the system work disk 1 (SYSWK1).

The system library IJSYSRS resides on the volume DOSRES. The PRD1 and PRD2 libraries reside in VSAM-managed space that is usually allocated on volumes DOSRES and SYSWK1.

Fixes to z/VSE are managed with a product called maintain system history program (MSHP). It uses the system history file (IJSYSHF) to store information about installed products and fixes. The system history file also contains information about the library and sublibrary where a product is installed. This information is used to catalog fixes into the correct sublibrary.

z/VSE provides a service dialog that controls the service application process. Primarily, the service dialog controls the indirect service application. With indirect service application the fixes are first installed to the volume SYSWK1. Then the z/VSE is started from the SYSWK1 and the fixes can be tested. Finally, the fixes are merged from volume SYSWK1 to volume DOSRES.

z/VSE has three standard system libraries: IJSYSRS, PRD1, and PRD2. They contain sublibraries such as IJSYSRS.SYSLIB, PRD1.BASE, PRD1.MACLIB, PRD2.SCEEBASE, and PRD2.DB2740. Some of these are related to IPL processing, while others are related to the search order of invoked programs, as described next.

15.2.2 IJSYSRS

z/VSE allocates space for library IJSYSRS on the DOSRES volume. It has only one sublibrary, named SYSLIB. IJSYSRS and IJSYSRS.SYSLIB are also referred to as system library. The library IJSYSRS resides at the start location of the volume DOSRES because it contains the IPL records that are required for system startup. It cannot reside in VSAM-controlled space because the VSAM support is activated after IPL.

IJSYSRS.SYSLIB

This contains the base programs of z/VSE:

- *VSE/Advanced Functions* provides basic system control and includes the supervisor and system programs such as the librarian and the Linkage Editor.
- *VSE/SP Unique Code* provides productivity support such as the Interactive Interface and workstation-related support.
- *VSE/POWER* (VSE/Priority Output Writers, Execution Processors and Input Readers) provides spooling support and networking control.
- *VSE/ICCF* (VSE/Interactive Computing and Control Facility) provides interactive partition support and support for VSE/ICCF libraries.
- *VSE/VSAM* (VSE/Virtual Storage Access Method) provides data management support.
- *VSE/FastCopy* is a utility program providing FastCopy support for data on disk or tape devices.
- *ICKDSF* (Device Support Facilities) belongs to basic system control and provides support for maintaining devices such as disks and tapes.

15.2.3 PRD1

Library PRD1 contains base products of z/VSE, consisting of macros and programs that are not required to be in IJSYSRS. It includes the sublibraries BASE and MACLIB.

PRD1.BASE

This contains additional components of z/VSE:

- *OSA/SF for VSE/ESA* (Open Systems Adapter Support Facility) provides support for customizing and managing Open Systems Adapters.
- *REXX/VSE* provides programming language support.
- *VSE/OLTEP* (VSE/Online Test Executive Program) belongs to basic system control and is intended for use by IBM personnel.
- *VSE Connectors* includes the VSE Connector Server, the VSE Connector Client, the VSE/VSAM CLI (Call Level Interface), and the VSE/VSAM Redirector Connector.
- *VTAM* (Virtual Telecommunications Access Method) provides display station and networking control support.

- *CICS Transaction Server for VSE/ESA* provides enhanced (compared to CICS/VSE) transaction processing support.
- *TCP/IP for VSE/ESA* provides a set of protocols by which computers can communicate with any other computer that runs an operating system with a TCP/IP implementation.
- *High Level Assembler for VSE* provides enhanced programming language support at the Assembler level, replacing the DOS/VSE Assembler.
- *DITTO/ESA for VSE* (Data Interfile Transfer, Testing and Operations Utility) provides support for moving data fast from one media to another.
- *EREP* (Environmental Recording Editing and Printing) belongs to basic system control and provides diagnostics support.

PRD1.MACLIB

This contains the macros available for the customer:

- VSE/Advanced Functions Macros
- VSE/VSAM Macros
- VSE/POWER Macros

15.2.4 PRD2

Library PRD2 contains the sublibraries CONFIG and SAVE, and other default sublibraries for z/VSE optional IBM products and ISV products.

PRD2.CONFIG

This contains installation-unique members that are not required in IJSYSRS, like:

- Members created during initial installation
- Members created when you use the Interactive Interface (for example, CICS/VSE tables and VTAM startup books)

PRD2.SAVE

This contains mainly system procedures (PROCs) and is for fast service upgrade (FSU)[1] only.

[1] FSU is a special method to install a defined amount of service on z/VSE to reach a new release or modification level of the z/VSE system. See also 15.5, “Installing and maintaining the system” on page 387.

15.2.5 Console definitions

Consoles are used to operate the z/VSE system. You can, for example, start or stop the system and any program, enter system commands, receive messages, and act accordingly. z/VSE supports the following physical console types:

- Integrated console

  This is the console used on zSeries processors to control and maintain the configuration of the processor. It is also called HW console.

- Line mode console

  This is the existing line mode system console based on a virtual 3215 printer-keyboard. It is only used for z/VSE systems running under z/VM.

- 3270 console

  This is the existing z/VSE full-screen system console support based on a local non-SNA 3270 terminal.

- Interactive Interface consoles

  These consoles replace the former VSE/ICCF-based consoles. Their capabilities, presentation characteristics, and performance have been improved in such a way that they can effectively replace a physical 3270 console.

On a logical level, z/VSE distinguishes between system, master, and user consoles:

System console	The system console is identified by being assigned the logical unit name SYSLOG. The 3270 system console, the Line Mode console, or the Integrated Console may be used for this purpose. The system console is used to IPL the system and is active after IPL.
Master console	A z/VSE system may have several master consoles. They are activated by a console program. Each master console receives all system messages that are not routed exclusively to a specific console, and it can reply to all outstanding messages and enter all system commands.
User console	User consoles differ from master consoles with respect to their input capability or to the scope of messages routed to them.

15.2.6 I/O device configuration

The HW configuration is defined in the IOCDS, as described in Chapter 2, “Mainframe hardware systems” on page 35. To enable z/VSE to access these devices, they must be defined to z/VSE during IPL with so-called ADD statements (see 15.3.3, “IPL and system startup” on page 380). Adding, changing, or deleting devices is supported by the hardware configuration dialog. This dialog maintains the ADD statements, and the changed I/O configuration will then be used at the next IPL.

15.2.7 Customization parameters for the z/VSE configuration

Besides the I/O device configurations and the parameters specified in 15.3.3, “IPL and system startup” on page 380, there are a lot more parameters and commands to modify the z/VSE configurations. To mention all of them would exceed the limits of this book. But for two of them we should have a short look:

ALLOC	The ALLOC command allocates the virtual or real storage to the static partitions of the z/VSE system. You can define the storage size in a static partition as required by the application, which will run in this partition.
PRTY	You can use the PRTY command to define the priorities of the partitions. This is important for the system performance. For example, to avoid problems with spooling, POWER should have a higher priority than a simple batch job.

15.3 Initializing the system

An initial program load (IPL) is the act of loading a copy of the operating system from disk into the processor’s real storage and executing it.

z/VSE systems are designed to run continuously with many weeks or even months between reloads, allowing important production workloads to be continuously available. Change is the usual reason for a reload, and the level of change on a system dictates the reload schedule. For example:

- A test system may be IPLed daily or even more often.
- A high-availability banking system may only be reloaded once a year, or even less frequently, to refresh the software levels.
- Outside influences may often be the cause of IPLs, such as the need to test and maintain the power systems in the machine room.
- Sometimes badly behaved software uses up system resources that can only be replenished by an IPL, but this sort of behavior is normally the subject of investigation and correction.

For the system startup we have to differentiate between two different device characteristics of the disk from where the initialization programs are loaded. z/VSE supports channel-attached disks and FCP-attached SCSI disks.

15.3.1 IPL from channel-attached disk (CKD or ECKD)

For a native z/VSE system in a LPAR the initialization process from a channel-attached disk begins when the system programmer selects the LOAD function at the Hardware Management Console (HMC). You have to specify the device address of your SYSRES volume (which customarily has volume serial number DOSRES).

Running z/VSE as a VM guest, the IPL cuu command is used to start the system, where cuu is the disk address of your SYSRES volume.

15.3.2 IPL from FCP-attached SCSI disk

To perform an IPL of z/VSE from a FCP-attached SCSI disk, the SCSI IPL hardware feature must already be installed and enabled on your System z processor. If your z/VSE system is running under z/VM, the z/VM system must also support IPL from SCSI.

To IPL from a VM guest you must first use the SET LOADDEV command to supply the required parameters. You use the SET LOADDEV command to provide the machine loader with the parameters this program needs in order to access a SCSI disk. Then you can IPL the FCP device that attaches the SYSRES device.

To IPL from a LPAR you must first select **SCSI** in the Load panel at the HMC. Then you must enter these values:

- Load address

  This is the device number of the FCP device.
- World Wide Port Name (WWPN)

  This is the WWPN of the port on the ESS controller that is used to connect to the SCSI disk.
- Logical unit number

  This is the LUN number of the SCSI disk from which the z/VSE operating system is to be IPLed.

15.3.3 IPL and system startup

The process of IPL and system startup is also referred to as Automated System Initialization (ASI). The IPL process is either controlled by procedures from the IJSYSRS.SYSLIB sublibrary or by entering the IPL commands interactively at the console.

The first procedure that is to be executed is the ASI Master Procedure $ASIPROC. In an environment with multiple z/VSE systems sharing the DOSRES (system residence) disk device or an environment where two or more z/VSE Systems run as guest systems under VM, this procedure serves to define the following IPL and JCL procedures. You can also define a STOP parameter that allows stopping the IPL process for one or more IPL commands. This enables the operator to modify IPL parameters and commands interactively.

The next procedure is the IPL procedure. The default IPL procedure is the $IPLESA.PROC. It contains the following IPL commands that allows defining the z/VSE system for the specific needs and hardware configuration of the system:

- ADD

  The ADD command is used to define the physical I/O devices attached to the system.

- DEF

  The mandatory DEF command is used to assign a physical device to system files.

  - SYSREC - the logical device for the system recorder file, the hard-copy file, and the job manager file.
  - SYSCAT - the logical device for the VSE/VSAM master catalog.

- DEF SCSI

  The DEF SCSI command is used to associate the VSE SCSI device number (FBA) with the real SCSI Logical Unit Number (LUN) and its connection path (FCP,WWPN).

- DEL

  The DEL command is used to delete one or more of the I/O devices previously defined with the ADD command.

- DEV

  The DEV command is used to display all I/O devices sensed and added. This allows you to identify all devices that are not needed.

- DLF

  The DLF command is used to define or reference the cross-system communication file (lock file). The file must exist when two or more VSE systems share disk storage devices.

- DPD

  The DPD command is used to define the page data set (PDS). The PDS is required to store paged-out pages of programs executing in virtual mode.

- SET

  The optional SET command is used to set the system date, the time-of-day (TOD) clock, and the system time zone. It is required only if the TOD clock has not been set since the last POWER ON.

- SET XPCC

  This command is used to activate the VSE/XPCC/APPC/VM support, which enables VSE DB2 applications to share one or more DB2 databases with applications running on other VM guest machines.

- SET ZONEDEF / SET ZONEBDY

  The purpose of the SET ZONEDEF and SET ZONEBDY is to be able to switch between standard and daylight saving local times without changing the IPL startup procedure each time.

- SVA

  This command is mandatory and must be the last command entered during the IPL procedure. It is used to allocate space within the SVA into which the user can later load his phases.

- SYS

  The optional SYS command specifies various system options, such as number of partitions, supervisor buffers, or librarian table entries.

When all commands in the IPL procedure are processed, the IPL process is completed and the operating system is loaded completely.

However, there is only one partition, the BG partition, allocated. No program or subsystem is running in the z/VSE system. Therefore the JCL startup procedures and jobs are processed. The JCL startup procedure for the BG partition $0JCL.PROC is called. The ‘0’ in the second position of ‘$0JCL’ indicates the BG partition. The BG startup procedure performs several actions, for example:

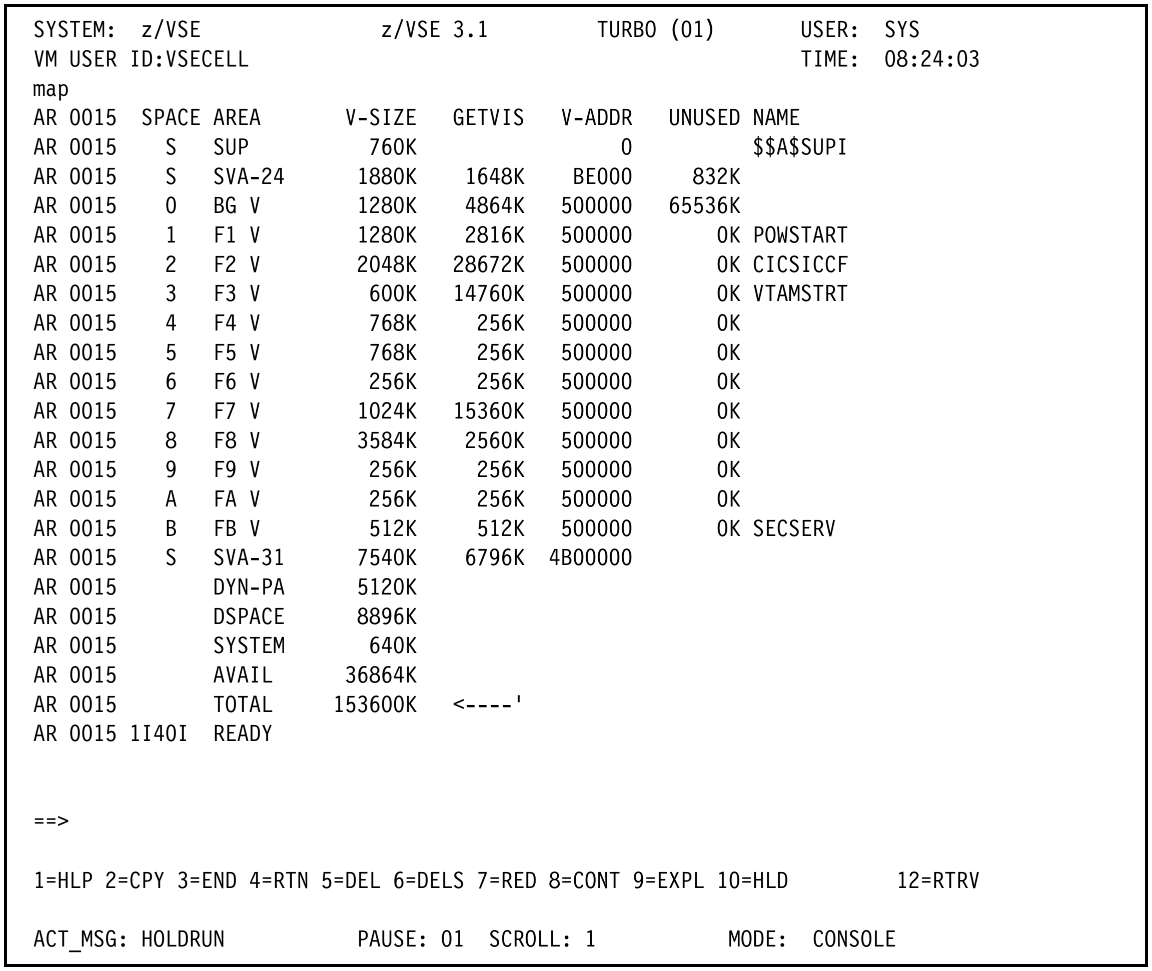
- Execute STDLABEL.PROC to write file labels into the system label information area.
- Execute SETSDL.PROC to load the System Directory List (SDL) and optionally load selected system phases into the Shared Virtual Area (SVA).

- Execute startup program DTRISTRT that decides which startup mode is to be used.
- The SDL is loaded with the Phases for subsystems and components like REXX, LE/VSE, and High Level Assembler.
- The static partitions are allocated.
- The security server partition is started.
- VSE/POWER is started by starting the F1 partition and executing the startup procedure for F1 partition $1JCL.PROC.

Under control of VSE/POWER, the other partitions and subsystems (like CICS, VTAM) are started.

The console command MAP gives an overview of the system when the system startup has been completed. The example in Figure 15-2 shows that:

- POWER uses partition F1.
- CICS uses partition F2.
- VTAM uses partition F3.
- The security server uses partition FB.



```
SYSTEM:  z/VSE              z/VSE 3.1        TURBO (01)       USER:  SYS
VM USER ID:VSECELL                                            TIME:  08:24:03
map
AR 0015  SPACE AREA       V-SIZE   GETVIS   V-ADDR   UNUSED NAME
AR 0015    S   SUP          760K                 0          $$A$SUPI
AR 0015    S   SVA-24      1880K    1648K    BE000     832K
AR 0015    0   BG V        1280K    4864K   500000   65536K
AR 0015    1   F1 V        1280K    2816K   500000       0K POWSTART
AR 0015    2   F2 V        2048K   28672K   500000       0K CICSICCF
AR 0015    3   F3 V         600K   14760K   500000       0K VTAMSTRT
AR 0015    4   F4 V         768K     256K   500000       0K
AR 0015    5   F5 V         768K     256K   500000       0K
AR 0015    6   F6 V         256K     256K   500000       0K
AR 0015    7   F7 V        1024K   15360K   500000       0K
AR 0015    8   F8 V        3584K    2560K   500000       0K
AR 0015    9   F9 V         256K     256K   500000       0K
AR 0015    A   FA V         256K     256K   500000       0K
AR 0015    B   FB V         512K     512K   500000       0K SECSERV
AR 0015    S   SVA-31      7540K    6796K  4B00000
AR 0015        DYN-PA      5120K
AR 0015        DSPACE      8896K
AR 0015        SYSTEM       640K
AR 0015        AVAIL      36864K
AR 0015        TOTAL     153600K   <----'
AR 0015 1I40I  READY



==>

1=HLP 2=CPY 3=END 4=RTN 5=DEL 6=DELS 7=RED 8=CONT 9=EXPL 10=HLD         12=RTRV

ACT_MSG: HOLDRUN           PAUSE: 01  SCROLL: 1          MODE:  CONSOLE
```

Figure 15-2 Example of the z/VSE mapping after startup

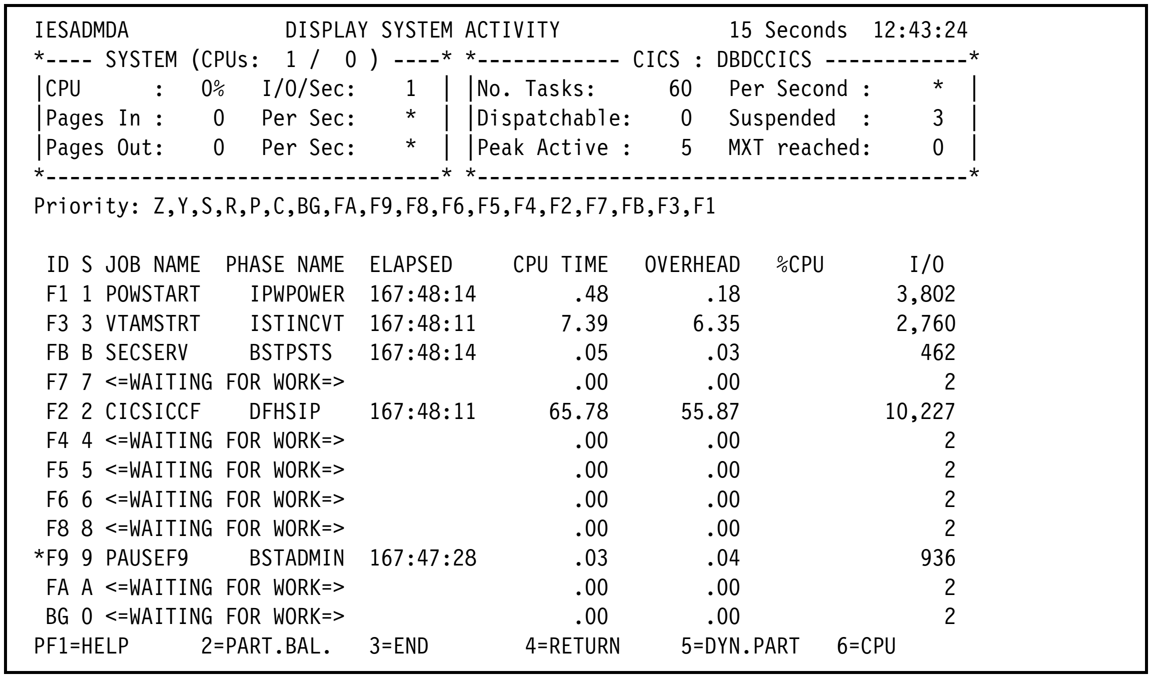
This is the installation default setup. The customer can change these assignments and may start additional programs in the other partitions.

15.4 Displaying system status

The Interactive Interface of z/VSE provides dialogs for displaying the status of the system:

- Display System Activity
- Display Channel and Device Activity
- Display Storage Layout
- Display CICS TS Storage

The first two dialogs, Display System Activity and Display Channel and Device Activity, dynamically provide system status information for daily operation (see Figure 15-3 and Figure 15-4 on page 385).



```
IESADMDA             DISPLAY SYSTEM ACTIVITY              15 Seconds  12:43:24
*---- SYSTEM (CPUs:  1 /  0 ) ----* *------------ CICS : DBDCCICS ------------*
|CPU      :   0%   I/O/Sec:    1  | |No. Tasks:      60   Per Second :     *  |
|Pages In :    0   Per Sec:    *  | |Dispatchable:    0   Suspended  :     3  |
|Pages Out:    0   Per Sec:    *  | |Peak Active :    5   MXT reached:     0  |
*---------------------------------* *-----------------------------------------*
Priority: Z,Y,S,R,P,C,BG,FA,F9,F8,F6,F5,F4,F2,F7,FB,F3,F1

 ID S JOB NAME  PHASE NAME  ELAPSED     CPU TIME   OVERHEAD   %CPU       I/O
 F1 1 POWSTART    IPWPOWER  167:48:14        .48        .18             3,802
 F3 3 VTAMSTRT    ISTINCVT  167:48:11       7.39       6.35             2,760
 FB B SECSERV     BSTPSTS   167:48:14        .05        .03               462
 F7 7 <=WAITING FOR WORK=>                   .00        .00                 2
 F2 2 CICSICCF    DFHSIP    167:48:11      65.78      55.87            10,227
 F4 4 <=WAITING FOR WORK=>                   .00        .00                 2
 F5 5 <=WAITING FOR WORK=>                   .00        .00                 2
 F6 6 <=WAITING FOR WORK=>                   .00        .00                 2
 F8 8 <=WAITING FOR WORK=>                   .00        .00                 2
*F9 9 PAUSEF9     BSTADMIN  167:47:28        .03        .04               936
 FA A <=WAITING FOR WORK=>                   .00        .00                 2
 BG 0 <=WAITING FOR WORK=>                   .00        .00                 2
PF1=HELP      2=PART.BAL.   3=END        4=RETURN     5=DYN.PART   6=CPU
```

Figure 15-3 Display system activity

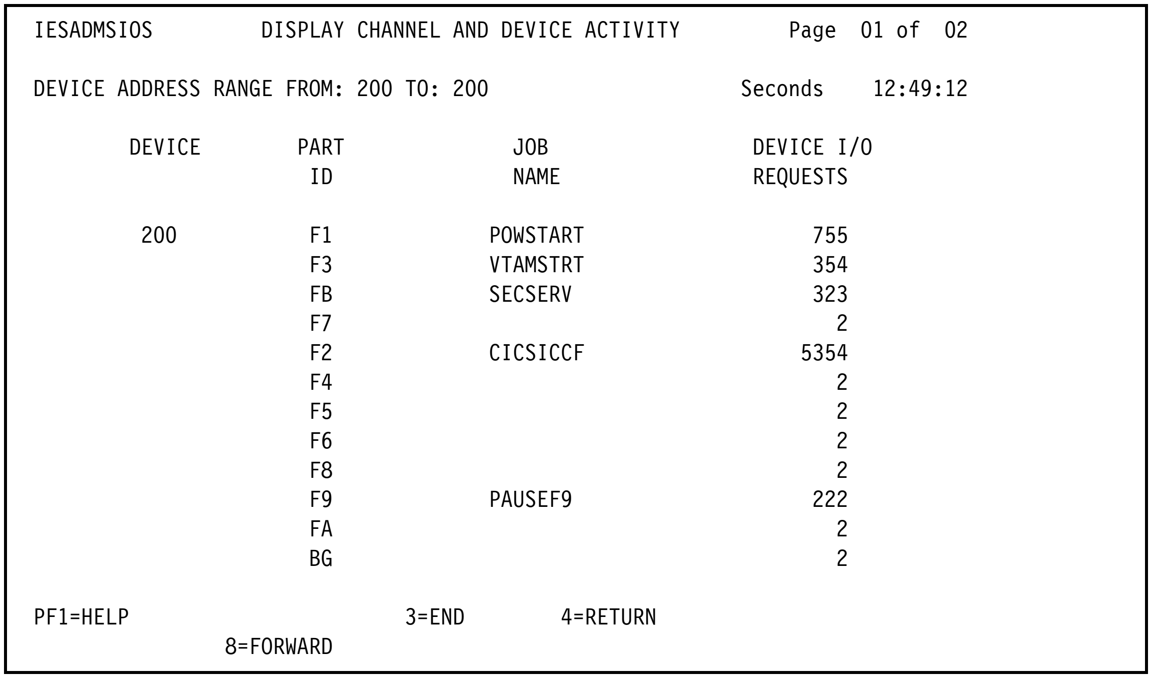
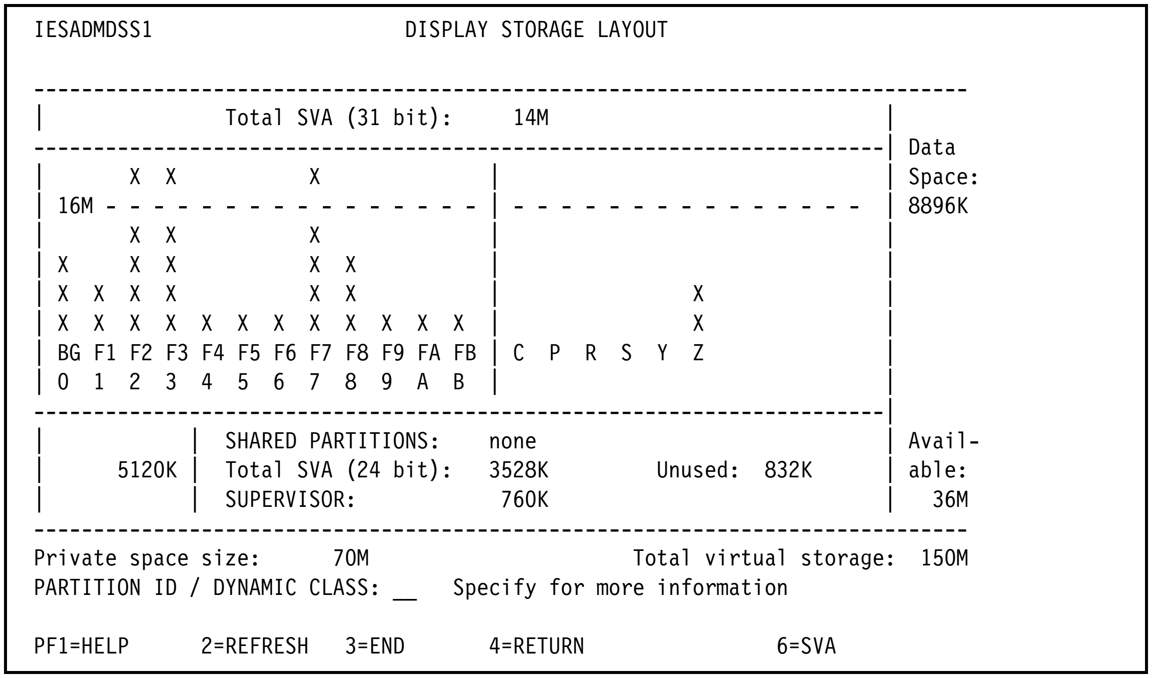

```
IESADMSIOS        DISPLAY CHANNEL AND DEVICE ACTIVITY        Page  01 of  02

DEVICE ADDRESS RANGE FROM: 200 TO: 200                     Seconds    12:49:12

        DEVICE        PART             JOB              DEVICE I/O
                       ID              NAME             REQUESTS

         200           F1            POWSTART                755
                       F3            VTAMSTRT                354
                       FB            SECSERV                 323
                       F7                                      2
                       F2            CICSICCF               5354
                       F4                                      2
                       F5                                      2
                       F6                                      2
                       F8                                      2
                       F9            PAUSEF9                 222
                       FA                                      2
                       BG                                      2

PF1=HELP                      3=END        4=RETURN
                8=FORWARD
```

Figure 15-4 Display channel and device activity

The dialogs Display System Activity (Figure 15-3 on page 384) and Display Channel and Device Activity (Figure 15-4) allow users such as you or the administrator to see and understand what is happening in the system. Each dialog interactively displays general information about current system activity. This includes:

- CPU use
- Paging
- I/O activity
- Status of CICS tasks

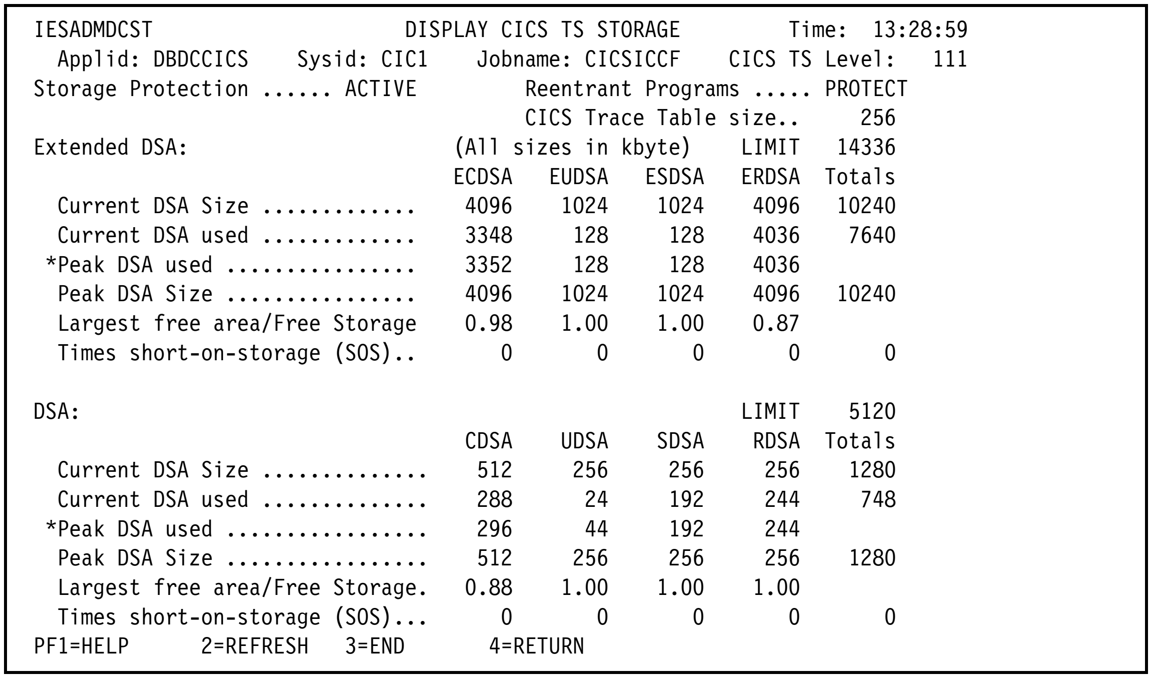
The Display Storage Layout dialog allows users such as you or the administrator to view the partition and SVA layout of the active system (see Figure 15-5). This information is of particular interest to the system administrator.



```
IESADMDSS1                     DISPLAY STORAGE LAYOUT

------------------------------------------------------------------------------
|               Total SVA (31 bit):     14M                             |
-----------------------------------------------------------------------| Data
|       X  X           X              |                                | Space:
| 16M - - - - - - - - - - - - - - - - | - - - - - - - - - - - - - - -  | 8896K
|       X  X           X              |                                |
| X     X  X           X  X           |                                |
| X  X  X  X           X  X           |                X               |
| X  X  X  X  X  X  X  X  X  X  X  X  |                X               |
| BG F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB | C  P  R  S  Y  Z               |
| 0  1  2  3  4  5  6  7  8  9  A  B  |                                |
-----------------------------------------------------------------------|
|            |  SHARED PARTITIONS:    none                             | Avail-
|      5120K |  Total SVA (24 bit):   3528K         Unused:  832K      | able:
|            |  SUPERVISOR:            760K                            |   36M
------------------------------------------------------------------------------
Private space size:      70M                      Total virtual storage:  150M
PARTITION ID / DYNAMIC CLASS: __   Specify for more information

PF1=HELP      2=REFRESH   3=END       4=RETURN                6=SVA
```

Figure 15-5 Display storage layout

The last dialog, Display CICS TS Storage, helps to control the CICS storage usage (see Figure 15-6).



```
IESADMDCST                    DISPLAY CICS TS STORAGE        Time:  13:28:59
  Applid: DBDCCICS    Sysid: CIC1    Jobname: CICSICCF    CICS TS Level:   111
Storage Protection ...... ACTIVE         Reentrant Programs ..... PROTECT
                                         CICS Trace Table size..     256
Extended DSA:                      (All sizes in kbyte)    LIMIT   14336
                                   ECDSA   EUDSA   ESDSA   ERDSA  Totals
  Current DSA Size ............     4096    1024    1024    4096   10240
  Current DSA used ............     3348     128     128    4036    7640
 *Peak DSA used ...............     3352     128     128    4036
  Peak DSA Size ...............     4096    1024    1024    4096   10240
  Largest free area/Free Storage    0.98    1.00    1.00    0.87
  Times short-on-storage (SOS)..       0       0       0       0       0

DSA:                                                       LIMIT    5120
                                    CDSA    UDSA    SDSA    RDSA  Totals
  Current DSA Size .............     512     256     256     256    1280
  Current DSA used .............     288      24     192     244     748
 *Peak DSA used ................     296      44     192     244
  Peak DSA Size ................     512     256     256     256    1280
  Largest free area/Free Storage.   0.88    1.00    1.00    1.00
  Times short-on-storage (SOS)...      0       0       0       0       0
PF1=HELP      2=REFRESH   3=END       4=RETURN
```

Figure 15-6 Display CICS TS storage

15.5 Installing and maintaining the system

As discussed in 3.8, “Additional software products for z/VSE” on page 103, a z/VSE system consists of base and optional products, as well as other IBM licensed products and software from ISVs. All of these products must be installed to make up the customer's z/VSE system. To ease this task, a special installation process has been developed for the z/VSE base products, which installs all base products at once. This installation can be an *initial installation* where the z/VSE base is installed newly, or can be done via a *fast service upgrade* (FSU), where the existing customization is preserved. Installation of additional products is supported via dialogs.

Once a z/VSE system is installed completely, it needs to be maintained to keep it running.

System maintenance describes the tasks of installing corrective and preventive service. Corrective service fixes problems you found on your system, whereas preventive service installs known fixes found elsewhere to avoid running into already known problems. Preventive service is essential to avoid disruption of the running system and needs to be planned thoroughly. You have to develop your own preventive maintenance strategy according to your own needs considering the following aspects:

- Overall system complexity
- Workload and size of your system
- Growth of transaction rates, batch workload, and file sizes
- Change activities for hardware and software

All maintenance activities are controlled by the maintain system history program (MSHP) and are documented in the system history file. Service installation tasks are supported by dialogs as well as displaying history information.

The means for corrective and preventive maintenance of z/VSE are illustrated in Figure 15-7.

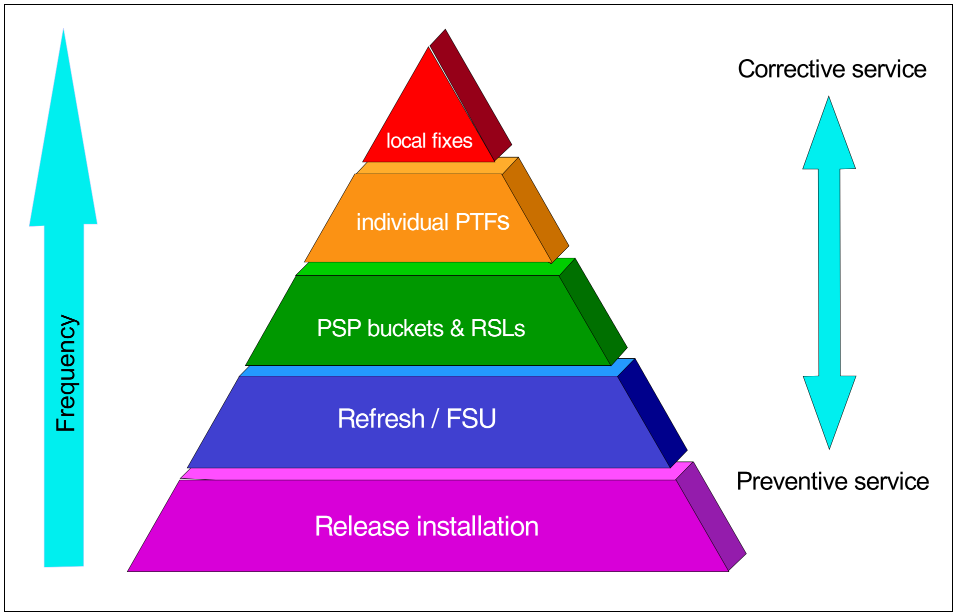


Figure 15-7 Maintenance concept

The different levels of the pyramid in Figure 15-7 are explained below:

- Release

  At the bottom of the pyramid you find the release installation. A release is a complete set of install tapes including base and optional products, which are used to install a new release of z/VSE. A new release contains new SW functions and new HW support, but has also integrated all previous service available until a certain cutoff date. Releases are tested intensively and are recordable via sales channels until a new refresh level or new release becomes available.

- Refresh/Fast service upgrade (FSU)

  For a given release, refreshes are normally provided every 6–12 month. A refresh is also a complete set of install tapes including base and optional products, which are typically installed via FSU. A refresh has applied all PTFs that are available until a certain cutoff date and has undergone an intensive regression test. A refresh is orderable via sales channels until a new refresh level or new release becomes available.

- PSP

  For each refresh level, a preventive service planning (PSP) bucket is maintained daily that lists all high impact or pervasive (HIPER) APARs[2] and the resolving PTF numbers. A z/VSE PSP bucket is divided into subsets for the individual base and optional products. PTFs listed in a PSP can be ordered via IBM service or via the Internet, are delivered as PTF tapes or as a downloadable file, and should be installed with the apply PTF dialog, as shown in Figure 4-7 on page 121.

- RSL

  A recommended service level (RSL) for z/VSE is defined bi-monthly and contains all available PTFs at a specified cutoff date. After cutoff a RSL is monitored 6–8 weeks for PTFs in error (PEs). RSLs are published via the z/VSE home page and are provided via special PSPs. Distribution and installation is the same as for PSPs.

- PTF

  A program temporary fix (PTF) is the official IBM solution for a code error. It is called temporary because the final solution is integrated into the code of the next release. Single PTFs correct an actual defect or problem. A PTF solves one or multiple APARs and can be ordered via the Internet or from IBM service for distribution on tape or for download. Since code errors are corrected on a certain service level, a PTF may require additional requisite PTFs as well. The standard PTF installation process takes care that a PTF is installed on the correct level.

- Local fixes

  A code error is documented with an APAR and may be fixed temporarily (until a PTF is available) with a local fix. A local fix is provided by an IBM support center as a MSHP correct job while a PTF is not yet available. A local fix will usually be replaced by a PTF at a later time.

[2] APAR is defined as authorized program analysis report.

Because an error correction normally requires the failing part to be on the latest service level, it should be the goal for preventive service to make sure that the number of PTFs to apply is kept small to solve a sudden and unexpected defect. This avoids an extensive upgrade having to be done under pressure.

Preventive service recommendations

Some recommendations are:

- Install the latest refresh every 12 months, but not later than 24 months. In addition, order the newest PSP bucket shortly before upgrading to install the actual HIPER service as well.
- Install the latest RSL every 6 months, but not later than 12 months. As before, additionally order the newest PSP bucket shortly before upgrading to install the actual HIPER service as well.
- Install the PSP bucket every 3 months, but not later than 6 months. Order the hardware PSP bucket when installing new hardware.

15.6 Summary

The role of the z/VSE system programmer is to install, customize, and maintain the operating system.

The system programmer must understand the following areas (and more):

- System customization
- System performance
- I/O device configuration
- Operations
- System maintenance

System start-up or IPL and controlling the system status is introduced with the topics:

- IPL and system startup
- Displaying the system status

Key terms in this section		
DOSRES	SYSWK1	MSHP
IJSYSRS	PRD1	PRD2
IPL	$IPLESA.PROC	Status dialogs
System library	ASI	SDL
Refresh	PSP	PTF

15.7 Questions for review

1. What are usually the Volume Identifiers (VOLID) of the volumes where a z/VSE system is installed?
2. In which libraries reside the z/VSE system code?
3. What is the name of the system library (sublibrary)?
4. How is the system startup process referred to?
5. Which dialogs are provided to monitor the system status?
6. Why is preventive maintenance important?
7. What are the different maintenance deliverables?

15.8 Exercises

1. On your system, find out the IPL device (DOSRES) address and then IPL.
2. On the command line of the console enter `IPL` and press F7 to redisplay the IPL statements and messages.
3. Using the z/VSE System Control Statements and z/VSE Messages and Codes manual, discuss the statements and messages shown on the console during IPL.
4. Use the DISPLAY CHANNEL AND DEVICE ACTIVITY dialog (fastpath 362) to verify by which partitions the devices are used.
5. The dialog fastpath 45 leads you to the RETRACE HISTORY FILE panel. Run different retraces and compare the output.

16

Security on z/VSE

Objective: In working with z/VSE, you need to understand the importance of security and the facilities utilized by z/VSE to implement it. An installation's data and application programs are among its most valuable resources. They must be protected from unauthorized access both internally (employees) and externally (customers, business partners, and hackers).

After completing this chapter, you will be able to:

- Explain security concepts.
- Explain BSM and its interface with the operating system.
- Explain the importance of change control.
- Explain the concept of risk assessment.

16.1 Why security?

Over time, it has become much easier to create and access computerized information. No longer is system access limited to a handful of highly skilled programmers. Information can now be created and accessed by almost anyone who takes a little time to become familiar with the newer, easier-to-use, high-level inquiry languages.

More and more people are becoming increasingly dependent on computer systems and the information they store in these systems. As general computer literacy and the number of people using computers has increased, the need for data security has taken on a new measure of importance. No longer can the installation depend on keeping data secure simply because no one knows how to access the data.

Further, making data secure not only means making confidential information inaccessible to those who should not see it—it also means preventing the inadvertent destruction of files by people who may not even know that they are improperly manipulating data.

An operating system provides the facilities to make a system secure. But how much security and how efficient security is established is always the responsibility of the organization.

16.2 Security facilities of z/VSE

In the following sections we cover the facilities of z/VSE that provide its high level of security.

Data about customers is a valuable resource that could be sold to competitors. So the aim of any security policy is to provide users with only their required level of access and to deny unauthorized users access. This is one reason why auditors prefer that users or groups are granted specific access, rather than using universal access facilities. The traditional focus of mainframe security was to focus on stopping unauthorized people from logging on to the system, and then ensuring that users were only allowed access to data on a need-to-know basis. As mainframes have been connected to the Internet, additional security has been required.

However, the main threat to company data has always been from within. An employee within a company has a much better chance of obtaining data than someone outside. A well-thought-out security policy is always the first line of defense.

Further, z/VSE provides a number of facilities to minimize intentional or accidental damage from other users. Many installations run several copies of CICS systems. They often do not permit users of one CICS to access resources of other CICS systems or the other CICS at all. This is used to separate test systems from production systems or different production systems from each other. z/VSE security controls can protect the production environment if they are properly configured and prevent a CICS user (either maliciously or accidentally) from impacting important production work.

16.3 Security roles

Usually it is the system programmer who, working with management, decides the overall security policy and procedures. The system administrator assigns user IDs and initial passwords and ensures that the passwords are non-trivial, random, and frequently changed. Because the user IDs and passwords are so critically important, special care must be taken to protect the files that contain them.

There may even be a separate security manager who sets the policies. If so, the system programmer may not have direct responsibility for security, other than advising the security manager about new products. Separation of duties is necessary to prevent any one individual from having uncontrolled access to the system.

16.4 z/VSE security components

z/VSE provides a basic security support. If an installation needs more security functions, it could use a security package from an Independent Software Vendor (ISV) (for example, CA-Top Secret from Computer Associates or Alert from Connectivity Systems Inc. (CSI)). These ISV products are not described here. The focus of this chapter is the IBM-provided security support in z/VSE.

The z/VSE security provisions include:

- Controlling the access of users (user ID and password) to the system
- Tracking the activities that an authorized user can perform on the systems' data files and programs
- Basic security manager (BSM)

This is the primary component of the z/VSE security. It works closely with z/VSE to protect its vital resources.

The topic of security can be a whole course by itself. In this textbook, we introduce you to the BSM component and show how its features are used to implement z/VSE security.

16.4.1 BSM

Access, in a computer-based environment, means the ability to do something with a computer resource (for example, use, change, or view something). Access control is the method by which this ability is explicitly enabled or restricted. Computer-based access controls are called *logical access controls*. These are protection mechanisms that limit users' access to information to only what is appropriate for them.

Logical access controls are often built into the operating system, or may be part of the logic of application programs or major utilities, such as database management systems. They may also be implemented in add-on security packages that are installed into an operating system. Such packages are available for a variety of systems, including PCs and mainframes. Additionally, logical access controls may be present in specialized components that regulate communications between computers and networks.

Basic Security Manager (BSM) is part of the z/VSE operating system to provide basic security for a mainframe system similar to the Resource Access Control Facility (RACF) on z/OS, the large mainframe operating system. Other than into z/OS, z/VSE differentiates between batch security and online security. Online security protects sign-on to CICS applications and access to CICS resources. Batch security protects batch jobs, VSE files, and VSE libraries. The customer can activate it according his needs.

BSM protects resources by granting access only to authorized users of the protected resources. BSM retains information about users, resources, and access authorities in special structures called *profiles* in its database, and it refers to these profiles when deciding which users should be permitted access to protected system resources. The authorization can be specified explicitly by specifying the user ID at this resource profile or implicitly by specifying a group including this user to the profile. The user ID specification overrules the group specification at the resource profile.

To accomplish its goals, BSM gives you the ability to:

- Identify and authenticate users.
- Authorize users to access protected resources.
- Log and report various attempts of unauthorized access to protected resources.

- Allow applications to use the macros for security requests (RACROUTE).

Figure 16-1 shows a simple view of BSM functions.

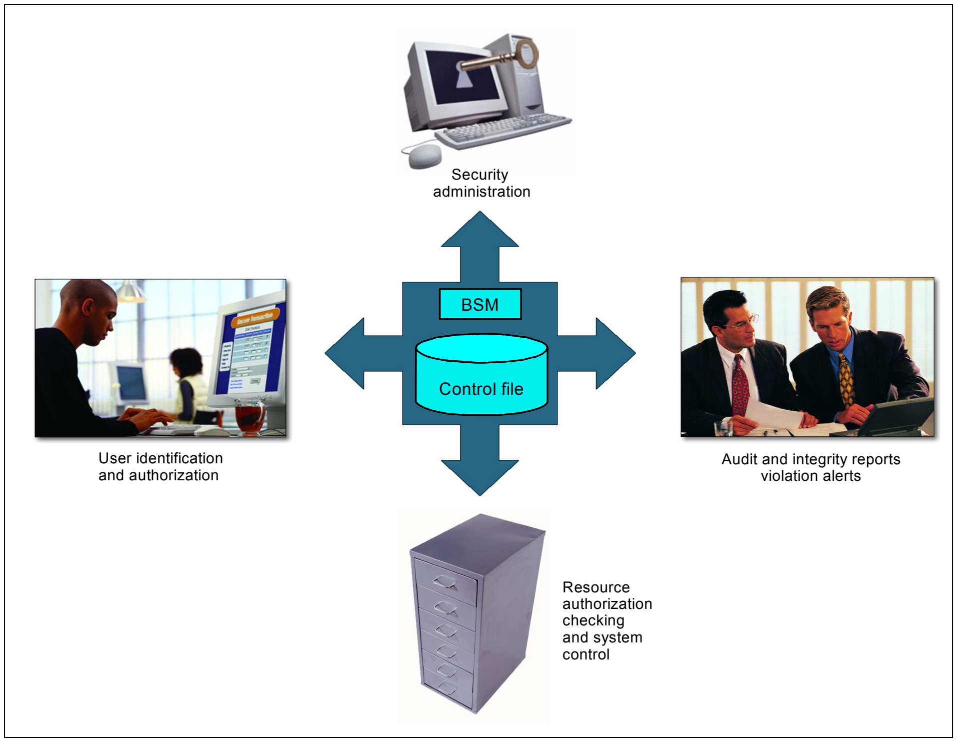

Security administration
BSM
Control file
User identification and authorization
Audit and integrity reports violation alerts
Resource authorization checking and system control

Figure 16-1 Overview of BSM functions

BSM uses a user ID and a password to perform its user identification and verification. The user ID identifies the person to the system as a BSM user. The password verifies the user's identity. Exits could be used to enforce a password policy such as lack of repeating characters or adjacent keyboard letters, and also the use of numerics as well as letters. Popular words such as *password* or the use of the user ID are often banned. General things like setting the minimum password length can be done by command.

The other important policy is the frequency of password change. If a user ID has not been used for a long time, it may be revoked and special action is needed to use it again. When someone leaves a company, there should be a special procedure that ensures that the user IDs are deleted from the system.

16.4.2 System Authorization Facility

The system authorization facility (SAF) was ported from the z/OS operating system. It is part of the z/VSE operating system and provides the interfaces to the callable services provided to perform authentication, authorization, and logging.

SAF does not require any other product as a prerequisite, but overall system security functions are greatly enhanced and complemented if it is used concurrently with BSM. The key element in SAF is the SAF router. This router is always present, even when BSM is not present.

The SAF router provides a common focal point for all products providing resource control. This focal point encourages the use of common control functions shared across products and across systems. The resource managing components and subsystems call the SAF router as part of certain decision-making functions in their processing, such as access-control checking and authorization-related checking. These functions are called *control points*.

The system authorization facility conditionally directs control to BSM (if BSM is present), or to a user-supplied processing routine, or both, when receiving a request from a resource manager via RACROUTE macro calls.

16.5 Security administration

Data security is the protection of data from accidental or deliberate unauthorized disclosure, modification, or destruction. Based on this definition, it is apparent that all data-processing installations have at least potential security or control problems. Users have found, from past experience, that data security measures can have a significant impact on operations in terms of both administrative tasks and demands made on the end user.

BSM gives the user defined with the ADMINISTRATOR attribute (the security administrator or type 1 user) many responsibilities at the system level. The security administrator is the focal point for planning security in the installation and needs to:

- Determine which BSM functions to use.
- Identify the level of BSM protection.
- Identify which data BSM is to protect.
- Identify administrative structures and users.

16.5.1 BSM with middleware

Major subsystems such as CICS use the facilities of BSM to protect transactions and files. Much of the work to configure BSM profiles for this subsystem is done by the CICS system programmers. So there is a need for people in these roles to have a useful understanding of BSM and how it relates to the software they manage.

16.6 Summary

Making data secure does not mean just making confidential information inaccessible to those who should not see it. It means preventing the inadvertent destruction of files by people who may not even know that they are improperly manipulating data. Without better awareness of good data security practices, technology evolution could result in a higher likelihood of unauthorized persons accessing, modifying, or destroying data, either inadvertently or deliberately. The z/VSE security components are a set of features that provide security implementation.

The system authorization facility is part of the z/VSE operating system and provides the interfaces to the callable services provided to perform authentication, authorization, and logging.

The Basic Security Manager (BSM) is a component of z/VSE and controls access to all protected z/VSE resources. BSM protects resources by granting access only to authorized users of the protected resources and retains information about the users, resources, and access authorities in specific profiles.

BSM provides the tools and databases to allow program products such as CICS and DITTO to check and verify a user’s access level and thus permit or deny the use of data sets, transactions, or applications.

BSM enables the organization to define individuals and groups who use the system BSM protects. For example, for a salesman in the organization, a security administrator uses BSM to define a user profile that defines the salesman’s user ID, initial password, and other information.

To accomplish its goals, BSM gives you the ability to:

- Identify and authenticate users.
- Authorize users to access the protected resources.
- Log and report attempts of unauthorized access to protected resources.
- Control the means of access to resources.

Key terms in this chapter		
Basic Security Manager (BSM)	User ID	Password
System administrator	Security policy	Logging
Online security	Batch security	Reporting

16.7 Questions for review

To help test your understanding of the material in this chapter, complete the following questions:

1. Is the following statement true or false?

   Access information in the resource profiles can be set only at group level. This means that it is impossible for a single user to have the update attribute to a specific resource if the BSM group to which the user is connected has only the read attribute.

2. If a VSE system is used as a CICS transaction server only, what is the minimum security that should be established?

   a. Batch security
   b. Online security

16.8 Topics for further discussion

1. On other platforms, how do you protect data sets or files? Is there a way to prevent the execution of a specific application?
2. BSM enables you to assign the security administrator attribute to users. With this, it is possible to access all resources. Discuss the pros and cons.

16.9 Exercises

1. Try to find out whether the BSM or an ISV security manager is active.

   Use the console command SIR. Check the line starting with `'SEC. MGR. ='`.

2. If BSM is active, can you see whether batch security or online security is active?

3. Execute the BSTADMIN program to check for which resource classes the protection is active:

```
// JOB BSTADMIN
// ID USER=…,PWD=…    IT MUST BE RUN WITH AN ADMINISTRATOR ATTRIBUTE
// EXEC BSTADMIN
   STATUS
   END
```

17

Network communications on z/VSE

Objective: In working with z/VSE network communications, you will need to interact with TCP/IP and SNA networks.

- How various communication network models compare with each other
- How the SNA subarea and APPN network topology compare
- How the IP network can be used to transport data between SNA applications
- Commonly used TCP/IP and VTAM commands

17.1 Communications in z/VSE

Network communications has both software and hardware aspects, and a separation of software and hardware communications duties is common in large enterprises. A skilled network expert, however, needs to understand both aspects.

As a system programmer, the network professional must bring a thorough understanding of z/VSE communications software to any project that involves working with the company's network. While network hardware technicians have specific skills and tools for supporting the physical network, their expertise often does not extend to the z/VSE communications software. When a nationwide retail chain opens a new store, the z/VSE system programmers and network hardware technicians must coordinate their efforts to open the new store.

17.2 Brief history of data networks

Established in 1969, TCP/IP is five years older than System Network Architecture (SNA). However, SNA was immediately made available to the public, while TCP/IP was limited at first to military and research institutions, for use in the interconnected networks that formed the precursors to the Internet.

In addition, SNA was designed to include network management controls not originally in TCP/IP through Synchronous Data Link Control (SDLC) protocol. In the 1980s, SNA was widely implemented by large corporations because it allowed their IT organizations to extend central computing capability worldwide with reasonable response times and reliability. For example, widespread use of SNA allowed the retail industry to offer new company credit card accounts to customers at the point-of-sale.

In 1983, TCP/IP entered the public domain in Berkeley BSD UNIX. TCP/IP maturity, applications, and acceptance advanced through an open standards committee, the Internet Engineering Task Force (IETF), using the Request-For-Comment (RFC) mechanism.

The term *internet* is used as a generic term for a TCP/IP network and should not be confused with the *Internet*, which consists of the large international backbone networks connecting all TCP/IP hosts that have links to the Internet backbone.

TCP/IP was designed for interconnected networks (an Internet) and seemed to be easier to set up, while SNA design was hierarchical with the *centralized* mainframe being at the top of the hierarchy. The SNA design included network management, data flow control, and the ability to assign *class of service* priorities to specific data workloads.

Communication between autonomous SNA networks became available in 1983. Before that, SNA networks could not talk to each other easily. The ability of independent SNA networks to share business application and network resources is called SNA Network Interconnect (SNI).

17.2.1 SNA and TCP/IP on z/VSE

System Network Architecture (SNA) was developed by IBM. SNA enabled corporations to communicate among its locations around the country. To do this, SNA included products such as Virtual Telecommunication Access Method (VTAM), Network Control Program (NCP), and terminal controllers, as well as the synchronous data link control (SDLC) protocol. What TCP/IP and the Internet were to the public in the 1990s, SNA was to large enterprises in the 1980s.

Transmission Control Protocol/Internet Protocol (TCP/IP) is an industry-standard, nonproprietary set of communications protocols that provides reliable end-to-end connections between applications over interconnected networks of different types. TCP/IP was widely embraced when the Internet came of age because it permitted access to remote data and processing for a relatively small cost. TCP/IP and the Internet resulted in a proliferation of small computers and communications equipment for chat, e-mail, conducting business, and downloading and uploading data.

Large SNA enterprises have recognized the increased business potential of expanding the reach of SNA-hosted data and applications to this proliferation of small computers and communications equipment in the customers' homes, small offices, and so on.

17.2.2 Layered network models

TCP/IP and SNA are both layered network models. Each can indirectly map to the international *Open Systems Interconnect* (OSI) network model (Figure 17-1).

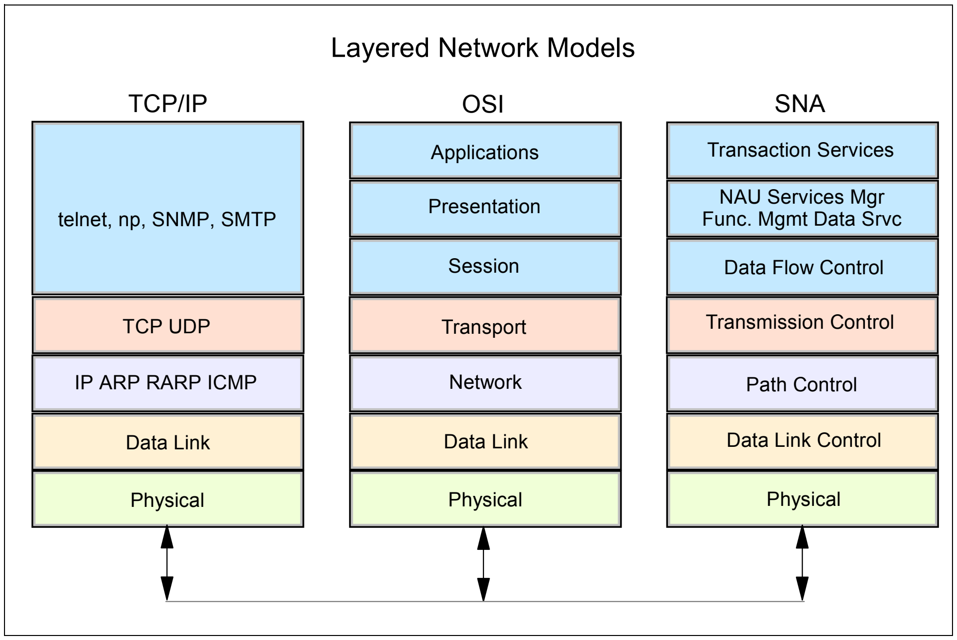


Figure 17-1 Open Systems Interconnect (OSI) network model

The OSI network model depicts the organization of the individual elements of technology involved with end-to-end data communication. As shown in Figure 17-1, the OSI network model provides some common ground for both SNA and TCP/IP. While neither technology maps directly into the OSI network model (TCP/IP and SNA existed before the OSI network model was formalized), common ground still exists due to the defined model layers.

The OSI network model is divided into seven layers. OSI layer 7 (Application) indirectly maps into the top layers of the SNA and TCP/IP stacks. OSI layer 1 (Physical) and layer 2 (Data Link) map into the bottom layers of SNA and TCP/IP stacks.

In one typical scenario, two geographically separated end-point software applications are connected at each end by a layered network model. Data is sent by one end-point application and received by the other end-point application. The applications can reside on large mainframes, PCs, point-of-sale (POS) devices, ATMs, terminals, or printer controllers. End-points in SNA are called logical units (LUs), while end-points in IP are called application ports (ports for short).

Consider how this model might be used in the network communications for a large chain of grocery stores. Each time a customer pays for groceries at one of the many point-of-sale (POS) locations in a grocery store, the layered network model is used twice:

- The POS application resides at the top of the local layered network stack.
- The application that records details of the sale and authorizes completion resides at the top of a remote layer network stack.

The local network stack might run on a non-mainframe system with attached POS devices, while the remote network stack would quite often run on a mainframe, to handle transactions received from all of the store locations. Method of payment, purchases, store location, and time are recorded by mainframe applications, and authorization to print a sales receipt is returned back through both layered network stacks to complete the sale.

This transactional model is commonly known as a request/server or client/server relationship.

17.2.3 Network reliability and availability

What if the network or attached mainframe for our example grocery store chain were to somehow become unavailable? Most POS systems in use today include the ability to accumulate transactions in an intelligent store POS controller or small store processor. When the outage is corrected, the accumulated transactions can then be sent in bulk to the mainframe.

In the previous example, the recovery of transactions would be essential to preventing bookkeeping and inventory problems at the store and in the chain's central office. The cumulative effect of unaddressed, inaccurate records could easily destroy a business. Therefore, reliability, availability, and serviceability (RAS) are just as important in the design of a network as they are in the mainframe itself.

17.2.4 Factors contributing to the continued use of SNA

SNA is stable, trusted, and relied upon for mission-critical business applications worldwide. A significant amount of the world's corporate data is handled by z/VSE-resident SNA applications[1]. A distinctive strength of SNA is that it is connection-oriented with many timers and control mechanisms to ensure reliable delivery of data.

Mainframe IT organizations are often reluctant and sceptical about moving away from SNA, despite the allure of TCP/IP and Web-based commerce. This

[1] SNA applications running on z/VSE are also known as VTAM applications.

reluctance is often justified. Rewriting stable, well-tuned business applications to change from SNA program interfaces to TCP/IP sockets can be costly and time consuming, and risks negatively impacting response time performance.

Many businesses choose to use Web-enabling technologies to make the vast amount of centralized data available to the TCP/IP-based Web environment, while maintaining the SNA APIs. This *best of both worlds* approach ensures that SNA and VTAM will be around well into the foreseeable future.

17.3 TCP/IP overview

TCP/IP is the general term used to describe the suite of protocols that form the basis for the Internet. It was first included in the UNIX system offered by the University of California at Berkeley, and is now delivered with essentially all network-capable computers in the world.

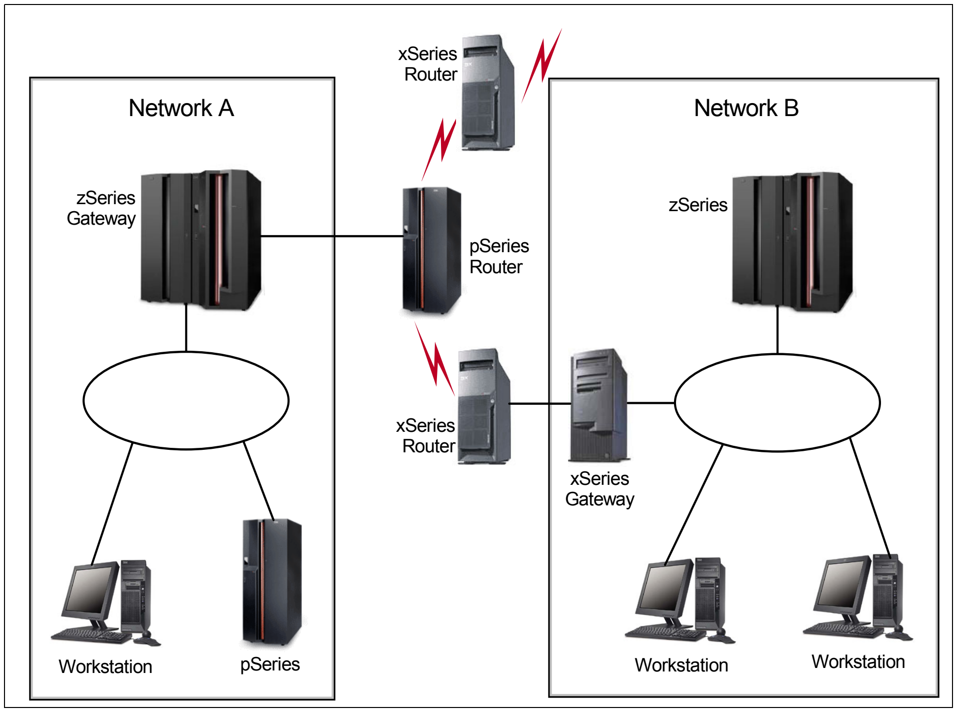


Figure 17-2 TCP/IP introduction

All systems, regardless of size, appear the same to other systems in the TCP/IP network. TCP/IP can be used over Local Area Network (LAN) hardware using most common protocols, and over Wide Area Networks (WANs).

In a TCP/IP network environment, a machine that is running TCP/IP is called a *host*. A TCP/IP network consists of one or more hosts linked together through various communication links. Any host can address all the other hosts directly to establish communication. The links between networks are invisible to an application communicating with a host. TCP/IP can be hosted on mainframes running z/VSE. TCP/IP configuration files are built, activated, and run on the z/VSE host. TCP/IP display commands are entered from the VSE operator console.

17.3.1 Using commands to monitor TCP/IP

z/VSE supports TCP/IP commands such as:

- QUERY
- PING
- TRACERT
- DISCOVER

You can enter these TCP/IP commands from the z/VSE console preceded by the partition ID of the TCP/IP partition.

Example 17-1 shows the results you might see when entering the QUERY LINKS and QUERY CONNECTIONS commands from the z/VSE operator console.

Example 17-1 Sample QUERY output

**** QUERY LINKS output ****

```
ID: OSAX            TYPE: OSAX   DEV: (0500,0501)
MTU:  1492
DRIVER: IPNLOSAX
ROUTER: NONE
AUTORESTART: OFF DELAY: 600 LIMIT: 50
CURRENT STATUS: ACTIVE
```

**** QUERY CONNECTIONS output ****

```
IPN253I << TCP/IP CONNECTIONS >>
IPT353I >4115 9.152.210.83,2386 TCP ESTABLISHED
IPT45I  OPEN BY PHASE $VTMAIN IN R1
IPT341I  START: 09:38:34 END: 09:38:34 DURATION: 0.024
IPT343I  LINKID: OSAX; MTU: 1,492, SEND MSS:
1,452 (1,460), RECV MSS: 1,452, BUFFER: 43,560/10,890
```

If problems are indicated, the z/VSE system programmer can use utilities such as QUERY, PING, and TRACERT to further investigate problems in a network. There are also a number of diagnose commands available, including traces.

17.3.2 Using commands to manage TCP/IP

The initial setup of the network is specified in the ipinit file, which is used by TCP/IP as a configuration file during startup. All network parameters not specified in the ipinit file will be set to their default value.

When TCP/IP is up and running, the settings of the network parameter can be displayed with the QUERY SET command. They may be changed with the SET and other TCP/IP commands.

In general, the TCPIP commands to manage the network include options to:

- Drop specific socket connections.
- Start or stop communication devices.
- Display activity statistics.
- Display configuration information.
- Alter the TCP/IP network configuration.

Related reading: For more information about IP commands see the CSI[2] publication *TCP/IP for VSE Operator Commands.*

17.3.3 Features of TCP/IP for VSE

File Transfer Protocol (FTP)

TCP/IP for VSE gives you the capability to transfer files using FTP. There are various methods available to transfer the files: using a VSE batch program, using CICS, or calling FTP from a VSE application program.

TN3270

When z/VSE hosts existed in a SNA-only environment, dedicated terminal controllers and *dumb* (non-programmable) terminals were the cornerstone of communicating with z/VSE. The data protocol used to communicate between the dumb terminal and z/VSE is called the 3270 data stream (more on this later, when we discuss VTAM). As TCP/IP became a world standard, the 3270 data stream was adapted for use on a TCP/IP network. By blending 3270 with the existing Telnet standard, the Telnet 3270 standard emerged. Today, TN3270 has been further refined into TN3270 Enhanced, or TN3270E. This standard is defined in RFC 2355.

TCP/IP for VSE includes a Telnet daemon that supports the TN3270 protocol. This enables you to log on to VTAM applications (like CICS) directly from your workstation or from another client that is connected to TCP/IP.

[2] Together with z/VSE, the product *TCP/IP for VSE/ESA* is shipped. This is a product of Connectivity Systems, Inc. (CSI).

Line Printer Requester (LPR)

Using the TCP/IP for VSE Line Printer Requester (LPR) client you can print mainframe output or mainframe files on printers that are connected to remote TCP/IP hosts.

E-mail

You can use TCP/IP for VSE to send e-mails using the standard SMTP mechanism. Most e-mail systems can receive e-mails and attachments that use this protocol. There are various methods to run the TCP/IP for VSE e-mail client: using a VSE batch program; using CICS; or running an application program that uses the REXX, Assembler, COBOL, or PL/I socket interface.

General Print Server (GPS)

The General Print Server enables your VTAM-based applications to print on TCP/IP-based printers without modifying the applications. The applications may run under CICS or they may be stand-alone VTAM-applications.

Network File System (NFS)

The Network File System for z/VSE is a package of software programs that allow you to mount a VSE file system to any other computer system that runs an NFS client. This means that your PC workstation, your UNIX server, and even your VM system can access any VSE file system object as if that VSE file is on that client's disk drive.

17.4 VTAM overview

In z/VSE, VTAM provides the SNA layer network communication stack to transport data between applications and the end user. VTAM manages the SNA-defined resources, establishes sessions between these resources, and tracks session activity.

VTAM performs a number of tasks in a network, for example:

- Monitors and controls the activation and connection of resources.
- Establishes connections and manages the flow and pacing of sessions.
- Provides application programming interfaces (for example, an APPC API for LU 6.2 programming) that allow access to the network by user-written application programs and IBM-provided subsystems.
- Provides interactive terminal support for CICS/TS.
- Provides support for both locally and remotely attached resources.

z/VSE runs only one VTAM address space. Each application that uses VTAM, such as CICS/TS, requires a VTAM definition. The application and VTAM use this definition to establish connections to other applications or end users.

Each end point of a VTAM application session is known as a logical unit (LU). Each LU is assigned a unique network addressable unit (NAU) to facilitate communication. An LU is a device or program by which an end user (application program, a terminal operator, or an input/output mechanism) gains access to the SNA network. VTAM-established sessions are known as LU-to-LU sessions. In an SNA network, CICS/TS, for example, is considered an LU and typically has many sessions with other LUs, such as displays, printers, POS devices, and other remote CICS/TS regions.

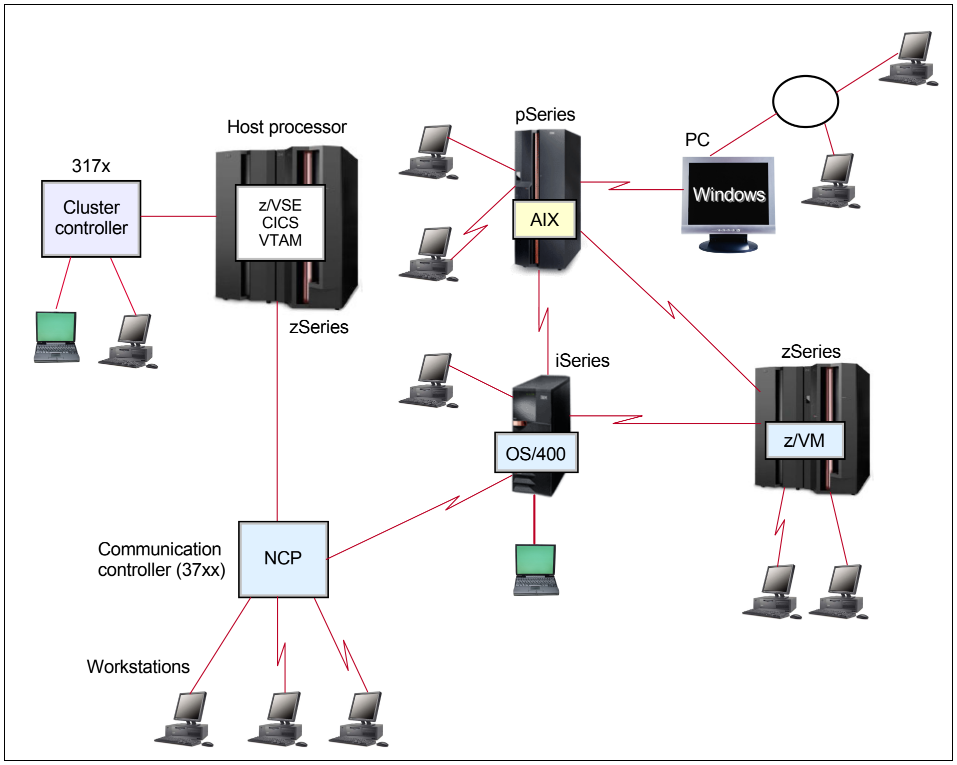


Figure 17-3 VTAM overview - the SNA environment

A physical unit (PU) controls one or more LUs. A PU is not literally a physical device in the network. Rather, it is a portion of a device (usually programming or circuitry, or both) that performs control functions for the device in which it is located and, in some cases, for other devices that are attached to the PU-containing device.

A PU exists in each node of an SNA network to manage and monitor the resources (such as attached links and adjacent link stations) of the node.

The PU exists either within the device or within an attached controlling device. VTAM must activate the PU before it can activate and own each LU attached to the PU.

Even the mainframe is a type of PU, with attached LUs of which CICS/TS is an example. There are three types of PUs:

- PU Type 5 is in the mainframe.
- PU Type 4 is a wide-area network communication controller (CC).
- PU Type 2 is a peripheral communication controller (CC). These can be directly attached to the mainframe or to a PU Type 4.

17.4.1 Network topologies supported by VTAM

Although TCP/IP is by far the most common way to communicate over a network with a z/VSE host, some environments still use native SNA, and many environments now carry (encapsulate) SNA traffic over UDP/IP. The hierarchical design of SNA serves the centralized data processing needs of large enterprises. At the top of this hierarchy is VTAM. VTAM serves the following types of network topologies:

- Subarea
- Advanced Peer-to-Peer Network (APPN)
- Subarea/APPN mixed

The part of VTAM that manages a subarea topology is called System Services Control Point (SSCP). The part of VTAM that manages APPN topology is called Control Point (CP).

VTAM subarea networks predate APPN. In many large enterprises, migration of subarea networks to an APPN topology is a desired technical objective. VTAM administration and required coordination between communication hardware and software personnel can be significantly reduced with a pure APPN topology as a result of its increased flexibility over subarea networks.

All three VTAM configuration types (subarea, APPN, and mixed subarea/APPN) exist throughout the world’s large enterprises.

17.4.2 What is a subarea network topology?

The distinguishing characteristics of a VTAM subarea network include the ownership and sharing of SNA resources. A subarea is a collection of SNA resources controlled by a single VTAM address space.

A single VTAM and the SNA resources it owns is called a *domain.* A cross-domain resource manager (CDRM) allows for communication between VTAMs in the SNA network. When an LU requests a session to be established with an LU in a separate VTAM domain, the VTAMs cooperate to establish a *cross-domain session.*

Figure 17-4 shows a pure VTAM subarea network. This diagram might, for example, be representative of a business that is based in New York City with a large presence in Los Angeles, and a later expansion into Chicago. Figure 17-4 includes three VTAM domains and six subareas. The Chicago subarea then becomes part of the domain of its controlling VTAM.

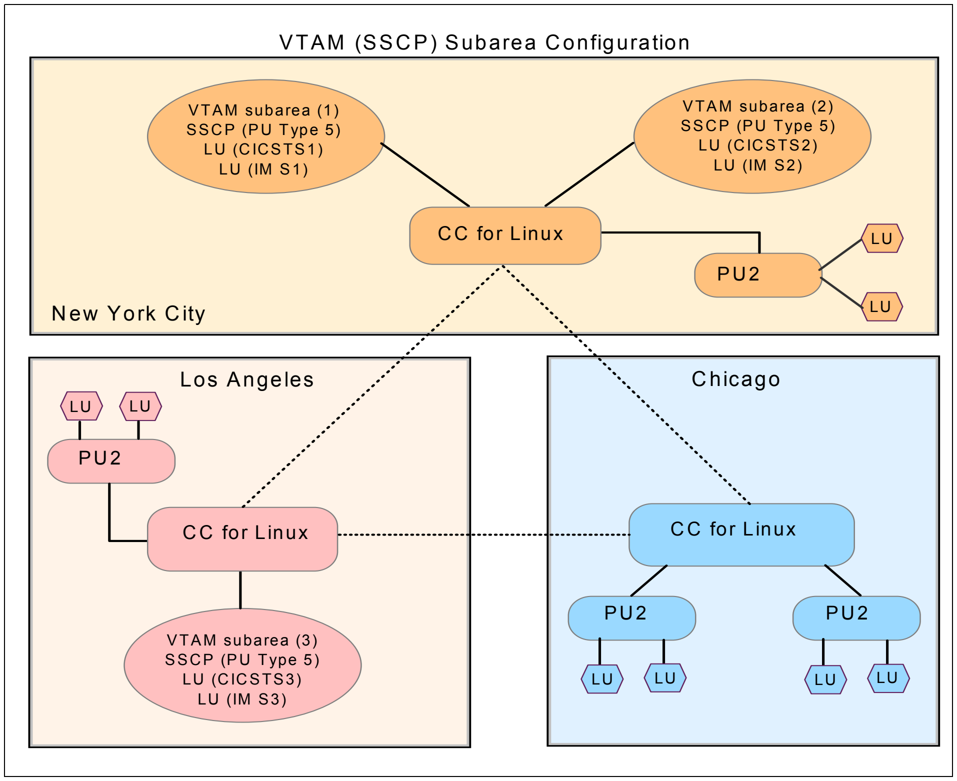

VTAM (SSCP) Subarea Configuration
VTAM subarea (1) SSCP (PU Type 5) LU (CICSTS1) LU (IM S1)
VTAM subarea (2) SSCP (PU Type 5) LU (CICSTS2) LU (IM S2)
CC for Linux
PU2
LU
LU
New York City
Los Angeles
LU
LU
PU2
CC for Linux
VTAM subarea (3) SSCP (PU Type 5) LU (CICSTS3) LU (IM S3)
Chicago
CC for Linux
PU2
PU2
LU
LU
LU
LU

Figure 17-4 A pure VTAM subarea network

In general, full migration from the older subarea topology to an APPN topology is a desired technical objective due to opportunities to leverage a newer IP network infrastructure and the cost reduction associated with elimination of older SNA network equipment. It also simplifies VTAM network management through more dynamic capabilities.

17.4.3 What is an APPN network topology?

Advanced Peer-to-Peer Networking (APPN) is a type of data communications support that routes data in a network between two or more systems that do not need to be directly connected.

APPN topology does not have a subarea number, nor does it have exclusive ownership of the SNA resources. Each APPN-participating VTAM is included in a geographically dispersed collection of shared SNA resources, eliminating the need for a cross-domain resource manager to establish sessions.

17.4.4 Summary of VTAM topologies

VTAM can be a subarea SSCP, an APPN CP, or both SSCP and CP serving a mixed network.

17.4.5 Using commands to monitor VTAM

The list below is a small sampling of VTAM commands used to gather information about a VTAM environment:

- List the status of VTAM resources with DISPLAY(D NET,) commands, for example:
  - D NET,VTAMOPTS

    Displays VTAM startup options.
  - D NET,APPLS

    Displays the status of defined applications (ACBs).
  - D NET,MAJNODES

    Displays the status of VTAM activated ATCCONxx members.
  - D NET,TOPO,LIST=SUMMARY

    Displays APPN topology information.
  - D NET,SESSIONS

    Displays the status of subarea SSCP-SSCP sessions.

  - D NET,CDRMS

    Displays status of subarea cross domain resource managers.
  - D NET,EXIT

    Displays the status of VTAM exit points.
- Activate/deactivate VTAM resources with VARY (V NET,) commands.
- Alter the VTAM environment with the MODIFY (F VTAM,) commands.

z/VSE system programmers use products such as NetView to monitor and report on the status of VTAM resources.

Related reading: For more information about VTAM commands see the IBM publication *VTAM Operation.*

17.4.6 Background: 3270 data stream

What HTML is to a Web application and browser, the 3270 data stream is to an SNA application and device in an LU-LU session. Specialized commands are embedded in the data of display screen devices and printers. The 3270 data stream is data with these embedded instructions and data field descriptors. The 3270 data stream commands are created and read by SNA applications, Physical Unit (PU) controllers managing the displays, and printers, as well as TN3270 emulators available in AIX and PC operating systems.

One of the most notable advantages of the 3270 data stream is that a full screen of data entries and corrections is sent to the receiving SNA application when the Enter key or PF key is pressed.

The 3270 data stream includes column and row addresses of data fields, along with data descriptors such as color, protected screen areas, and unprotected screen areas.

When an SNA application sends data to a display screen, it includes column/row location placement of individual data fields, descriptors of the data fields, and the screen position of the cursor. The 3270 data stream ability to permit completion of data entry before sending to the SNA applications saves the CPU from unnecessary interruptions. Conversely, every key stroke of a VT100 vi session requires CPU attention. When the key stroke Ctrl+G is entered at a VT100, something needs to understand this keystroke so that the status line can be displayed.

The SNA 3270 data stream is critical to the success of the SNA network ability to centrally manage many thousands of geographically dispersed display screens and printers.

17.5 Summary

Enterprise networks can be designed, customized, operated, and supported using combined features and functions of both SNA and TCP/IP network layers using the Communications Server on z/VSE, AIX, Windows, Linux, and Linux on System z.

A significant number of large enterprises use 3270 and SNA applications and have no need to rewrite the business application APIs. As a result, VTAM continues to be supported while integrating it with technologies such as APPN. In addition, TCP/IP uses VTAM for TN3270 sessions.

Key terms in this chapter		
APPN	Layered network model	Internet
LU-to-LU	NCP	OSI
SDLC	SNA	TCP/IP
PU	VTAM	LU
Stack	Internet segment	Subarea

17.6 Questions for review

To help test your understanding of the material in this chapter, complete the following questions:

1. What components are common between the SNA and TCP/IP network layers?
2. Is the majority of the world's corporate data served by z/VSE SNA applications?
3. Does a business need to rewrite SNA business applications to Web-enable the application?
4. What is the difference between an SNA subarea network and APPN topology?
5. Why is APPN topology more desirable than a SNA subarea network?
6. What do HTML and a 3270 data stream have in common?
7. What is common about an IP address and an SNA *network addressable unit* (NAU)?

17.7 Exercises

1. From the z/VSE console, enter the TCP/IP command `QUERY SET`.
   - What is the home IP address?
   - What is the time interval for packet retransmission (retransmit time)?
   - Is this TCP/IP host acting as a gateway?
2. From the z/VSE console, enter the following VTAM commands:

   ```
   D NET,APPLS
   D NET,MAJNODES
   D NET,TOPO,LIST=SUMMARY
   D NET,SESSIONS
   D NET,SESSIONS,LIST=ALL
   ```

   Briefly describe how the output of this command could be useful.
3. From a WINDOWS XP system:
   - Enter `ping your.ip.addr.ess`.
   - Enter `tracert your.ip.addr.ess`.
   - Enter `nslookup`.
     - Enter a WWW address, like www.ibm.com.
     - Exit the nslookup (Nameserver Lookup) (exit).

A

A brief look at IBM mainframe history

This appendix discusses the development of the IBM mainframe from 1964 to the present, as shown in Figure A-1.

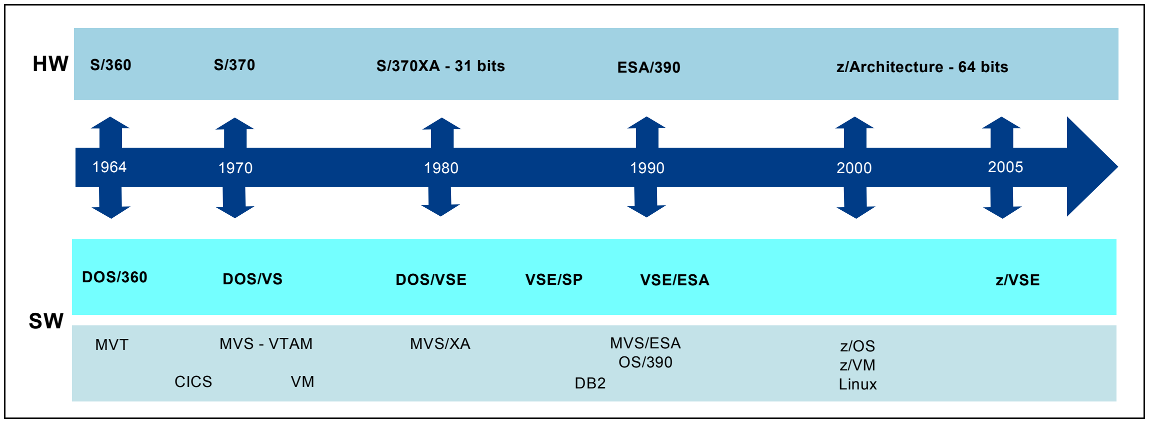


Figure A-1 IBM mainframe time line

On April 7, 1964 IBM introduced System/360, a family of five increasingly powerful computers that ran the same operating system and could use the same 44 peripheral devices. Along with S/360 were also born the I/O subsystem concept (namely defining processors to transfer data between memory and I/O devices) and parallel channels (channels to transmit data in parallel to I/O devices).

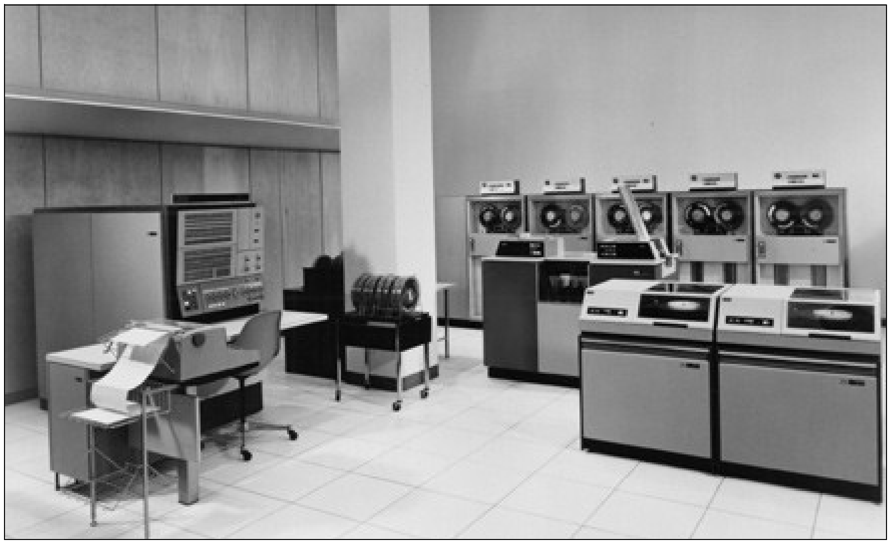


Figure A-2 S/360 Model 30

Together with the earliest S/360s, IBM introduced an new entry operating system (DOS/360) to complement OS/360. DOS/360 focused on the needs of customers who bought the smaller models of the S/360 family, including the S/360 Model 30 and Model 40.

For the first time companies could run mission-critical applications for business on a highly secure platform.

In 1968, IBM introduced Customer Information Control System (CICS). It allowed workplace personnel to enter, update, and retrieve data online. To date, CICS remains one of the industry's most popular transaction monitors.

In 1969, Apollo 11's successful landing on the moon was supported by several System 360s, Information Management System (IMS) 360, and IBM software.

In the summer of 1970, IBM announced a family of machines with an enhanced instruction set, called System/370. Through the 1970s the machines got faster and more capable. The 370 Model 145 was the first computer with fully integrated monolithic memory (circuits in which all of the same elements (resistors, capacitors, and diodes) are fabricated on a single slice of silicon) and 128-bit bipolar chips. More than 1,400 microscopic circuit elements were etched onto each one-eighth-inch-square chip.

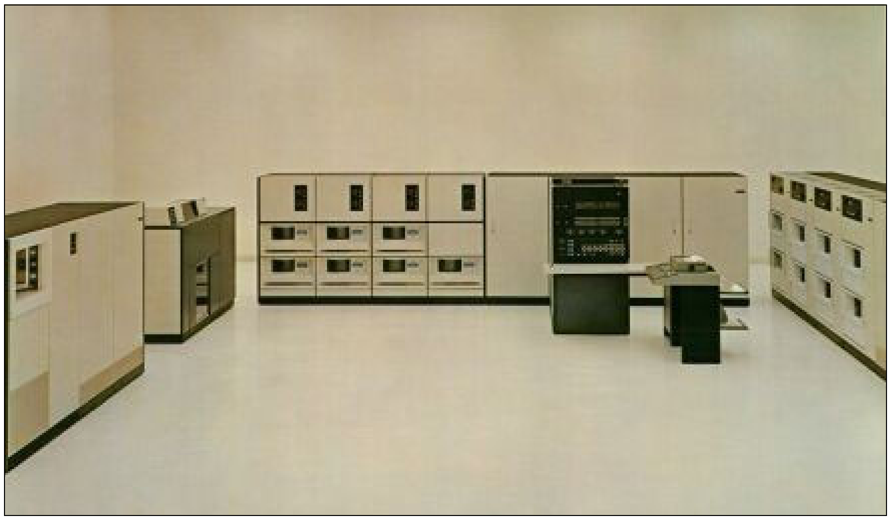


Figure A-3 S370 Model 145

Able to run System/360 programs, thus easing the upgrade burden for customers, System/370 was also one of the first lines of computers to include *virtual memory* technology. Virtual memory is a technique developed to expand the capabilities of the computer by using space on the hard drive to accommodate the memory requirements of software. DOS/360 was enhanced to exploit the new virtual memory technology and became known as DOS/VS.

1979 saw the introduction of the IBM 4300 family. The entry 4300 processor, the 4331, offered a S/370 compatible processor in a more compact, less expensive form factor. The 4331 offered standard channels and I/O. Another option was cheap, integrated I/O adapters (disk, TP, and so on). The 4300 was designed using Large Scale Integration (LSI) technology and solid state memory based on a 64K bit chip. At the time of the 4300 series, DOS/VS became known as DOS/VSE. DOS/VSE added Fixed Block Architecture (FBA). FBA is a simple, low-cost disk architecture. Over time, FBA was largely replaced by Extended Count-Key-Data (ECKD) architecture. Later, FBA formed the basis for SCSI disk support in z/VSE Version 3 in 2005.

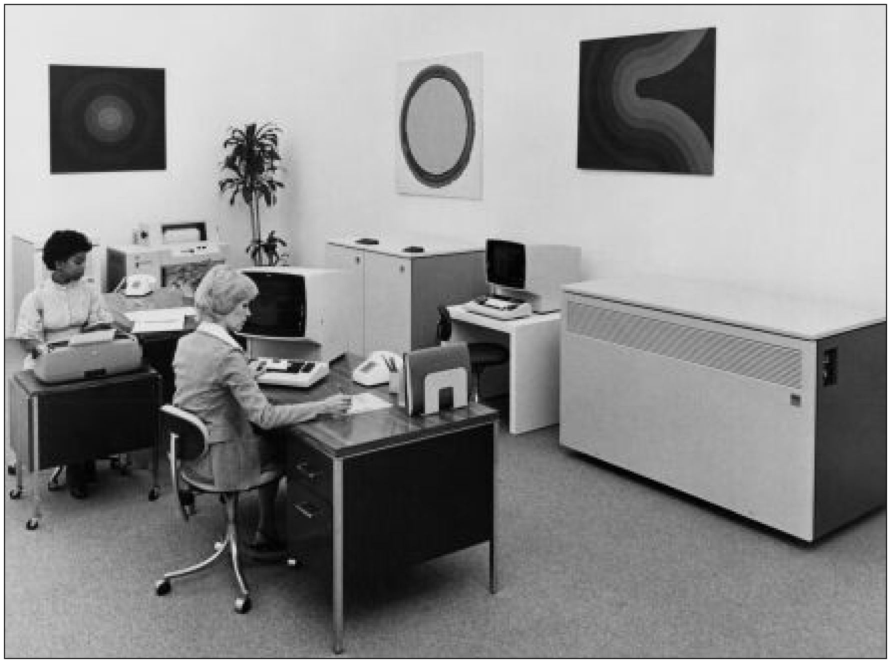


Figure A-4 4331 system (on the right against the wall)

Around 1982, addresses were extended from 24 bits to 31 bits (370XA). However, DOS/VSE was not enhanced to provide 370XA support and remained essentially a 24-bit system. Instead, VSE focused on integration and ease-of-use. DOS/VSE became VSE/SP.

In 1984, IBM announced a 1-megabit Silicon and Aluminum Metal Oxide Semiconductor (SAMOS) chip. Although *mega* means million, the chip actually holds 1,048,576 bits of information in a space smaller than a child's fingernail.

In the mid-to-late 1980s, IBM 3081 and 3090 processors represented the state-of-the-art in high-end mainframes. Both featured Thermal Conduction Modules (TCMs) that reduce space, cooling, and power requirements. The IBM 9370 family was designed to meet the needs of smaller or distributed mainframe users. The 9370 was designed around inexpensive, industry standard packaging based on 19-inch *racks*. The 9370 was popular among VSE/SP users.

In 1990, IBM introduced the ES/9000 family. ES/9000s featured Enterprise Systems Architecture (ESA), an advance beyond 370XA architecture. To exploit ESA and the new ES9000 servers, VSE/SP added 31-bit real and virtual addressing. VSE/ESA replaced VSE/SP. It introduced a number of enhancements designed to help enable VSE workload growth by providing significant constraint relief.

Some industry pundits, however, did not think that the mainframe would survive the early 1990s. They predicted that the rapid growth in personal computers and small servers would render *Big Iron* (industry jargon for mainframe) obsolete. But IBM believed that serious, security-rich, industrial-strength computing would always be in demand, hence System/390. IBM stuck with the mainframe, but reinvented it from the inside, infusing it with an entirely new technology core and reducing its price.

Figure A-5 9672 Parallel Enterprise Server

Complementary Metal Oxide Semiconductor (CMOS)-based processors were introduced into the mainframe environment, replacing the bipolar technology and setting the new direction for modern mainframe technology. CMOS chips required less power than chips using just one type of transistor.

In the same decade, IBM introduced the parallel channel by Enterprise System Connectivity (ESCON) and began the integration of the network adapter to the mainframe, Open System Adapter (OSA).

In 1998 IBM introduced a new module capable of surpassing the 1,000 MIPS barrier, making it one of the world's most powerful mainframes. Also in this period, the concept of logical partition was extended to support 15 partitions.

One year later, IBM introduced the first enterprise server to use IBM innovative copper chip technology. The synergy helped extend customers' ability to handle millions of e-business workload transactions and large-scale Enterprise Resource Planning applications.

FICON, a new fiber optic channel, was introduced with up to eight times the capacity of ESCON channels. Also in 1999, Linux appeared on System/390 for the first time. Many VSE/ESA customers found the combination of VSE and Linux on the same mainframe to be a compelling solution.

In October 2000, IBM announced the first generation of the zSeries mainframes. The zSeries is based on the 64-bit z/Architecture, which is designed to reduce memory and storage bottlenecks. The z/Architecture is an extension of ESA/390. Dynamic channel management was also introduced, as well as specialized cryptographic capability.

The mainframe became *open* and capable of executing Linux. Special processors (IFLs) were developed. The IFLs significantly enhanced the mainframe value proposition and further enhanced the appeal of Linux, especially to VSE/ESA users.

z900 was launched in 2000 and was the first IBM server "designed from the ground up for e-business." The z900 was followed by the z800, a mid-sized server based on z900 technology.

Figure A-6 z800

The z900 was replaced by the z990. The z990 reached 9000 MIPS. The increased scalability was further supported by the increase in the number of logical partitions available from 15 to 30 LPARs. There is still a 256-channel limit per operating system image, but z990 can have 1024 channels distributed in four Logical Channel SubSystems (LCSSs).

Once again, the z990 was followed by the smaller z890 based on the same technology and containing many of the same components. Along with the introduction of the z890, VSE/ESA became z/VSE Version 3.

Figure A-7 z890

The z990 provides a multibook system structure that supports the configuration of one to four *books*. Each book is comprised of a Multiple Chip Module (MCM) with 12 processors, of which eight can be configured as standard processors; memory cards that can support up to 64 GB of memory per book; and high-performance Self-Timed Interconnects. The maximum number of processors available on a z990 is 32.

To support the highly scalable multibook system design, the Channel SubSystem (CSS) has been enhanced with Logical Channel SubSystems (LCSSs), which offers the capability to install up to 1024 ESCON channels across three I/O cages. With Spanned Channel support, HiperSockets, ICB, ISC-3, OSA-Express, and FICON Express can be shared across LCSSs for additional flexibility. High-speed interconnects for TCP/IP communication, known as HiperSockets, allow TCP/IP traffic to travel among partitions and virtual servers at memory speed, rather than network speed.

Figure A-8   z9 BC

The latest generation of mainframes, the IBM System z, includes the z9 Enterprise Class (z9 EC) and smaller z9 Business Class (z9 BC). The z9 EC is the next step in the evolution of the IBM mainframe family. It uses the z/Architecture and instruction set (with some extensions) of the z900 and z990 servers. This architecture, formerly known as ESAME Architecture, is commonly known as 64-bit architecture, although it provides much more than 64-bit capability. The physical appearance of z9 EC and z990 servers is very similar. However, in addition to extending z990 technology, System z9 EC delivers enhancements in the areas of performance, scalability, availability, security, and virtualization.

As before, a year after the introduction of the z9 EC, it was followed by the smaller z9 BC based on the same technology. To exploit z9 EC and z9 BC technology, z/VSE Version 3 became z/VSE Version 4. z/VSE V4 is designed to support z/Architecture and 64-bit real addressing along with innovative technology such as hardware-assisted encryption.

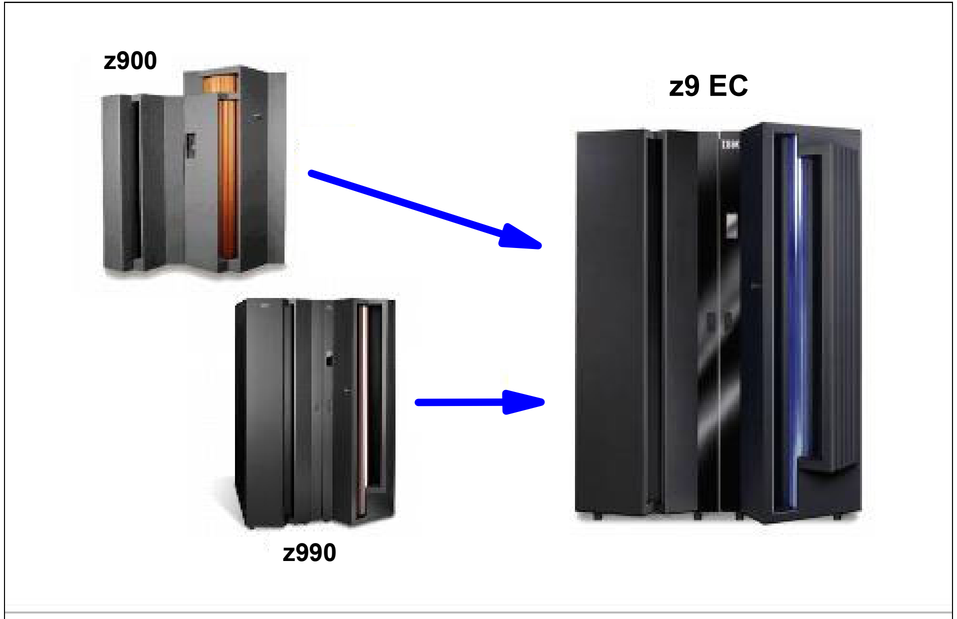


Figure A-9 z9 EC

Examples of further mainframe evolution in the z9 EC include:

- A modular multi-book design that supports one to four books and up to 54 processor units (customer-usable PUs) per server
- Full 64-bit real and virtual storage support, and any logical partition can be defined for 31-bit or 64-bit addressability
- Up to 512 GB of system memory
- Up to 60 logical partitions

In previous generations of mainframes, the number of I/O devices in a system was limited by the number of channels, the number of control units on each channel, and the number of devices on each channel. The addressing structure also provided a limitation. The fixed three-byte addresses (one byte each for channel, control unit, and device) of early systems evolved into four-byte device numbers, allowing up to almost 64K device addresses. The z9 EC server continues this growth by providing Multiple Subchannel Sets (MSS), allowing up to almost 128K device addresses.

Channel performance has grown from parallel channels to ESCON channels to FICON channels. The z9 EC server continues such growth by providing a significantly higher-performance option for channel programming.

Basic *real* systems evolved into virtual systems, and this evolution has extended to systems, processors, memory, I/O devices, LAN interfaces, and so forth. The z9 EC server continues this direction with new instructions that improve the performance of virtual machine QDIO operations. This is done by creating a passthrough architecture designed to reduce host programming overhead, avoiding the stopping of guest processing when adapter interruptions are present.

Cryptographic hardware assistance has been available in many forms on earlier systems, and with much more emphasis in more recent servers. The z9 EC server continues the evolution of cryptographic hardware processing by extending the functions of the basic cryptographic instructions and by consolidating the options (secure coprocessor and accelerator) in a single feature.

Transparent hardware recovery has been a keystone in mainframe design and has evolved in many directions. The z9 EC server continues this evolution by extending such transparent recovery functions to include the paths from I/O cages to system memory.

Concurrent maintenance is a major design goal for modern mainframes and often involves balancing a design between replicated components and more integration onto chips and MCMs. The z9 EC server allows for a single book, in a multi-book configuration, to be concurrently removed and reinstalled during an upgrade or repair.

B

Utility programs

There is no specific definition of what constitutes a z/VSE utility program today, but common usage includes only a few z/VSE-provided programs as utilities. The UNIX community, by contrast, considers many of the standard commands as utilities. This includes compilers, backup programs, filters, and many other types of programs. To the z/VSE community these are applications or programs, not utilities. The difference is simply one of terminology, but it can be confusing to new z/VSE users.

z/VSE utilities are batch programs (although some of them can be used in the CICS foreground with appropriate transaction). Most z/VSE users are familiar with LIBR and DITTO. VSAM users must be familiar with IDCAMS.

Considering the wide-ranging functions and abilities of z/VSE, there are only a small number of system-provided utilities. This has resulted in a large number of customer-written utility programs (although most users refrain from naming them utilities), and many of these are widely shared by the user community. Independent software vendors also provide many similar products (for a fee). Some of these can be categorized as utilities. Of these, some compete with IBM utilities, while many others provide functions not included with the IBM-provided utilities.

Basic utilities

A few utility programs (using the traditional terminology) are widely used in batch jobs. These are described in some detail here.

LIBR

The LIBRarian program can be considered as a utility program. With LIBR you can:

- Define libraries and sublibraries.
- Catalog members.
- Copy or move libraries, sublibraries, and members.
- Delete libraries, sublibraries, and members.
- List directory information and member contents.
- Punch member contents.
- Rename sublibraries and members.
- Backup and restore libraries, sublibraries, and members.
- Alter member contents.
- Control library, sublbrary, and member attributes and run integrity tests.

DITTO

The DITTO program provides many functions for working with tape devices, disk devices, VTOCs and catalogs, VSE/VSAM data, VSE library members, and card images.

IDCAMS

The IDCAMS program is not part of the basic set of z/VSE utilities documented in the z/VSE Utilities manual. The IDCAMS program is primarily used to create and manipulate VSAM data sets. It has other functions (such as catalog updates), but it is most closely associated with VSAM. It provides many complex functions, and whole manuals are needed to describe all of them.

A typical example of a simple use of IDCAMS is as follows:

```
* $$ JOB JNM=OGDEN12,CLASS=0,DISP=D
// JOB OGDEN12
// DLBL OGDEN,'OGDEN.DATA.VSAM',,VSAM,CAT=VSESPUC
// EXEC IDCAMS,SIZE=AUTO
  DELETE (OGDEN.DATA.VSAM) CLUSTER PURGE   -
     CATALOG(VSESP.USER.CATALOG)
/*
// EXEC IDCAMS,SIZE=AUTO
  DEFINE CLUSTER(NAME(OGDEN.DATA.VSAM)     -
     FILE(OGDEN)                       -
     RECORDS (2000 1000)                 -
     TO (99366)                          -
     INDEXED                             -
     KEYS(9 8)                         -
     RECORDSIZE(72 100)               -
     CONTROLINTERVALSIZE(4096)           -
     SPANNED                             -
     SHR(4)                              -
     VOLUMES (DOSRES,SYSWK1))            -
     DATA (NAME (OGDEN.DATA.VSAM.@D@))     -
     INDEX (NAME (OGDEN.DATA.VSAM.@I@))    -
     CATALOG(VSESP.USER.CATALOG)
/*
```

This example illustrates a number of points:

- There are two job steps. The first step deletes the data set that will be created by the second step. This is a clean-up function. The data set might not exist at this point, and the first step will have a completion code indicating that the action failed. This is ignored.
- The second step creates a VSAM data set (with the DEFINE CLUSTER command). The DEFINE CLUSTER command is continued over several records. The continuation indicator is the hyphen (-) character.
- The VSAM data set is on volumes DOSRES and SYSWK1. It uses 2000 records for primary space and 1000 records for secondary allocation. The average record size is 72 bytes and the maximum record size is 100 bytes. (VSAM data sets always use variable length records.) The primary key (for accessing records in the data set) is 8 bytes long and begins at an offset of 9 bytes into each record.

System-oriented utilities

The programs discussed in this section provide several basic utility functions for system administrators and are only briefly described.

VSE/FastCopy

VSE/FastCopy exists in two different versions:

- VSE/FastCopy online
- VSE/FastCopy stand-alone

Generally, both versions work the same way and have the same purpose. The main difference is that VSE/FastCopy stand-alone supports only a subset of the functions supported by VSE/FastCopy online.

You can perform the following tasks using VSE/FastCopy:

- Create a FastCopy backup from disk to tape using the FCOPY DUMP command.
- Restore a FastCopy backup from tape to disk using the FCOPY RESTORE command.
- Copy from disk to disk using the FCOPY COPY command.

With the DUMP command you can backup from disk to tape:

- A complete disk volume
- A partial disk volume
- A single file
- A multivolume file

With the RESTORE command you can restore the data from tape produced by a DUMP command.

With the COPY command you can copy data from a source disk to a target disk directly.

CLRDK

The Clear Disk utility CLRDK provides functions to:

- Clear one or more extents on a Count Key Data (CKD) or Extended Count Key Data (ECKD) disk, or create a file label in the VTOC.
- Preformat the tracks of the cleared extents.

INTTP

With the INTTP utility you can write volume labels on magnetic tape or data cartridge for standard label checking.

IESUPDCF

The batch utility program IESUPDCF allows the system administrator to maintain user profiles in the VSE Control File (IESCNTL) and in the VSE/ICCF DTSFILE. With this program, you can ADD, ALTer, and DELete user profiles. IESUPDCF helps you save time when configuring user profiles.

DTRSETP

You use DTRSETP when tailoring or creating CPUVARn procedures or user-written SETPARM procedures. The program must be run in a batch partition. z/VSE uses the SETPARM Procedure CPUVAR1, or the appropriate CPUVARn procedure, to save system variables for startup from one IPL to the next.

DTRIINIT

With the utility program DTRIINIT you can load jobs into the VSE/POWER reader queue.

Printing or displaying system information

A couple of programs exist that allow showing and managing system information.

LSERV

The LSERV program prints the contents of the label information area to SYSLST.

LVTOC

The LVTOC program is used to list the file labels in a Volume Table of Contents (VTOC). A VTOC is an index of all files, and the remaining space, on a disk volume.

EREP

With the Environmental Recording, Editing and Printing (EREP) program you can:

- Save the recorder file contents.
- Reinitialize the recorder file.
- Build and update recorder file history tapes.
- Produce machine and device-related error reports.

The system recorder file is used to save information about machine or device related hardware error or device related statistical information.

PRINTLOG

With PRINTLOG you can print the hardcopy file. Each line that appears on the screen of the operator console is written to the hardcopy file, which resides on SYSREC. It may be necessary to print the hardcopy file or parts of it before it becomes full.

LISTLOG

With LISTLOG you can gather information about how a particular job has run on the system. This will provide a listing on SYSLST of the following items:

- All job control statements that are written to the console
- All console messages for the job
- All operator responses for the job
- Any attention routine messages and commands issued while the job was running

C

EBCDIC - 7-bit ASCII table

Hx	Dec	E	A	Hx	Dec	E	A	Hx	Dec	E	A	Hz	Dec	E	A
00	00		NUL	20	32		SP	40	64	SP	@	60	96	-	'
01	01			21	33		!	41	65		A	61	97	/	a
02	02			22	34		“	42	66		B	62	98		b
03	03			23	35		#	43	67		C	63	99		c
04	04			24	36		$	44	68		D	64	100		d
05	05			25	37		%	45	69		E	65	101		e
06	06			26	38		&	46	70		F	66	102		f
07	07			27	39		'	47	71		G	67	103		g
08	08			28	40		(	48	72		H	68	104		h
09	09			29	41		)	49	73		I	69	105		i
0A	10			2A	42		*	4A	74	^	J	6A	106	|	j
0B	11			2B	43		+	4B	75	.	K	6B	107	,	k
0C	12			2C	44		,	4C	76	<	L	6C	108	%	l
0D	13			2D	45		-	4D	77	(	M	6D	109	_	m
0E	14			2E	46		.	4E	78	+	N	6E	110	>	n
0F	15			2F	47		/	4F	79	|	O	6F	111	?	o
10	16			30	48		0	50	80	&	P	70	112		p
11	17			31	49		1	51	81		Q	71	113		q
12	18			32	50		2	52	82		R	72	114		r
13	19			33	51		3	53	83		S	73	115		s
14	20			34	52		4	54	84		T	74	116		t
15	21			35	53		5	55	85		U	75	117		u
16	22			36	54		6	56	86		V	76	118		v

Hx	Dec	E	A	Hx	Dec	E	A	Hx	Dec	E	A	Hz	Dec	E	A
17	23			37	55		7	57	87		W	77	119		w
18	24			38	56		8	58	88		X	78	120		x
19	25			39	57		9	59	89		Y	79	121		y
1A	26			3A	58		:	5A	90	!	Z	7A	122	:	z
1B	27			3B	59		;	5B	91	$	[	7B	123	#	{
1C	28			3C	60		<	5C	92	*	\	7C	124	@	|
1D	29			3D	61		=	5D	93	)	]	7D	125	'	}
1E	30			3E	62		>	5E	94	;	^	7E	126	=	~
1F	31			3F	63		?	5F	95	not	_	7F	127	“	
80	128			A0	160			C0	192	{		E0	224	\	
81	129	a		A1	161			C1	193	A		E1	225		
82	130	b		A2	162	s		C2	194	B		E2	226	S	
83	131	c		A3	163	t		C3	195	C		E3	227	T	
84	132	d		A4	164	u		C4	196	D		E4	228	U	
85	133	e		A5	165	v		C5	197	E		E5	229	V	
86	134	f		A6	166	w		C6	198	F		E6	230	W	
87	135	g		A7	167	x		C7	199	G		E7	231	X	
88	136	h		A8	168	y		C8	200	H		E8	232	Y	
89	137	i		A9	169	z		C9	201	I		E9	233	Z	
8A	138			AA	170			CA	202			EA	234		
8B	139			AB	171			CB	203			EB	235		
8C	140			AC	172			CC	204			EC	236		
8D	141			AD	173	[		CD	205			ED	237		
8E	142			AE	174			CE	206			EE	238		
8F	143			AF	175			CF	207			EF	239		
90	144			B0	176			D0	208	}		F0	240	0	
91	145	j		B1	177			D1	209	J		F1	241	1	
92	146	k		B2	178			D2	210	K		F2	242	2	
93	147	l		B3	179			D3	211	L		F3	243	3	
94	148	m		B4	180			D4	212	M		F4	244	4	
95	149	n		B5	181			D5	213	N		F5	245	5	
96	150	o		B6	182			D6	214	O		F6	246	6	
97	151	p		B7	183			D7	215	P		F7	247	7	
98	152	q		B8	184			D8	216	Q		F8	248	8	
99	153	r		B9	185			D9	217	R		F9	249	9	
9A	154			BA	186			DA	218			FA	250		
9B	155			BB	187			DB	219			FB	251		
9C	156			BC	188			DC	220			FC	252		
9D	157			BD	189	]		DD	221			FD	253		
9E	158			BE	190			DE	222			FE	254		
9F	159			BF	191			DF	223			FF	255		

D

Programming experience

This chapter discusses programming options, samples, and techniques that suit to generate applications that run with *Language Environment for z/VSE*.

The following starting points for *hands-on* experience are covered in this context:

- Simple batch programs in the COBOL, PL/I, and C language
- How to enable programs for execution in a z/VSE batch environment
- Involving Assembler routines by using the LE/VSE-provided macro support
- LE/VSE Callable services that suit to solve common application programming problems
- Application programming under CICS
- Interlanguage communication (ILC) applications
- Programming with other products
- Fourth Generation Language Programming and VSE

Basic language samples in COBOL, PL/I, and C

Within this section we show basic programming samples in the z/VSE languages COBOL/VSE, PLI/VSE, and C/VSE.

COBOL

Example D-1 outlines the structure of a basic COBOL/VSE batch program that defines a table with eight slots.

COBOL/VSE sample

Example: D-1 Table definition in COBOL

```
       ID DIVISION.
       PROGRAM-ID. COBSAMP.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  J     PIC 9(4) USAGE COMP.
       01  TABLE-X.
        02  SLOT PIC 9(4) USAGE COMP OCCURS 8 TIMES.
       PROCEDURE DIVISION.
           MOVE 8 TO J.
           MOVE 1 TO SLOT (J).
           GOBACK.
```

Job control to generate COBOL/VSE executable

Example: D-2 Skeleton to generate COBOL/VSE program (batch environment)

```
* $$ JOB JNM=COBSAMP,CLASS=A,DISP=D
* ***************************************** *
*                                           *
* SKELETON: GENERATE COBOL/VSE BATCH PROGRAM *
*                                           *
* JOB CONTROL TYPE: COMPILE/LINK/RUN        *
*                                           *
* ***************************************** *
*
* ---------------------------------
* STEP 1: COMPILE PROGRAM: COBSAMP
* ---------------------------------
*
// JOB COBSAMP COMPILE PROGRAM COBSAMP
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.PROD)
// SETPARM CATALOG=1
// IF CATALOG = 1 THEN
// GOTO CAT1
// OPTION ERRS,SXREF,SYM,LIST,NODECK
// GOTO ENDCAT1
/. CAT1
// LIBDEF PHASE,CATALOG=USER.LIB
// OPTION ERRS,SXREF,SYM,NODECK,CATAL
   PHASE COBSAMP,*
/. ENDCAT1
// EXEC IGYCRCTL,SIZE=IGYCRCTL
 CBL LIB,APOST,NOADV,RENT,BUF(4096) DYNAM LIST
* SOURCE TO COMPILE COMES HERE !
/*
*
* ------------------------------------
* STEP 2: LINK PROGRAM PHASE: COBSAMP
* ------------------------------------
*
// IF CATALOG NE 1 OR $MRC GT 4 THEN
// GOTO NOLNK1
// EXEC LNKEDT
/*
*
* ------------------------------------
* STEP 3: RUN/INVOKE PROGRAM: COBSAMP
* ------------------------------------
*
// LIBDEF *,SEARCH=(USER.LIB,PRD2.SCEEBASE)
// EXEC COBSAMP,SIZE=COBSAMP
/*
/. NOLNK1
/&
* $$ EOJ
```

PLI/VSE

Example D-3 demonstrates the structure of a basic PLI/VSE program that calls a subroutine that does not exist. This error condition forces an LE/VSE dump at execution time.

PLI/VSE sample

Example: D-3   Calling a nonexistent subroutine from PL/I main routine

```
 EXAMPLE:  PROC  OPTIONS(MAIN);
  DCL Prog01      entry external;
  On error
    Begin;
      On error system;
      Call plidump('tbnfs','Plidump called from error ON-unit');
    End;
  Call Prog01;              /* Call external program PROG01   */
 End Example;
```

Job control to generate PLI/VSE executable

Example: D-4   Skeleton to generate PLI/VSE program (batch environment)

```
* $$ JOB JNM=PLISAMP,CLASS=A,DISP=D
* *************************************** *
*                                         *
* SKELETON: GENERATE PLI/VSE BATCH PROGRAM *
*                                         *
* JOB CONTROL TYPE: COMPILE/LINK/RUN      *
*                                         *
* *************************************** *
*
* -------------------------------
* STEP 1: COMPILE PROGRAM PLISAMP
* -------------------------------
*
// JOB PLISAMP COMPILE PROGRAM PLISAMP
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.PROD)
// SETPARM CATALOG=1
// IF CATALOG = 1 THEN
// GOTO CAT1
// OPTION NODECK
// GOTO ENDCAT1
/. CAT1
// LIBDEF PHASE,CATALOG=USER.LIB
// OPTION CATAL
   PHASE PLISAMP,*
/. ENDCAT1
// EXEC IEL1AA,SIZE=256K
*PROCESS INCLUDE,SYSTEM(VSE),FLAG(I),XREF(SHORT),MAP,LIST;
*PROCESS LINECOUNT(100);
* SOURCE TO COMPILE COMES HERE !
/*
*
* ------------------------------------
* STEP 2: LINK PROGRAM PHASE: PLISAMP
* ------------------------------------
*
// IF CATALOG NE 1 OR $MRC GT 4 THEN
// GOTO NOLNK1
// EXEC LNKEDT,SIZE=256K
/*
*
* ------------------------------------
* STEP 3: RUN/INVOKE PROGRAM: PLISAMP
* ------------------------------------
*
// LIBDEF *,SEARCH=(USER.LIB,PRD2.SCEEBASE)
// EXEC PLISAMP,SIZE=PLISAMP
/*
/. NOLNK1
/&
* $$ EOJ
```

C/VSE

Example D-5 shows the structure of a basic C/VSE program that takes user input from VSE console.

C/VSE sample

Example: D-5 Communicate with VSE console

```
/* Header file includes */
 #include <stdio.h>
 #include <stdlib.h>
 #include <__messag.h>
 int main()
 {
    /* ----------------------------------------------------------- */
    /* C/VSE application exploiting __console() function           */
    /*                                                             */
    /* Purpose: Show LE/VSE batch run-time options on VSE console  */
    /* ----------------------------------------------------------- */
    char xmsg[300];
    char rmsg[300];
    struct __cons_msg x_msg;
    sprintf(xmsg,"Please enter: <Hello World> here");
    x_msg.__format.__f1.__msg        = xmsg;
    x_msg.__format.__f1.__msg_length = strlen(xmsg);
    __console(&x_msg,rmsg);
    printf("\nUser reply from VSE Console : %s\n",rmsg);
    return 0;
 }
```

Job control to generate C/VSE executable

Example: D-6 Skeleton to generate C/VSE program (batch environment)

```
* $$ JOB JNM=CSAMP,CLASS=A,DISP=D
* ************************************** *
*                                        *
* SKELETON: GENERATE C/VSE BATCH PROGRAM *
*                                       *
* JOB CONTROL TYPE: COMPILE/LINK/RUN    *
*                                       *
* ************************************* *
*
* ------------------------------
* STEP 1: COMPILE PROGRAM: CSAMP
* ------------------------------
*
// JOB CSAMP COMPILE PROGRAM CSAMP
// DLBL SYSMSGS,'CVSE.COMP.MSGS',0,VSAM,RECSIZE=3000,RECORDS=35,       X
               CAT=VSESPUC
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.DBASE)
// SETPARM CATALOG=1
// IF CATALOG = 1 THEN
// GOTO CAT1
// OPTION ERRS,SXREF,SYM,LIST,NODECK
// GOTO ENDCAT1
/. CAT1
// LIBDEF PHASE,CATALOG=USER.LIB
// OPTION ERRS,SXREF,SYM,NODECK,CATAL
   PHASE CSAMP,*
/. ENDCAT1
// EXEC EDCCOMP,SIZE=EDCCOMP,PARM='NATLANG(ENU)/LONGNAME'
* SOURCE TO COMPILE COMES HERE !
/*
*
* -------------------------------------------
* STEP 2: PRELINK & LINK PROGRAM PHASE: CSAMP
* -------------------------------------------
*
// IF CATALOG NE 1 OR $MRC GT 4 THEN
// GOTO NOLNK1
// EXEC EDCPRLK,SIZE=EDCPRLK,PARM='NATLANG(ENU)/UPCASE'
/*
// EXEC LNKEDT,SIZE=256K
/*
*
* ---------------------------------
* STEP 3: RUN/INVOKE PROGRAM: CSAMP
* ---------------------------------
*
// LIBDEF *,SEARCH=(USER.LIB,PRD2.SCEEBASE)
// EXEC CSAMP,SIZE=CSAMP
/*
/. NOLNK1
/&
```

```
* $$ EOJ
```

LE/VSE callable services (all languages)

An overview of application programming services and examples that can be used cross-language is given in *LE/VSE Programming Reference*, SC33-6685.

Solution areas

The major solution areas that can be covered with coding LE/VSE callable services are:

- Condition handling
- Date and time
- Dynamic storage
- General
- Initialization and termination
- Locales
- Math
- Message handling
- National language support

Need a tool to play with it?

A sample tool exists that provides an integrated Java front end to download and try out, which is shipped with LE/VSE sample source programs, written in the COBOL, PL/I, and C programming language. The associated focus is with the above-listed solution areas, especially LE/VSE callable service groups.

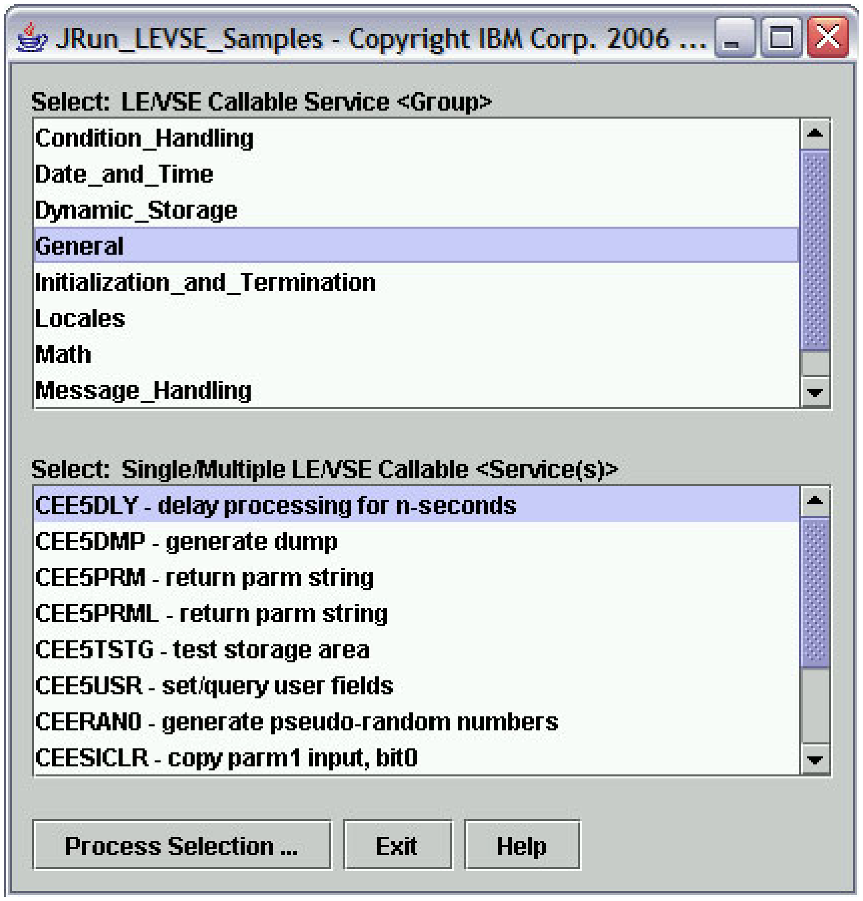

JRun_LEVSE_Samples - Copyright IBM Corp. 2006 ...
Select: LE/VSE Callable Service <Group>
Condition_Handling
Date_and_Time
Dynamic_Storage
General
Initialization_and_Termination
Locales
Math
Message_Handling
Select: Single/Multiple LE/VSE Callable <Service(s)>
CEE5DLY - delay processing for n-seconds
CEE5DMP - generate dump
CEE5PRM - return parm string
CEE5PRML - return parm string
CEE5TSTG - test storage area
CEE5USR - set/query user fields
CEERAN0 - generate pseudo-random numbers
CEESICLR - copy parm1 input, bit0
Process Selection ...
Exit
Help

Figure D-1 JRun_LEVSE_Samples (main panel view)

For tool download options please refer to *JRun_LEVSE_Samples* on z/VSE at:

http://www.ibm.com/servers/eserver/zseries/zvse/downloads/

Involving Assembler routines

You may find it useful to involve Assembler routines to call high-level language (HLL) routines or get called from HLL-routines such as COBOL/VSE, PLI/VSE, or C/VSE.

Understanding the key scenarios

To communicate with LE/VSE and other applications in a common run-time environment, Assembler applications must preserve the use of certain registers and storage areas. There are several LE/VSE-provided programming facilities that support the use of these conventions or help to code your programs.

- The CEEENTRY/CEETERM macro support provides assistance in entry and exit coding of Assembler routines, mapping of fields and communication areas.
- The pre-initialization service CEEPIPI suits for Assembler driver programs that do not have established a common run-time environment, however, intend to call high-level language routines.
- Assembler routines that are coded in between a calling and called C program can use the EDCPRLG macro to generate prolog code (indicating *Assembler start*) and the EDCEPIL macros to generate epilog code (signaling *Assembler end*).
- The Assembler macro CEEFETCH allows an LE/VSE-conforming Assembler routine to dynamically load an LE/VSE-conforming HLL subroutine. This loaded LE/VSE-conforming HLL subroutine can then be later released using CEERELES.

The above facilities, related details, and many samples are discussed in *LE/VSE Programming Guide*, SC33-6684, and *LE/VSE C Run-Time Programming Guide*, SC33-6688.

See also "Assembler - HLL communication via Language Environment" on page 245.

Samples on z/VSE home page

A detailed discussion and sample reference about the above scenarios is also available as a PDF document on the z/VSE Home Page:

http://www.ibm.com/servers/eserver/zseries/zvse/downloads/samples.html#le

Go to **LE/VSE Samples** → **Coding Techniques for Mixed Language Applications under LE/VSE (involving Assembler)**.

Developing applications under CICS

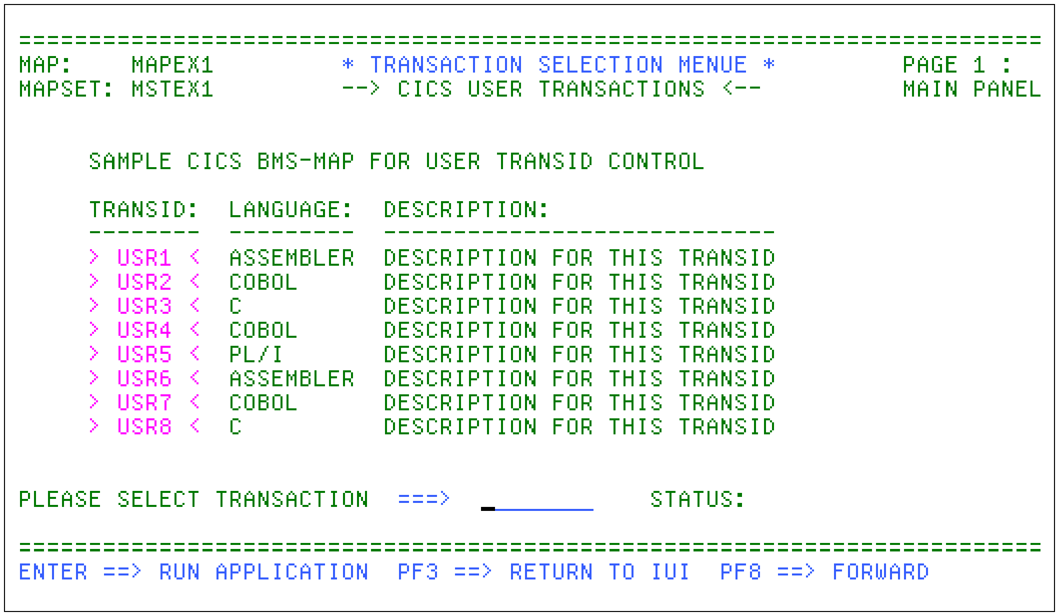
This section introduces a COBOL/CICS sample that exploits CICS Basic Mapping Support (BMS) to invoke a set of fictive user-written transactions (USR1-USR8).

In general, BMS maps serve as an application programming interface between CICS programs and terminal devices (3270 support). This form represents the *traditional* way to communicate with online applications.

For an introduction to 3270 family of terminals refer to *CICS Transaction Server for VSE/ESA Application Programming Guide*, SC33-1657.

Visible representation of BMS map MAPEX1

Figure D-2 shows our sample BMS map: MAPEX1 (in *traditional* 3270 view).



```
==========================================================================
MAP:     MAPEX1          * TRANSACTION SELECTION MENUE *         PAGE 1 :
MAPSET:  MSTEX1          --> CICS USER TRANSACTIONS <--          MAIN PANEL


     SAMPLE CICS BMS-MAP FOR USER TRANSID CONTROL

     TRANSID:  LANGUAGE:  DESCRIPTION:
     --------  ---------  ----------------------------
     > USR1 <  ASSEMBLER  DESCRIPTION FOR THIS TRANSID
     > USR2 <  COBOL      DESCRIPTION FOR THIS TRANSID
     > USR3 <  C          DESCRIPTION FOR THIS TRANSID
     > USR4 <  COBOL      DESCRIPTION FOR THIS TRANSID
     > USR5 <  PL/I       DESCRIPTION FOR THIS TRANSID
     > USR6 <  ASSEMBLER  DESCRIPTION FOR THIS TRANSID
     > USR7 <  COBOL      DESCRIPTION FOR THIS TRANSID
     > USR8 <  C          DESCRIPTION FOR THIS TRANSID


PLEASE SELECT TRANSACTION  ===>  ________    STATUS:

==========================================================================
ENTER ==> RUN APPLICATION  PF3 ==> RETURN TO IUI  PF8 ==> FORWARD
```

Figure D-2 Sample BMS map MAPEX1

Source code for BMS map MAPEX1

The CICS macro definitions representing BMS map MAPEX1 (map set MSTEX1) are shown in Example D-7.

Example: D-7 Map definition MAPEX1

```
MACRO DEFINITIONS FOR MAPEX1                *
******************************************************
MSTEX1  DFHMSD TYPE=MAP,MODE=INOUT,STORAGE=AUTO,LANG=COBOL,TIOAPFX=YES,*
               EXTATT=YES
*              MAPATTS=(COLOR,HILIGHT,PS,VALIDN)
*              DSATTS=(COLOR,HILIGHT,PS,VALIDN)
```

```
MAPEX1  DFHMDI SIZE=(24,80),COLUMN=1,LINE=1,CTRL=FREEKB
* -------------------- *
* HEADER DEFINITIONS   *
* -------------------- *
        DFHMDF POS=(2,1),LENGTH=74,ATTRB=(PROT,NORM),                 *
               COLOR=GREEN,                                           *
               INITIAL='==============================================*
               ============================'
        DFHMDF POS=(3,1),LENGTH=14,ATTRB=(PROT,NORM),                 *
               COLOR=GREEN,                                           *
               INITIAL='MAP:    MAPEX1'
        DFHMDF POS=(3,24),LENGTH=31,ATTRB=(PROT,NORM),                *
               COLOR=BLUE,                                            *
               INITIAL='* TRANSACTION SELECTION MENUE *'
        DFHMDF POS=(3,64),LENGTH=8,ATTRB=(PROT,NORM),                 *
               COLOR=GREEN,                                           *
               INITIAL='PAGE 1 :'
        DFHMDF POS=(4,1),LENGTH=14,ATTRB=(PROT,NORM),                 *
               COLOR=GREEN,                                           *
               INITIAL='MAPSET: MSTEX1'
        DFHMDF POS=(4,24),LENGTH=32,ATTRB=(PROT,NORM),                *
               COLOR=GREEN,                                           *
               INITIAL='--> CICS USER TRANSACTIONS <--'
        DFHMDF POS=(4,64),LENGTH=10,ATTRB=(PROT,NORM),                *
               COLOR=GREEN,                                           *
               INITIAL='MAIN PANEL'
* ---------------------------------------- *
*    DATA-FIELDS AND DESCRIBTION           *
* ---------------------------------------- *
        DFHMDF POS=(7,6),LENGTH=44,ATTRB=(PROT,NORM),                 *
               COLOR=GREEN,                                           *
               INITIAL='SAMPLE CICS BMS-MAP FOR USER TRANSID CONTROL'
        DFHMDF POS=(9,6),LENGTH=33,ATTRB=(PROT,NORM),                 *
               COLOR=GREEN,                                           *
               INITIAL='TRANSID:  LANGUAGE:  DESCRIPTION:'
        DFHMDF POS=(10,6),LENGTH=49,ATTRB=(PROT,NORM),                *
               COLOR=GREEN,                                           *
               INITIAL='--------  ---------  --------------------------*
               ---'
ID01    DFHMDF POS=(11,6),LENGTH=8,ATTRB=(PROT,NORM),                 *
               COLOR=PINK,INITIAL='> USR1 <'
        DFHMDF POS=(11,16),LENGTH=9,ATTRB=(PROT,NORM),                *
               COLOR=GREEN,INITIAL='ASSEMBLER'
        DFHMDF POS=(11,27),LENGTH=30,ATTRB=(PROT,NORM),               *
               COLOR=GREEN,INITIAL='DESCRIPTION FOR THIS TRANSID  '
ID02    DFHMDF POS=(12,6),LENGTH=8,ATTRB=(PROT,NORM),                 *
               COLOR=PINK,INITIAL='> USR2 <'
        DFHMDF POS=(12,16),LENGTH=9,ATTRB=(PROT,NORM),                *
               COLOR=GREEN,INITIAL='COBOL    '
```

```
         DFHMDF POS=(12,27),LENGTH=30,ATTRB=(PROT,NORM),               *
               COLOR=GREEN,INITIAL='DESCRIPTION FOR THIS TRANSID  '
ID03     DFHMDF POS=(13,6),LENGTH=8,ATTRB=(PROT,NORM),                 *
               COLOR=PINK,INITIAL='> USR3 <'
         DFHMDF POS=(13,16),LENGTH=9,ATTRB=(PROT,NORM),                *
               COLOR=GREEN,INITIAL='C        '
         DFHMDF POS=(13,27),LENGTH=30,ATTRB=(PROT,NORM),               *
               COLOR=GREEN,INITIAL='DESCRIPTION FOR THIS TRANSID  '
ID04     DFHMDF POS=(14,6),LENGTH=8,ATTRB=(PROT,NORM),                 *
               COLOR=PINK,INITIAL='> USR4 <'
         DFHMDF POS=(14,16),LENGTH=9,ATTRB=(PROT,NORM),                *
               COLOR=GREEN,INITIAL='COBOL    '
         DFHMDF POS=(14,27),LENGTH=30,ATTRB=(PROT,NORM),               *
               COLOR=GREEN,INITIAL='DESCRIPTION FOR THIS TRANSID  '
ID05     DFHMDF POS=(15,6),LENGTH=8,ATTRB=(PROT,NORM),                 *
               COLOR=PINK,INITIAL='> USR5 <'
         DFHMDF POS=(15,16),LENGTH=9,ATTRB=(PROT,NORM),                *
               COLOR=GREEN,INITIAL='PL/I     '
         DFHMDF POS=(15,27),LENGTH=30,ATTRB=(PROT,NORM),               *
               COLOR=GREEN,INITIAL='DESCRIPTION FOR THIS TRANSID  '
ID06     DFHMDF POS=(16,6),LENGTH=8,ATTRB=(PROT,NORM),                 *
               COLOR=PINK,INITIAL='> USR6 <'
         DFHMDF POS=(16,16),LENGTH=9,ATTRB=(PROT,NORM),                *
               COLOR=GREEN,INITIAL='ASSEMBLER'
         DFHMDF POS=(16,27),LENGTH=30,ATTRB=(PROT,NORM),               *
               COLOR=GREEN,INITIAL='DESCRIPTION FOR THIS TRANSID  '
ID07     DFHMDF POS=(17,6),LENGTH=8,ATTRB=(PROT,NORM),                 *
               COLOR=PINK,INITIAL='> USR7 <'
         DFHMDF POS=(17,16),LENGTH=9,ATTRB=(PROT,NORM),                *
               COLOR=GREEN,INITIAL='COBOL    '
         DFHMDF POS=(17,27),LENGTH=30,ATTRB=(PROT,NORM),               *
               COLOR=GREEN,INITIAL='DESCRIPTION FOR THIS TRANSID  '
ID08     DFHMDF POS=(18,6),LENGTH=8,ATTRB=(PROT,NORM),                 *
               COLOR=PINK,INITIAL='> USR8 <'
         DFHMDF POS=(18,16),LENGTH=9,ATTRB=(PROT,NORM),                *
               COLOR=GREEN,INITIAL='C        '
         DFHMDF POS=(18,27),LENGTH=30,ATTRB=(PROT,NORM),               *
               COLOR=GREEN,INITIAL='DESCRIPTION FOR THIS TRANSID  '
SEL1     DFHMDF POS=(21,1),LENGTH=26,ATTRB=(PROT,NORM),                *
               COLOR=GREEN,                                            *
               INITIAL='PLEASE SELECT TRANSACTION '
         DFHMDF POS=(21,28),LENGTH=5,ATTRB=(PROT,NORM),                *
               COLOR=BLUE,INITIAL='===> '
TRAN1    DFHMDF POS=(21,34),LENGTH=8,ATTRB=(UNPROT,NORM,IC),           *
               COLOR=BLUE,HILIGHT=UNDERLINE
         DFHMDF POS=(21,43),LENGTH=1,ATTRB=(ASKIP,NORM)
STAT1    DFHMDF POS=(21,46),LENGTH=8,ATTRB=(PROT,NORM),                *
               COLOR=GREEN,INITIAL='STATUS: '
CONT1    DFHMDF POS=(21,56),LENGTH=9,ATTRB=(PROT,BRT),                 *
```

```
              COLOR=BLUE
* ---------------------------------------------- *
*    TRAILOR/FAILURE MESSAGES + INFORMATION...     *
* ---------------------------------------------- *
        DFHMDF POS=(23,1),LENGTH=74,ATTRB=(PROT,NORM),              *
              COLOR=GREEN,                                          *
              INITIAL='=============================================*
              ========================='
        DFHMDF POS=(24,1),LENGTH=65,ATTRB=(PROT,NORM),              *
              COLOR=BLUE,                                           *
              INITIAL='ENTER ==> RUN APPLICATION  PF3 ==> RETURN TO IU*
              I  PF8 ==> FORWARD'
        DFHMSD TYPE=FINAL
        END
```

For an introduction to CICS Basic Mapping Support and map definitions via the DFHMSD, DFHMDI, and DFHMDF macros refer to *CICS Transaction Server for VSE/ESA Application Programming Guide*, SC33-1657.

JCL to generate BMS map load module and data structure

Example D-8 shows the transformation of the BMS source to a load module (called a *physical* map set) and data structure (called a *symbolic* map set). In essence this is accomplished by an option "SYSPARM=MAP" and "SYSPARM=DSECT" assembly.

Example: D-8 JCL to generate BMS map

```
* $$ JOB JNM=MAPPREP,DISP=D,CLASS=4,NTFY=YES
// JOB MAPPREP  GENERATE CICS BMS MAP
* --------------------------------------
* STEP 1:  CATALOG MAP SOURCE STATEMENTS
* --------------------------------------
// EXEC LIBR
    ACCESS SUBL=PRD2.CONFIG
    CATALOG MAPEX1.A       REPLACE=YES
* $$ SLI ICCF=(MAPEX1),LIB=(13)
/+
/*
* ----------------------------------------------------------------
* STEP 2:  ASSEMBLE AND LINKEDT A LOAD MODULE (CALLED: PHYSICAL MAP)
* ----------------------------------------------------------------
// OPTION CATAL,NODECK,SYSPARM='MAP',ALIGN
   PHASE MSTEX1
// LIBDEF *,SEARCH=(PRD2.CONFIG,PRD1.BASE)
// LIBDEF PHASE,CATALOG=PRD2.CONFIG
// EXEC ASSEMBLY,SIZE=450K
         COPY MAPEX1
         END
/*
// EXEC LNKEDT
*
* --------------------------------------------------------------------
* STEP 3:  ASSEMBLE AND CATALOG DATA STRUCTURE (CALLED: SYMBOLIC MAP)
* --------------------------------------------------------------------
// OPTION DECK,SYSPARM='DSECT'
* $$ SLI ICCF=(CKDPCH),LIB=(13)
// EXEC ASSEMBLY,SIZE=450K
         PUNCH ' ACCESS SUBLIB=PRD2.CONFIG'
         PUNCH ' CATALOG MSTEX1.C REPLACE=YES '
         COPY MAPEX1
         END
/*
CLOSE SYSPCH,UA
* $$ SLI ICCF=(CKDIPT),LIB=(13)
// EXEC LIBR
CLOSE SYSIPT,UA
/*
/&
// JOB RESET
ASSGN SYSIPT,FEC        IF 1A93D, CLOSE SYSIPT,SYSRDR
ASSGN SYSPCH,FED        IF 1A93D, CLOSE SYSPCH,FED
/&
* $$ EOJ
```

Note: The above job control presumes a disk extent to be available for SYSIPT and SYSPCH processing of generated parts. This area is referred to via * $$ SLI statements. The members are stored in ICCF library 13 (the term *SLI* stands for source library inclusion).

For details on physical and symbolic maps sets also refer to *CICS Transaction Server for VSE/ESA Application Programming Guide*, SC33-1657.

Content of member CKDPCH referring a disk punch area

```
* $$ SLI ICCF=(CKDPCH),LIB=(13)
```

Example: D-9   SYSPCH disk reference

```
* CHANGE THE EXTENT TO VALUES SUITABLE FOR YOUR INSTALLATION
// DLBL IJSYSPH,'ASM.WORKFILE',0
// EXTENT SYSPCH,,1,0,13260,300
ASSGN SYSPCH,DISK,VOL=SYSWK1,SHR
```

Member CKDIPT referring the same disk extent for input

```
* $$ SLI ICCF=(CKDIPT),LIB=(13)
```

Example: D-10   SYSIPT disk reference

```
* CHANGE THE EXTENT TO VALUES SUITABLE FOR YOUR INSTALLATION
// DLBL IJSYSIN,'ASM.WORKFILE',0
// EXTENT SYSIPT,,1,0,13260,300
ASSGN SYSIPT,DISK,VOL=SYSWK1,SHR
```

CICS COBOL source to send map SMAPEX1

We are now at the point where the BMS map can be bound to CICS applications by copying the data structure and using EXEC CICS commands to send the map to a terminal. A related CICS COBOL program is shown in Example D-11.

Example: D-11   CICS COBOL program to present BMS map

```
ID DIVISION.
      *----------------------------------------
         SHOW CICS BMS-MAP: MAPEX1 ON TERMINAL
      *----------------------------------------
       PROGRAM-ID. 'SMAPEX1'
       ENVIRONMENT DIVISION.
       DATA DIVISION.
      *----------------------------------------
       WORKING-STORAGE SECTION.
        COPY MSTEX1.
      *----------------------------------------
       PROCEDURE DIVISION.
      *    CLEAR OUTPUT-FIELDS FOR MAPEX1
           MOVE LOW-VALUE TO MAPEX1O.
       SENDMAP.
           EXEC CICS SEND MAP('MAPEX1') MAPSET('MSTEX1')
                ERASE FREEKB END-EXEC.
           EXEC CICS RETURN TRANSID ('EX1P') END-EXEC.
           STOP RUN.
```

Note: The above program clears output fields, sends the map, and returns control to a further program (transaction *EX1P*) that processes user input.

The generated map can even be tested without availability of a program similar to the one above. A related command-level interpreter is supplied via transaction *CECI*.

For details on system-provided transactions refer to *CICS Transaction Server for VSE/ESA(TM) CICS-Supplied Transactions*, SC33-1655.

Display BMS map via CICS command-level interpreter

The following steps can be used to display a map with CICS command-level interpreter. By use of the VSE Interactive Interface, main panel, PF6 "ESCAPE TO CICS" function the following enabling commands can be entered:

Step 1 CEMT SET PROGRAM(MSTEX1) NEWCOPY

This tells CICS to use the latest version of our map load module.

Step 2 CECI SEND MAP(MAPEX1) MAPSET(MSTEX1)

This instructs CICS to test and display the BMS map built.

JCL to compile CICS COBOL program SMAPEX1

The JCL generation process to build an executable for program SMAPEX1 is the next focus point. It allows multiple CICS users to invoke the program once a related transaction has been defined and security enabled. It is assumed that transaction EX1S is associated with program SMAPEX1 (for example, by use of CICS-supplied CEDA transaction).

Example D-12 shows the job control to compile the CICS COBOL program SMAPEX1.

Example: D-12 JCL to compile CICS COBOL program SMAPEX1

```
* $$ JOB JNM=COMSMAP1,DISP=D,CLASS=A,NTFY=YES
* $$ LST DISP=D,CLASS=Q,PRI=3
* $$ PUN DISP=I,DEST=*,PRI=9,CLASS=A
// JOB COMSMAP1 TRANSLATE PROGRAM SMAPEX1
// ASSGN SYSIPT,SYSRDR
// EXEC IESINSRT
$ $$ LST DISP=D,CLASS=Q,PRI=3
// JOB COMSMAP1 COMPILE PROGRAM SMAPEX1
// LIBDEF *,SEARCH=(PRD2.SCEEBASE)
// SETPARM CATALOG=1
// IF CATALOG = 1 THEN
// GOTO CAT
// OPTION ERRS,SXREF,SYM,LIST,NODECK
// GOTO ENDCAT
/. CAT
// LIBDEF PHASE,CATALOG=PRD2.CONFIG
// OPTION ERRS,SXREF,SYM,CATAL,NODECK
   PHASE SMAPEX1,*
   INCLUDE DFHELII
/. ENDCAT
// EXEC IGYCRCTL,SIZE=IGYCRCTL
 CBL LIB,APOST,NOADV,NODYNAM,RENT,BUF(4096)
* $$ END
```

```
// ON $CANCEL OR $ABEND GOTO ENDJ2
// OPTION NOLIST,NODUMP,DECK
// EXEC DFHECP1$,SIZE=512K
 CBL XOPTS(COBOL3 CICS DEBUG)
* $$ SLI ICCF=(SMAPEX1),LIB=(0013)
/*
/. ENDJ2
// EXEC IESINSRT
/*
// IF CATALOG NE 1 OR $MRC GT 4 THEN
// GOTO NOLNK
// EXEC LNKEDT,SIZE=256K
/. NOLNK
#&
$ $$ EOJ
* $$ END
/&
* $$ EOJ
```

For information about how to define resources in CICS (for example, programs, map sets, transactions, and so on) refer to *CICS Transaction Server for VSE/ESA, Resource Definition Guide*, SC33-1653.

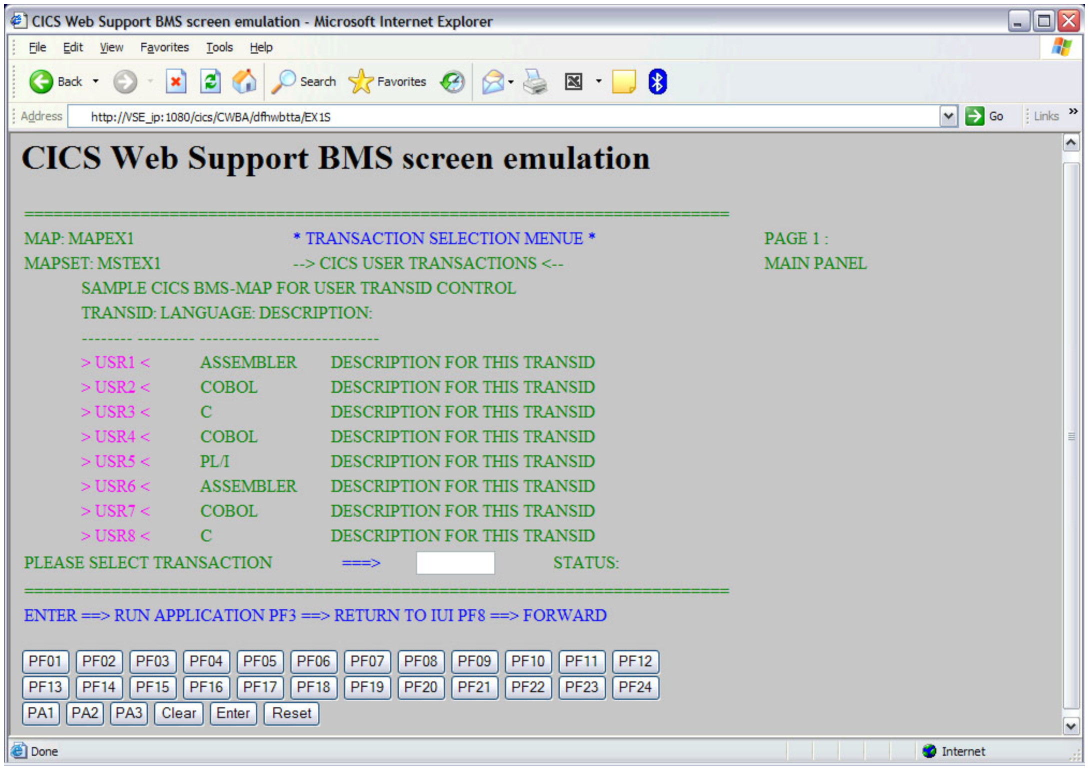
For information about how to security-enable user-defined transactions refer to *z/VSE Administration*.

Enabling map MAPEX1 for CICS Web Support

In this section we show a HTML view of the introduced BMS map. Its visual representation is a result of a more comprehensive alternative to generate BMS maps adding // OPTION SYSPARM='TEMPLATE'.

HTML correspondent of BMS map MAPEX1

The browser view shown in Figure D-3 is based on file MSTEX1A.HTML and invocation of user transaction "EX1S" (via CICS Web Support and associated transactions for HTTP requests).



```
CICS Web Support BMS screen emulation

==========================================================================
MAP: MAPEX1                  * TRANSACTION SELECTION MENUE *          PAGE 1 :
MAPSET: MSTEX1               --> CICS USER TRANSACTIONS <--           MAIN PANEL
      SAMPLE CICS BMS-MAP FOR USER TRANSID CONTROL
      TRANSID: LANGUAGE: DESCRIPTION:
      -------- --------- --------------------------
      > USR1 <      ASSEMBLER     DESCRIPTION FOR THIS TRANSID
      > USR2 <      COBOL         DESCRIPTION FOR THIS TRANSID
      > USR3 <      C             DESCRIPTION FOR THIS TRANSID
      > USR4 <      COBOL         DESCRIPTION FOR THIS TRANSID
      > USR5 <      PL/I          DESCRIPTION FOR THIS TRANSID
      > USR6 <      ASSEMBLER     DESCRIPTION FOR THIS TRANSID
      > USR7 <      COBOL         DESCRIPTION FOR THIS TRANSID
      > USR8 <      C             DESCRIPTION FOR THIS TRANSID
PLEASE SELECT TRANSACTION      ===>  [        ]     STATUS:
==========================================================================
ENTER ==> RUN APPLICATION PF3 ==> RETURN TO IUI PF8 ==> FORWARD
```

CICS Web Support BMS screen emulation - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address http://VSE_ip:1080/cics/CWBA/dfhwbtta/EX1S
Go
Links
PF01 PF02 PF03 PF04 PF05 PF06 PF07 PF08 PF09 PF10 PF11 PF12
PF13 PF14 PF15 PF16 PF17 PF18 PF19 PF20 PF21 PF22 PF23 PF24
PA1 PA2 PA3 Clear Enter Reset
Done
Internet

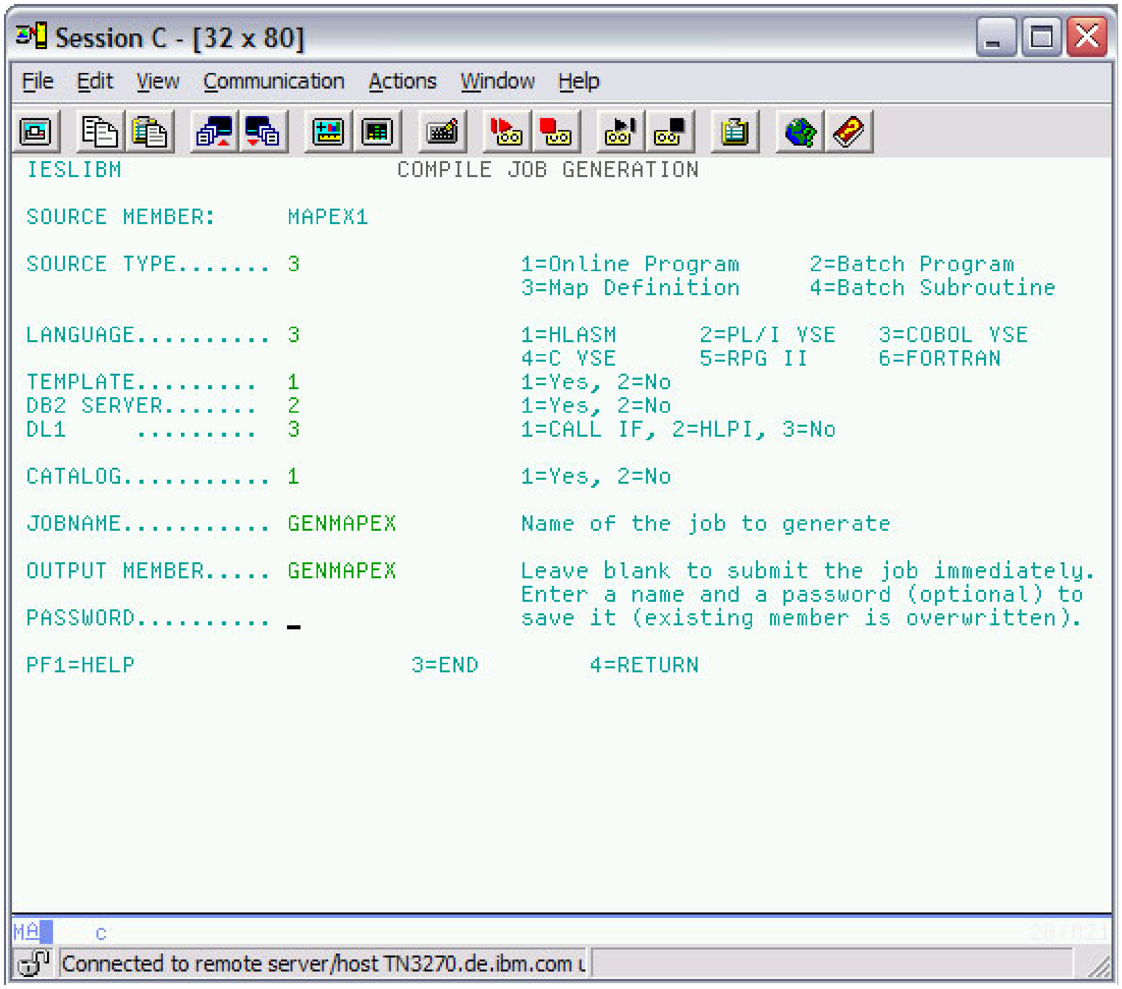
Figure D-3 HTML correspondent of BMS map MAPEX1

Note: For information about setup and configuration of CICS Web Support refer to *CICS Transaction Server for VSE/ESA Enhancements Guide,* GC34-5763-06, and *z/VSE V3R1.0 e-business Connectors, User's Guide,* SC33-8231.

Note that the map generation job shown in “JCL to generate BMS map load module and data structure” on page 449 represents basic JCL for 3270 terminals. It does not cover the SYSPARM=’TEMPLATE’ step, which is a prerequisite for the above HTML view.

To create VSE job control that includes a SYSPARM='TEMPLATE' map generation please use the Interactive Interface (Primary ICCF library, OPTION 8=COMPILE), as described in the next section

Full map capabilities - including SYSPARM='TEMPLATE'



```
IESLIBM                    COMPILE JOB GENERATION

SOURCE MEMBER:      MAPEX1

SOURCE TYPE....... 3                1=Online Program     2=Batch Program
                                    3=Map Definition     4=Batch Subroutine

LANGUAGE.......... 3                1=HLASM      2=PL/I VSE    3=COBOL VSE
                                    4=C VSE      5=RPG II      6=FORTRAN
TEMPLATE.........  1                1=Yes, 2=No
DB2 SERVER.......  2                1=Yes, 2=No
DL1     .........  3                1=CALL IF, 2=HLPI, 3=No

CATALOG........... 1                1=Yes, 2=No

JOBNAME........... GENMAPEX         Name of the job to generate

OUTPUT MEMBER..... GENMAPEX         Leave blank to submit the job immediately.
                                    Enter a name and a password (optional) to
PASSWORD.......... _                save it (existing member is overwritten).

PF1=HELP                   3=END        4=RETURN
```

Connected to remote server/host TN3270.de.ibm.com u

Figure D-4 Generate compile job to build BMS map with HTML correspondent

Reworking the generated HTML version of map MAPEX1

The generated MSTEX1A.HTML file might not be the final visual representation you would like to put in place.

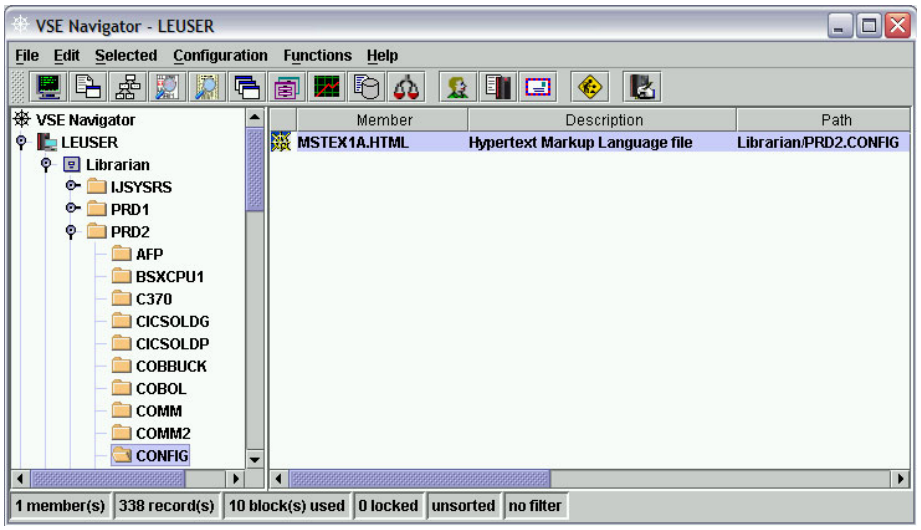
Attached is a sample procedure making use of VSE Navigator to directly access and modify the MSTEX1A.HTML file (located on the VSE host).

For more information and a download of VSE Navigator refer to z/VSE Home page and click **Downloads**:

`http://www.ibm.com/servers/eserver/zseries/zvse/downloads/`

VSE Navigator (librarian member view)

This panel shows the VSE librarian (library/member structure) with our HTML file subject to refinement.



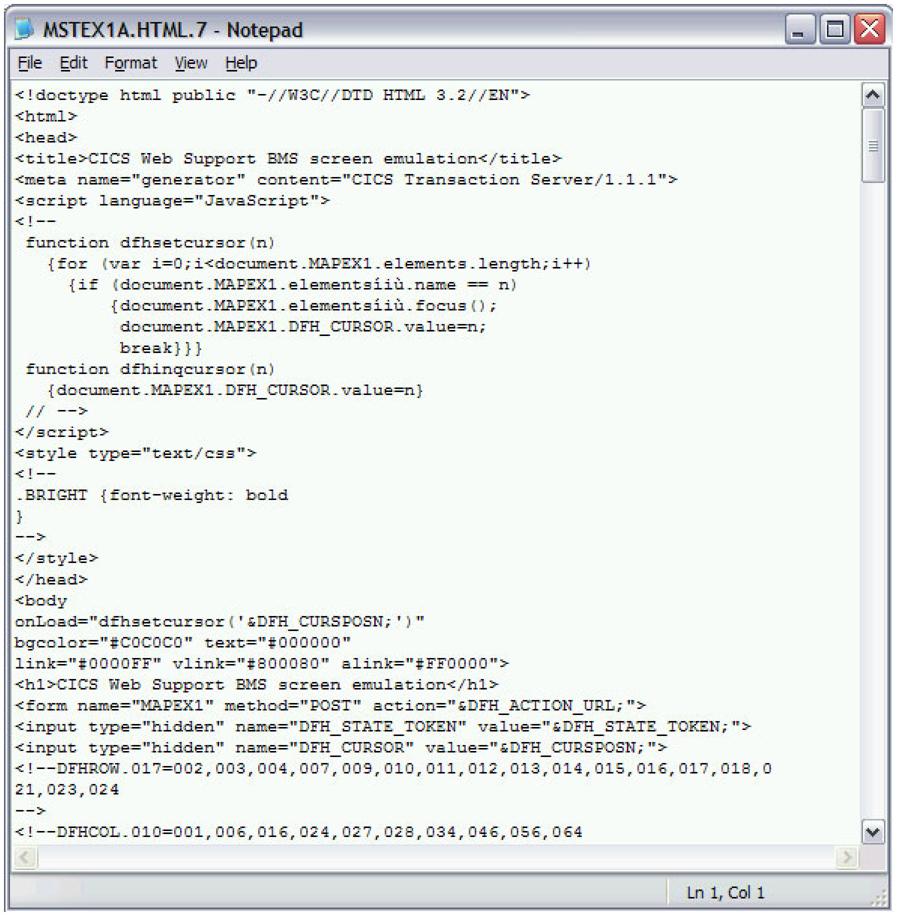
Member	Description	Path
MSTEX1A.HTML	Hypertext Markup Language file	Librarian/PRD2.CONFIG

1 member(s) | 338 record(s) | 10 block(s) used | 0 locked | unsorted | no filter

Figure D-5 VSE Navigator view of librarian resources

Editor view of subject HTML-file

Figure D-6 shows an extract of the MSTEX1A.HTML source file generated with the SYSPARM='TEMPLATE' option.

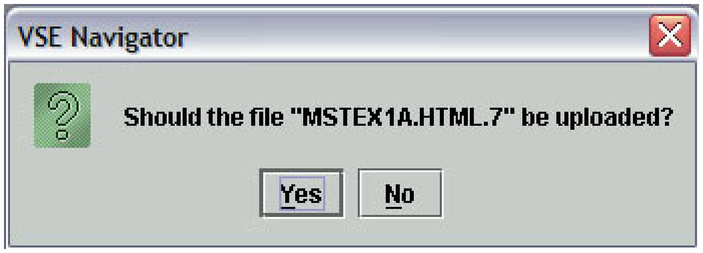


```
<!doctype html public "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
<title>CICS Web Support BMS screen emulation</title>
<meta name="generator" content="CICS Transaction Server/1.1.1">
<script language="JavaScript">
<!--
 function dfhsetcursor(n)
   {for (var i=0;i<document.MAPEX1.elements.length;i++)
     {if (document.MAPEX1.elementsíiù.name == n)
         {document.MAPEX1.elementsíiù.focus();
          document.MAPEX1.DFH_CURSOR.value=n;
          break}}}
 function dfhinqcursor(n)
   {document.MAPEX1.DFH_CURSOR.value=n}
 // -->
</script>
<style type="text/css">
<!--
.BRIGHT {font-weight: bold
}
-->
</style>
</head>
<body
onLoad="dfhsetcursor('&DFH_CURSPOSN;')"
bgcolor="#C0C0C0" text="#000000"
link="#0000FF" vlink="#800080" alink="#FF0000">
<h1>CICS Web Support BMS screen emulation</h1>
<form name="MAPEX1" method="POST" action="&DFH_ACTION_URL;">
<input type="hidden" name="DFH_STATE_TOKEN" value="&DFH_STATE_TOKEN;">
<input type="hidden" name="DFH_CURSOR" value="&DFH_CURSPOSN;">
<!--DFHROW.017=002,003,004,007,009,010,011,012,013,014,015,016,017,018,0
21,023,024
-->
<!--DFHCOL.010=001,006,016,024,027,028,034,046,056,064
```

Figure D-6 HTML source for BMS map generated with SYSPARM=TEMPLATE

Upload confirmation view

Having completed MSTEX1A.HTML editing, VSE Navigator will ask to confirm the upload to VSE host.

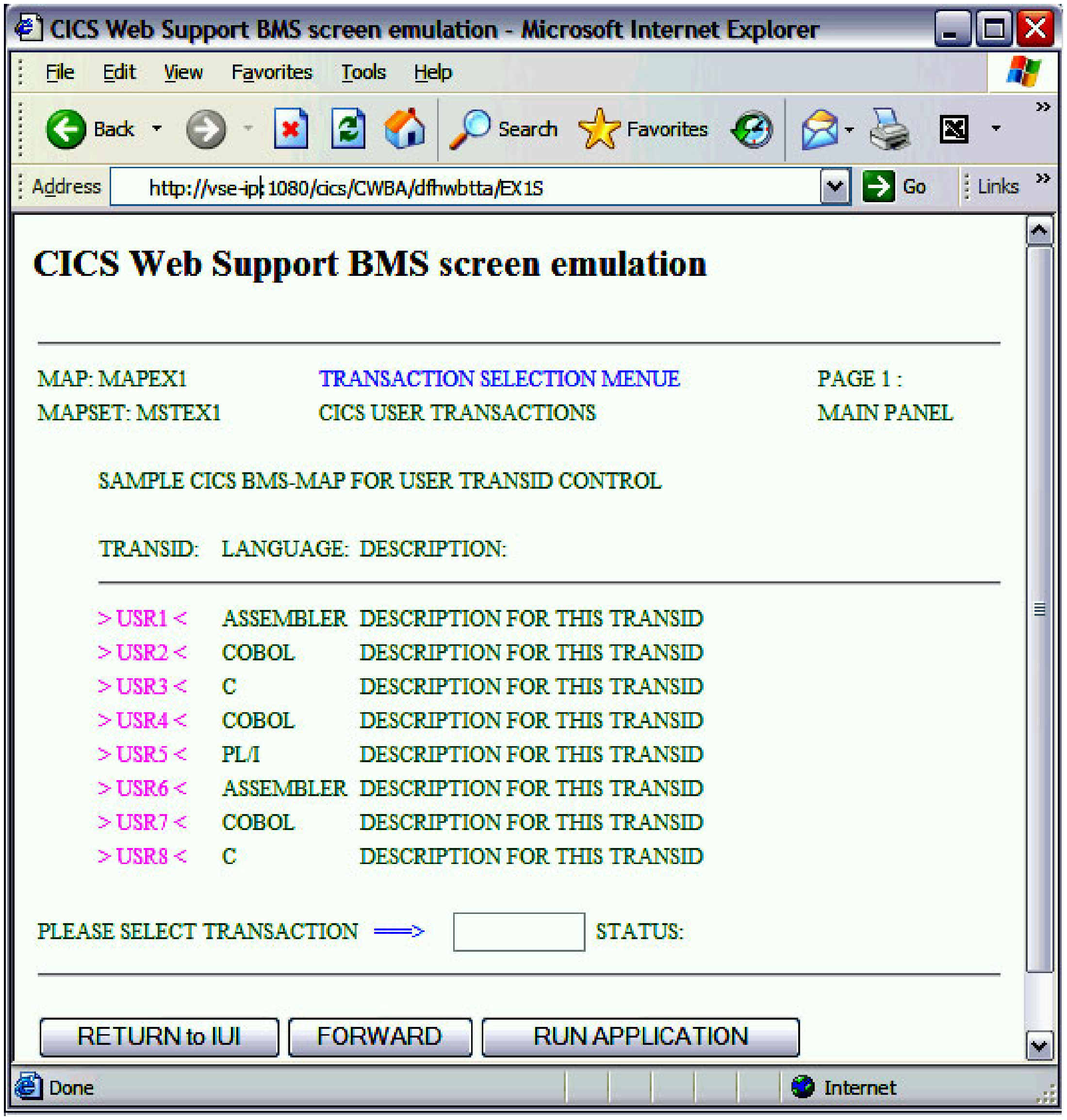


Should the file "MSTEX1A.HTML.7" be uploaded?

Figure D-7 Confirmation of VSE Navigator library member upload

Refreshed browser view

Figure D-8 shows how a refined MSTEX1A.HTML version may look. Remember the initial 3270 map on Figure D-2 on page 446—now you have the same presentation on a HTML browser.


CICS Web Support BMS screen emulation - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
Address
http://vse-ip:1080/cics/CWBA/dfhwbtta/EX1S
Go
Links
Done
Internet

CICS Web Support BMS screen emulation

```
MAP: MAPEX1            TRANSACTION SELECTION MENUE        PAGE 1 :
MAPSET: MSTEX1         CICS USER TRANSACTIONS             MAIN PANEL

     SAMPLE CICS BMS-MAP FOR USER TRANSID CONTROL

     TRANSID:   LANGUAGE:  DESCRIPTION:

     > USR1 <   ASSEMBLER  DESCRIPTION FOR THIS TRANSID
     > USR2 <   COBOL      DESCRIPTION FOR THIS TRANSID
     > USR3 <   C          DESCRIPTION FOR THIS TRANSID
     > USR4 <   COBOL      DESCRIPTION FOR THIS TRANSID
     > USR5 <   PL/I       DESCRIPTION FOR THIS TRANSID
     > USR6 <   ASSEMBLER  DESCRIPTION FOR THIS TRANSID
     > USR7 <   COBOL      DESCRIPTION FOR THIS TRANSID
     > USR8 <   C          DESCRIPTION FOR THIS TRANSID

PLEASE SELECT TRANSACTION ===>  [          ] STATUS:
```

RETURN to IUI | FORWARD | RUN APPLICATION

Figure D-8 Representation of user-refined BMS map (HTML version)

CICS COBOL program to process map MAPEX1

So far we did not consider that user BMS map input has to be accepted and processed. This is what is covered next.

The following CICS COBOL source denotes the major coding aspects to process the BMS map "MAPEX1".

- Copybook required for map data structure (MSTEX1.C).
- Copybooks required for PF-KEY checking and symbolic attributes (DFHAID, DFHBMSCA).
- BMS map receive processing.
- User input field checking to determine which transaction has to be processed. (TRAN1 is a input field of data structure MSTEX1.C.)
- The mechanism to transfer control to the user program associated with a transaction (USR1 → USR1PGM). The EXEC CICS LINK command ensures returning to the user map when having completed transaction processing.
- PF_KEY action processing, for example, program calls to display further map or exit to Interactive Interface (IUI).

For more information about EXECUTE CICS commands refer to *CICS Transaction Server for VSE/ESA Application Programming Reference,* SC33-1658.

Coding extract to process map MAPEX1

Attached are the major COBOL/CICS source code lines to take care of BMS user input and control the process flow. Program SMAPEX2 is supposed to send a further BMS map (not shown here) if the user presses the PF8 key.

Example: D-13 COBOL/CICS code extract to process BMS map

```
PROGRAM-ID. PMAPEX1.
       ENVIRONMENT DIVISION.
      *
       DATA DIVISION.
       WORKING-STORAGE SECTION.
...
      *     COPYBOOK CONTAINING MSTEX1.C DATA STRUCTURE
       COPY MSTEX1.
...
      *     COPYBOOK TO ALLOW PF-KEY PROCESSING/SYMBOLIC ATTRIBUTES
       COPY DFHAID.
       COPY DFHBMSCA.
...
       PROCEDURE DIVISION.
```

```
...
       GETMAP.
               EXEC CICS RECEIVE MAP('MAPEX1')
                         MAPSET('MSTEX1')
                         END-EXEC.
...
      *  RETURNING TO IUI PANEL HIERARCHY
               IF EIBAID = DFHPF3 THEN
                  PERFORM EXITIUI.
      *  PF8 KEY
               IF EIBAID = DFHPF8 THEN
                  PERFORM RUNPF8.
...
               IF TRAN1I = 'USR1' THEN
                  EXEC CICS LINK PROGRAM('USR1PGM') END-EXEC.
               IF TRAN1I = 'USR2' THEN
                  EXEC CICS LINK PROGRAM('USR2PGM') END-EXEC.
...
      *        PARAGRAPHS FOR PF-KEY HANDLING
       EXITIUI.
               EXEC CICS XCTL PROGRAM('IESFPEP') END-EXEC.
       RUNPF8.
               EXEC CICS XCTL PROGRAM('SMAPEX2') END-EXEC.
```

General help for developing applications under CICS

For LE/VSE-related aspects on developing an application under CICS refer to *LE/VSE Programming Guide*, SC33-6684.

Interlanguage communication (ILC)

To get started on LE/VSE Interlanguage communication with two or more high-level languages (HLLs) and possibly Assembler, we recommend consulting *LE/VSE Writing ILC Applications*, SC33-6686. That manual contains many samples and a detailed discussion on the followings subjects:

- Getting started with LE/VSE ILC
- Communicating between C and COBOL
- Communicating between C and PL/I
- Communicating between COBOL and PL/I
- Communicating between multiple HLLs
- Communicating between Assembler and HLLs
- ILC under CICS
- Condition handling responses

Programming with other products

Beyond the above discussions, there are various LE/VSE interfaces, callable services, and further options available to involve other products. This serves to satisfy application programming requests for other products services like:

- Data Language/One (DL/1)
- DB2 Server for VSE & VM database management system (DB2)
- DFSORT/VSE (sort/merge/copying facility)
- Query Management Facility (QMF)

For a more detailed discussion refer to:

- *LE/VSE Programming Guide,* SC33-6684
- *LE/VSE C Run-Time Programming Guide,* SC33-6688

The latter includes samples for the interface between LE/VSE C Run-Time and DB2, DL/I, QMF.

Fourth generation language programming and VSE

With the availability of *VisualAge Generator EGL plug-in VSE* there evolved new Application Development (AD) opportunities for the VSE environment.

Based on IBM Rational Application Developer (RAD), an integrated development environment, it is possible to generate COBOL/VSE applications from a fourth generation language (4GL) that involve Web architectures.

The interaction of workstation and VSE components allows the generation and deployment of COBOL/VSE back-end programs that are called from an Enterprise Generation Language (EGL) front end (Java applications that are located on a Web Application Server and invoked from a Web browser).

See also 9.12, "Traditional languages and Web orientation in z/VSE" on page 247.

For a related overview, software prerequisites, and starter information refer to z/VSE Home Page and select **Solutions** → **Interoperability** → **Multiplatform Development and VSE**:

http://www.ibm.com/servers/eserver/zseries/zvse/solutions/egl.html

Notices

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Intel®	Windows®	Java™
Linux™	Microsoft®	UNIX®

The following terms are trademarks of other companies:

Java, JavaServer, JDBC, JSP, JVM, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Java™	JVM™	JDBC™
J2EE™	JSP™	JavaServer™

Excel, Microsoft, Visual Basic, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Microsoft®	Visual Basic®
Windows®	Excel®

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

Intel®	Pentium®

UNIX is a registered trademark of The Open Group in the United States and other countries.
UNIX®

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
Linux®

Other company, product, or service names may be trademarks or service marks of others.

E

Back matter

This appendix contains the following:

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this IBM Redbooks publication.

IBM provides access to z/VSE manuals on the Internet. To view, search, and print z/VSE manuals, visit the z/VSE Internet Library:

http://www.ibm.com/servers/eserver/zseries/zvse/documentation/

Mainframe architecture references

- *z/Architecture Principles of Operation,* SA22-7832

z/VSE JCL and utilities references

- *z/VSE System Control Statements,* SC33-8225
- *z/VSE System Utilities,* SC33-8234
- *VSE/POWER Administration and Operation,* SC33-8247

z/VSE system programming

- *z/VSE Guide to System Functions,* SC33-8233
- *z/VSE Planning,* SC33-8221
- *z/VSE Installation,* SC33-8222
- *z/VSE Administration,* SC33-8224

z/VSE communication references

- *z/VSE e-business Connectors, User's Guide,* SC33-8231
- *TCP/IP for VSE/ESA IBM Program Setup and Supplementary Information,* SC33-6601

Language references

- *HLASM General Information,* GC26-4943
- *HLASM Installation and Customization Guide,* SC26-3494
- *HLASM Language Reference,* SC26-4940
- *VSE/REXX Reference,* SC33-6642
- *REXX/VSE User's Guide,* SC33-6641
- *LE/VSE Concepts Guide,* GC33-6680
- *LE/VSE Programming Guide,* SC33-6684
- *LE/VSE Programming Reference,* SC33-6685
- *COBOL/VSE Language Reference,* SC26-8073

- *COBOL/VSE Programming Guide,* SC26-8072
- *PL/I VSE Language Reference,* SC26-8054
- *PL/I VSE Programming Guide,* SC26-8053
- *C/VSE Language Reference,* SC09-2425
- *C/VSE V1R1 User's Guide,* SC09-2424

CICS references

- *CICS Application Programming Guide,* SC33-1657
- *CICS Application Programming Reference,* SC33-1658

DL/I references

- *DL/I DOS/VS Application and Data Base Design,* SH24-5022
- *DL/I DOS/VS Application Programming: High Level Programming Interface,* SH24-5009

DB2 references

- *DB2 Server for VSE & VM Application Programming,* SC09-2889
- *DB2 Server for VSE & VM Database Administration,* SC09-2888
- *DB2 Server for VSE & VM SQL Reference,* SC09-2989

MQSeries (WebSphere MQ) references

- *MQSeries for VSE System Management Guide,* GC34-5364
- *MQSeries Application Programming Guide,* SC33-0807

IBM Redbooks

For information about ordering these publications see “How to get IBM Redbooks” on page 469. Note that some of the documents referenced here may be available in softcopy only:

- *Using MQSeries for VSE*, SG24-5647
- *CICS Transaction Server for VSE/ESA: CICS Web Support*, SG24-5997
- *e-business Connectivity for VSE/ESA*, SG24-5950
- *e-business Solutions for VSE/ESA*, SG24-5662

Online resources

These Web sites and URLs are also relevant as further information sources:

- z/VSE documentation

  http://www.ibm.com/servers/eserver/zseries/zvse/documentation/

- z/VSE Web site

  http://www-03.ibm.com/servers/eserver/zseries/zvse/

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

Glossary

A

abend. Abnormal end.

abnormal end. End of a task, job, or subsystem because of an error condition that cannot be resolved by recovery facilities while the task is performed. See also *abnormal termination*.

abnormal termination. (1) The end of processing prior to scheduled termination. (2) A system failure or operator action that causes a job to end unsuccessfully. Synonymous with *abend*, *abnormal end*.

ACB. Access control block.

access authority. An authority that relates to a request for a type of access to protected resources. In BSM, the access authorities are NONE, READ, UPDATE, and ALTER.

access control. A function of VSE that ensures that the system and the data and programs stored in it can be accessed only by authorized users in authorized ways.

access control table (DTSECTAB). A table used by the system to verify a user's right to access a certain resource.

access list. A list within a profile of all authorized users and their access authorities.

access method. A technique for moving data between main storage and I/O devices.

ACF/VTAM. Advanced communications function for virtual telecommunications access method. See also VTAM.

ACID properties. The properties of a transaction: atomicity, consistency, isolation, and durability. In CICS, the ACID properties apply to a unit of work (UoW).

address space. The complete range of addresses available to a program. In z/VSE, an address space can range up to 2 gigabytes of contiguous virtual storage addresses that the system creates for the user. Unlike a data space, an address space contains user data and programs, as well as system data and programs, some of which are common to all address spaces. See also *virtual address space*.

addressing mode (AMODE). A program attribute that refers to the address length that is expected to be in effect when the program is entered. In z/Architecture, addresses can be 24, 31, or 64 bits in length. z/VSE currently supports addresses of 24 and 31 bits in length.

administrator. A person responsible for administrative tasks such as access authorization and content management. Administrators can also grant levels of authority to users.

allocate. To assign a resource for use in performing a specific task.

alphanumeric character. A letter or a number.

AMODE. Addressing mode.

ANSI. American National Standards Institute.

AOR. Application-owning CICS region.

APAR. Authorized program analysis report.

API. Application programming interface.

APPC. Advanced Program-to-Program Communications.

application. A program or set of programs that performs a task. Some examples are payroll, inventory management, and word processing applications.

application-owning region (AOR). In a CICSPlex configuration, a CICS region devoted to running applications.

application program. A collection of software components used to perform specific types of work on a computer, such as a program that does inventory control or payroll.

APPN. Advanced Peer-to-Peer Network.

ASCII (American Standard Code for Information Interchange). A character encoding based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that work with text. Most modern character encodings—which support many more characters—have a historical basis in ASCII. ASCII was first published as a standard in 1967 and was last updated in 1986.

ASI (automated system initialization) procedure. A set of control statements that specifies values for an automatic system initialization.

assembler. A computer program that converts assembler language instructions into binary machine language (object code).

assembler language. A symbolic programming language that comprises instructions for basic computer operations that are structured according to the data formats, storage structures, and registers of the computer.

asynchronous processing. A series of operations that are done separately from the job in which they were requested. For example, submitting a batch job from an interactive job at a work station. See also *synchronous processing*.

ATM. Automated teller machine.

audit. To review and examine the activities of a data processing system mainly to test the adequacy and effectiveness of procedures for data security and data accuracy.

authority. The right to access objects, resources, or functions.

authorization checking. The action of determining whether a user is permitted access to a BSM-protected resource.

authorized program analysis report (APAR). A request for correction of a problem caused by a defect in a current unaltered release of a program. The correction is called a PTF.

automated operations. Automated procedures to replace or simplify actions of operators in both systems and network operations.

automated system initialization (ASI). A function that allows control information for system startup to be cataloged for automatic retrieval during system startup.

autostart. A facility that starts up VSE/POWER with little or no operator involvement.

auxiliary storage. All addressable storage other than processor storage.

B

background. (1) In multiprogramming, the environment in which low-priority programs are executed. (2) Under Interactive Interface the environment in which jobs are submitted through the SUBMIT. One job step at a time is assigned to a region of central storage, and it remains in central storage to completion. Contrast with *foreground*.

background partition. An area of virtual storage in which programs are executed under control of the system. By default, the partition has a processing priority lower than any of the existing foreground partitions.

backout. A request to remove all changes to resources since the last commit or backout or, for the first unit of recovery, since the beginning of the application. Backout is also called *rollback* or *abort*.

backup. The process of creating a copy of a data set to ensure against accidental loss.

BAM. Basic access method. It includes BDAM and BSAM.

basic security manager (BSM). A security manager supplied by z/VSE. The BSM provides access control by identifying and verifying the users to the system, authorizing access to protected resources, logging the detected unauthorized attempts to enter the system, and logging the detected accesses to protected resources.

batch. A group of records or data processing jobs brought together for processing or transmission. Pertaining to activity involving little or no user action. Contrast with *interactive*.

batch job. A predefined group of processing actions submitted to the system to be performed with little or no interaction between the user and the system. Contrast with *interactive job*.

batch processing. A method of running a program or a series of programs in which one or more records (a batch) are processed with little or no action from the user or operator. Contrast with *interactive processing*.

BDAM. Basic direct access method (used for ICCF).

big endian. A format for the storage of binary data in which the most significant byte is placed first. Big endian is used by most hardware architectures including the z/Architecture. Contrast with *little endian*.

binary data. (1) Any data not intended for direct human reading. Binary data may contain unprintable characters, outside the range of text characters. (2) A type of data consisting of numeric values stored in bit patterns of 0s and 1s. Binary data can cause a large number to be placed in a smaller space of storage.

bind. (1) To combine one or more control sections or program modules into a single program module, resolving references between them. (2) In SNA, a request to activate a session between two logical units (LUs).

block. Usually, a block consists of several records of a file that are transmitted as a unit. But if records are very large, a block can also be part of a record only. On an FBA disk, a block is a string of 512 bytes of data.

block size. (1) The number of data elements in a block. (2) A measure of the size of a block, usually specified in units such as records, words, computer words, or characters. (3) Synonymous with *block length*. (4) Synonymous with *physical record size*.

BSAM. Basic sequential access method.

BSM. Basic security manager.

buffer. A portion of storage used to hold input or output data temporarily.

byte. The basic unit of storage addressability. It has a length of 8 bits.

byte stream. A simple sequence of bytes stored in a stream file. See also *record data*.

C

C language. A high-level language used to develop software applications in compact, efficient code that can be run on different types of computers with minimal change.

cabinet. Housing for panels organized into port groups of patchports, which are pairs of fibre adapters or couplers. Cabinets are used to organize long, complex cables between processors and controllers, which may be as far away as another physical site. Also known as *fiber management cabinets*.

cable *in inventory*. Unused cables.

cache. A random access electronic storage in selected storage controls used to retain frequently used data for faster access by the channel.

called routine. A routine or program that is invoked by another.

carriage control character. An optional character in an input data record that specifies a write, space, or skip operation.

carriage return (CR). (1) A keystroke generally indicating the end of a command line. (2) In text data, the action that indicates to continue printing at the left margin of the next line. (3) A character that will cause printing to start at the beginning of the same physical line in which the carriage return occurred.

CART. Command and response token.

case-sensitive. Pertaining to the ability to distinguish between uppercase and lowercase letters.

catalog. (1) A directory of files and libraries, with reference to their locations. (2) To store a library member such as a phase, module, or book in a sublibrary. (3) The collection of all data set indexes that are used by the control program to locate a volume containing a specific data set.
z/VSE uses a VSAM master catalog and one or more VSAM user catalogs.

cataloged data set. A data set that is represented in an index or hierarchy of indexes that provide the means for locating it.

cataloged procedure. A set of job control language (JCL) statements placed in a library and retrievable by name.

CCW. Channel command word.

CEMT. The CICS-supplied transaction that allows checking of the status of terminals, connections, and other CICS entities from a console or from CICS terminal sessions.

central processor (CP). The part of the computer that contains the sequencing and processing facilities for instruction execution, initial program load, and other machine operations.

central processor complex (CPC). A physical collection of hardware that includes main storage, one or more central processors, timers, and channels.

central processing unit (CPU). Synonymous with *processor*.

central storage. (1) In z/VSE, the storage of a computing system from which the central processing unit can directly obtain instructions and data, and to which it can directly return results. (Formerly referred to as *real storage*.) (2) Synonymous with *processor storage*.

CGI. Common Gateway Interface.

channel adapter. A device that groups two or more controller channel interfaces electronically.

channel connection address (CCA). The input/output (I/O) address that uniquely identifies an I/O device to the channel during an I/O operation.

channel interface. The circuitry in a storage control that attaches storage paths to a host channel.

channel path identifier. The logical equivalent of channels in the physical processor.

channel subsystem (CSS). A collection of subchannels that directs the flow of information between I/O devices and main storage. Logical partitions use subchannels to communicate with I/O devices. The maximum number of CSSs supported by a processor also depends on the processor type. If more than one CSS is supported by a processor, each CSS has a processor unique single hexadecimal digit CSS identifier (CSS ID).

channel-to-channel (CTC). The communication (transfer of data) between programs on opposite sides of a channel-to-channel adapter (CTCA).

channel-to-channel adapter (CTCA). An input/output device that is used by program in one system to communicate with a program in another system.

channel-to-channel (CTC) connection. A connection between two CHPIDs on the same or different processors, either directly or through a switch. When connecting through a switch, both CHPIDs must be connected through the same or a chained switch.

character. A letter, digit, or other symbol. A letter, digit, or other symbol that is used as part of the organization, control, or representation of data. A character is often in the form of a spatial arrangement of adjacent or connected strokes.

character set. A defined set of characters with no coded representation assumed that can be recognized by a configured hardware or software system. A character set may be defined by alphabet, language, script, or any combination of these items.

checkpoint. (1) A place in a routine where a check, or a recording of data for restart purposes, is performed. (2) A point at which information about the status of a job and the system can be recorded so that the job step can be restarted later.

checkpoint data set. A data set in which information about the status of a job and the system can be recorded so that the job step can be restarted later.

checkpoint write. Any write to the checkpoint data set. A general term for the primary, intermediate, and final writes that update any checkpoint data set.

CHPID. Channel path identifier.

CI. Control interval.

CICS. Customer Information Control System.

CICS Transaction Server for VSE/ESA. This is the successor system to CICS/VSE.

CICSplex. A configuration of interconnected CICS systems in which each system is dedicated to one of the main elements of the overall workload. See also *application owning region* and *terminal owning region*.

CICS/VSE. Customer Information Control System/VSE.

CKD. Count-key data.

client. A functional unit that receives shared services from a server. See also *client-server*.

client-server. In TCP/IP, the model of interaction in distributed data processing in which a program at one site sends a request to a program at another site and awaits a response. The requesting program is called a client. The answering program is called a server.

CMOS. Complementary Metal Oxide Semiconductor.

CMS. Conversational Monitor System.

COBOL. Common Business-Oriented Language.

coded character set. A set of unambiguous rules that establishes a character set and the one-to-one relationships between the characters of the set and their coded representations.

code page. (1) An assignment of graphic characters and control function meanings to all code points; for example, assignment of characters and meanings to 256 code points for an 8-bit code, assignment of characters and meanings to 128 code points for a 7-bit code. (2) A particular assignment of hexadecimal identifiers to graphic characters.

code point. A 1-byte code representing one of 256 potential characters.

coexistence. Two or more systems at different levels (for example, software, service, or operational levels) that share resources. Coexistence includes the ability of a system to respond in the following ways to a new function that was introduced on another system with which it shares resources: ignore a new function; terminate gracefully; support a new function.

command. A request to perform an operation or run a program. When parameters, arguments, flags, or other operands are associated with a command, the resulting character string is a single command.

command and response token (CART). A parameter on WTO, WTOR, MGCRE, and REXX execs that allows you to link commands and their associated message responses.

command prefix. A one-character to eight-character command identifier. The command prefix distinguishes the command as belonging to an application or subsystem rather than to z/VSE.

COMMAREA. A communication area made available to applications running under CICS.

commit. A request to make all changes to resources since the last commit or backout or, for the first unit of recovery, since the beginning of the application.

Common Business-Oriented Language (COBOL). A high-level language, based on English, that is primarily used for business applications.

common library. An interactively accessible library that can be accessed by any user of the system or subsystem that owns the library.

compatibility. Ability to work in the system or ability to work with other devices or programs.

compilation unit. A portion of a computer program sufficiently complete to be compiled correctly.

compile. To translate a source program into an executable program (an object program). See also *assembler*.

compiler. A program that translates a source program into an executable program (an object deck).

compiler options. Keywords that can be specified to control certain aspects of compilation. Compiler options can control the nature of the load module generated by the compiler, the types of printed output to be produced, the efficient use of the compiler, and the destination of error messages. Also called compile-time options.

complementary metal oxide semiconductor (CMOS). A technology that combines the electrical properties of positive and negative voltage requirements to use considerably less power than other types of semiconductors.

component. A functional part of an operating system (for example, the scheduler or supervisor).

condition code. A code that reflects the result of a previous input/output, arithmetic, or logical operation.

conditional job control. The capability of the job control program to process or to skip one or more statements based on a condition that is tested by the program.

configuration. The arrangement of a computer system or network as defined by the nature, number, and chief characteristics of its functional units.

connection. In TCP/IP, the path between two protocol applications that provides reliable data stream delivery service. In Internet communications, a connection extends from a TCP application on one system to a TCP application on another system.

consistent copy. A copy of a data entity (for example, a logical volume) that contains the contents of the entire data entity from a single instant in time.

console. Any device from which operators can enter commands or receive messages.

control area (CA). In VSE/VSAM, a group of control intervals used as a unit for formatting a data set before adding records to it. Also, in a key-sequenced data set, the set of control intervals, pointed to by a sequence-set index record, that is used by VSAM for distributing free space and for placing a sequence-set index record adjacent to its data.

control block. A storage area used by a computer program to hold control information.

control interval (CI). A fixed-length area or disk in which VSAM stores records and creates distributed free space. Also, in a key-sequenced data set or file, the set of records that an entry in the sequence-set index record points to. The control interval is the unit of information that VSAM transmits to or from disk. A control interval always includes an integral number of physical records. For FBA, it must be an integral multiple, to be defined at cluster definition, of the block size.

control section (CSECT). The part of a program specified by the programmer to be a relocatable unit, all elements of which are to be loaded into adjoining main storage locations.

control statement. In programming languages, a statement that is used to alter the continuous sequential execution of statements. A control statement can be a conditional statement, such as IF, or an imperative statement, such as RETURN. In JCL, a statement in a job that is used in identifying the job or describing its requirements to the operating system.

control unit (CU). Each physical controller contains one or more control units, which translate high-level requests to low-level requests between processors and devices. Synonymous with *device control unit*.

control unit address. The high order bits of the storage control address, used to identify the storage control to the host system.

controller. A device that translates high-level requests from processors to low-level requests for I/O devices, and vice versa. Each physical controller contains one or more logical control units, channel and device interfaces, and a power source. Controllers can be divided into segments, or grouped into subsystems.

conversation. A logical connection between two programs over an LU type 6.2 session that allows them to communicate with each other while processing a transaction.

conversational. Pertaining to a program or a system that carries on a dialog with a terminal user, alternately accepting input and then responding to the input quickly enough for the user to maintain a train of thought.

conversational monitor system (CMS). A virtual machine operating system that provides general interactive time sharing, problem solving, and program development capabilities, and operates only under the control of the z/VM control program.

CORBA. Common Object Request Broker Architecture.

count-key data. A disk storage device for storing data in the format: count field normally followed by a key field followed by the actual data of a record. The count field contains, in addition to other information, the address of the record in the format: CCHHR (where CC is the two-digit cylinder number, HH is the two-digit head number, and R is the record number) and the length of the data. The key field contains the record's key.

CP. Central processor.

CPC. Central processor complex.

CPU. Central processing unit.

cross-memory linkage. A method for invoking a program in a different address space. The invocation is synchronous with respect to the caller.

cryptographic key. A parameter that determines cryptographic transformations between plaintext and ciphertext.

cryptography. The transformation of data to conceal its meaning.

CSS. Channel subsystem.

CSECT. Control section.

CTC. Channel-to-channel.

CTC connection. Channel-to-channel connection.

Customer Information Control System (CICS). An online transaction processing (OLTP) system that provides specialized interfaces to databases, files, and terminals in support of business and commercial applications. CICS enables transactions entered at remote terminals to be processed concurrently by user-written application programs.

D

daemon. In UNIX systems, a long-lived process that runs unattended to perform continuous or periodic system-wide functions, such as network control. Some daemons are triggered automatically to perform their task. Others operate periodically. An example is the cron daemon, which periodically performs the tasks listed in the crontab file. The z/OS equivalent is a started task.

DASD. Direct access storage device.

DASD sharing. An option that lets independent computer systems use common data on shared disk devices.

DASD volume. A DASD space identified by a common label and accessed by a set of related addresses. See also *volume*.

data control block (DCB). A control block used by access method routines in storing and retrieving data.

data entry panel. A panel in which the user communicates with the system by filling in one or more fields. See also *panel* and *selection panel*.

data division. In COBOL, the part of a program that describes the files to be used in the program and the records contained within the files. It also describes any WORKING-STORAGE data items, LINKAGE SECTION data items, and LOCAL-STORAGE data items that are needed.

Data Facility Sort (DFSORT). An IBM licensed program that is a high-speed data-processing utility. DFSORT provides a method for sorting, merging, and copying operations, as well as providing versatile data manipulation at the record, field, and bit level.

data integrity. The condition that exists when accidental or intentional destruction, alteration, or loss of data does not occur.

Data Interfile Transfer, Testing and Operations (DITTO) utility. An IBM program that provides file-to-file services for card I/O, tape, and disk devices. The latest version is called DITTO/ESA for VSE.

data set. In z/VSE, a named collection of related data records that is stored and retrieved by an assigned name. Equivalent to a file.

data set backup. Backup to protect against the loss of individual data sets.

data stream. (1) All information (data and control commands) sent over a data link usually in a single read or write operation. (2) A continuous stream of data elements being transmitted, or intended for transmission, in character or binary-digit form, using a defined format.

data type. The properties and internal representation that characterize data.

data warehouse. A system that provides critical business information to an organization. The data warehouse system cleanses the data for accuracy and currency, and then presents the data to decision makers so that they can interpret and use it effectively and efficiently.

database. A collection of tables, or a collection of table spaces and index spaces.

database administrator (DBA). An individual who is responsible for designing, developing, operating, safeguarding, maintaining, and using a database.

database management system (DBMS). A software system that controls the creation, organization, and modification of a database and the access to the data that is stored within it.

DBCS. Double-byte character set.

DBMS. Database management system.

DB2. Database 2. Generally, one of a family of IBM relational database management systems and, specifically, the system that runs under z/VSE.

DB2 data sharing group. A collection of one or more concurrent DB2 subsystems that directly access and change the same data while maintaining data integrity.

DCB. Data control block.

deadlock. (1) An error condition in which processing cannot continue because each of two elements of the process is waiting for an action by or a response from the other. (2) Unresolvable contention for the use of a resource. (3) An impasse that occurs when multiple processes are waiting for the availability of a resource that does not become available because it is being held by another process that is in a similar wait state.

Debug Tool for VSE/ESA. An IBM software product to detect, diagnose, and eliminate errors in programs generated with LE/VSE-conforming compilers.

dedicated. Pertaining to the assignment of a system resource—a device, a program, or a whole system—to an application or purpose.

default. A value that is used or an action that is taken when no alternative is specified by the user.

destination. A combination of a node name and one of the following: a user ID, a remote printer or punch, a special local printer, or LOCAL (the default if only a node name is specified).

destination node. The node that provides application services to an authorized external user.

device. A computer peripheral or an object that appears to the application as such.

device address. (1) The identification of an input/output device by its channel and unit number. (2) In data communication, the identification of any device to which data can be sent or from which data can be received.

device control unit. A hardware device that controls the reading, writing, or displaying of data at one or more I/O devices or terminals.

device number. A four-hexadecimal-character identifier (for example, 13A0) that you associate with a device to facilitate communication between the program and the host operator. The device number that you associate with a subchannel.

Device Support Facilities program (ICKDSF). A program used to initialize DASD volumes at installation and perform media maintenance.

device type. The general name for a kind of device (for example, 3390).

DFSORT. Data Facility Sort.

dialog. (1) An interactive pop-up window containing options that allow you to browse or modify information, take specific action relating to selected objects, or access other dialogs. (2) For z/VSE, a set of panels that can be used to complete a specific task (for example, defining a file).

direct access. Accessing data on a storage device using their address and not their sequence. This is the typical access on disk devices as opposed to magnetic tapes. Contrast with *sequential access*.

direct access storage device (DASD). A device in which the access time is effectively independent of the location of the data.

directory. (1) A type of file containing the names and controlling information for other files or other directories. Directories can also contain subdirectories, which can contain subdirectories of their own. (2) A file that contains directory entries. No two directory entries in the same directory can have the same name (POSIX.1). (3) A file that points to files and to other directories. (4) An index used by a control program to locate blocks of data that are stored in separate areas of a data set in direct access storage. (5) In VSE, specifically, the index for the program libraries. See also *library directory* and *sublibrary directory*.

disaster recovery. Recovery after a disaster, such as a fire, that destroys or otherwise disables a system. Disaster recovery techniques typically involve restoring data to a second (recovery) system, then using the recovery system in place of the destroyed or disabled application system. See also *recovery*, *backup*, and *recovery system*.

disk operating system residence volume (DOSRES). The z/VSE disk volume on which the system sublibrary IJSYSRS.SYSLIB is located, including the programs and procedures required for system startup.

DISP. Disposition, indicating to VSE/POWER how job input and output is to be handled. A job may, for example, be deleted or kept after processing.

display console. In z/VSE, an MCS console whose input/output function you can control.

distributed computing. Computing that involves the cooperation of two or more machines communicating over a network. Data and resources are shared among the individual computers.

Distributed Computing Environment (DCE). A comprehensive, integrated set of services that supports the development, use, and maintenance of distributed applications. DCE is independent of the operating system and network. It provides interoperability and portability across heterogeneous platforms.

distributed data. Data that resides on a DBMS other than the local system.

Distributed File Service (DFS). A DCE component. DFS joins the local file systems of several file server machines, making the files equally available to all DFS client machines. DFS allows users to access and share files stored on a file server anywhere in the network, without having to consider the physical location of the file. Files are part of a single, global namespace, so that a user can be found anywhere in the network by means of the same name.

DOSRES. Disk operating system residence volume.

double-byte character set (DBCS). A set of characters in which each character is represented by a two-bytes code. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires two bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS. Contrast with *single-byte character set*.

doubleword. A sequence of bits or characters that comprises eight bytes (two 4-byte words) and is referenced as a unit.

downwardly compatible. The ability of applications to run on previous releases of z/VSE.

drain. Allowing a printer to complete its current work before stopping the device.

driving system. The system used to install the program. Contrast with *target system*.

dummy device. A device address with no real I/O device behind it. Input and output for that device address are spooled on disk.

dump. A report showing the contents of storage. Dumps are typically produced following program failures, for use as diagnostic aids.

dynamic allocation. Assignment of system resources to a program at the time the program is executed rather than at the time it is loaded into central storage.

dynamic class table. Defines the characteristics of dynamic partitions.

dynamic partition. A partition created and activated on an *as needed* basis that does not use fixed static allocations. After processing, the occupied space is released. Contrast with *static partition*.

dynamic reconfiguration. The ability to make changes to the channel subsystem and to the operating system while the system is running.

E

e-business. (1) The transaction of business over an electronic medium such as the Internet. (2) The transformation of key business processes through the use of Internet technologies.

EB. See *exabyte*.

EBCDIC. Extended Binary Coded Decimal Interchange Code. EBCDIC was devised in 1963 and 1964 by IBM and was announced with the release of the IBM System/360 line of mainframe computers. It was created to extend the Binary-Coded Decimal that existed at the time. EBCDIC was developed separately from ASCII. EBCDIC is an 8-bit encoding, versus the 7-bit encoding of ASCII.

EC. Engineering change.

enclave. A transaction that can span multiple dispatchable units (SRBs and tasks) in one or more address spaces and is reported on and managed as a unit.

encoding scheme. The set of rules that specifies the values for control characters and graphic characters. Examples of encoding schemes include ASCII, ISO/IEC 10646, Unicode, and IBM EBCDIC.

encrypt. To systematically encode data so that it cannot be read without knowing the coding key.

endian. An attribute of data representation that reflects how certain multi-octet data is stored in memory. See big endian and little endian.

enterprise. The composite of all operational entities, functions, and resources that form the total business concern.

Enterprise Systems Connection (ESCON). A set of products and services that provides a dynamically connected environment using optical cables as a transmission medium.

entry name. In assembler language, a programmer-specified name within a control section that identifies an entry point and can be referred to by any control section. See also *entry point*.

entry point. The address or label of the first instruction that is executed when a routine is entered for execution. Within a load module, the location to which control is passed when the load module is invoked.

entry point name. The symbol (or name) that represents an entry point. See also *entry point*.

EOF. End of file.

ESCON. Enterprise Systems Connection.

exabyte. For processor, real and virtual storage capacities and channel volume: 1 152 921 504 606 846 976 bytes or $2^{(60)}$.

EXCP. Execute channel programs.

executable. A load module or program object that has yet to be loaded into memory for execution.

executable program. (1) A program in a form suitable for execution by a computer. The program can be an application or a shell script. (2) A program that has been link-edited and can therefore be run in a processor. (3) A program that can be executed as a self-contained procedure. It consists of a main program and, optionally, one or more subprograms. (4) See also *executable file*, *load module*.

Extended Binary-Coded Decimal Interchange Code (EBCDIC). An encoding scheme that is used to represent character data in the z/OS environment. Contrast with *ASCII* and *Unicode*.

external reference. In an object deck, a reference to a symbol, such as an entry point name, defined in another program or module.

F

fast service upgrade (FSU). A service function of z/VSE for the installation of a refresh release without regenerating control information such as library control tables.

FBA disk device. Fixed-block architecture disk device.

feature. A part of an IBM product that may be ordered separately by a customer.

feature code. A four-digit code used by IBM to process hardware and software orders.

fetch. The dynamic loading of a procedure.

Fiber Connection Environment (FICON). An optical fiber communication method offering channels with high data rate, high bandwidth, increased distance, and a large number of devices per control unit for mainframe systems. It can work with, or replace, ESCON links.

fiber link. The physical fiber optic connections and transmission media between optical fiber transmitters and receivers. A fiber link can comprise one or more fiber cables and patchports in fiber management cabinets. Each connection in the fiber link is either permanent or mutable.

FICON. Fiber Connection Environment.

FIFO. First in, first out.

file. A named collection of related data records that is stored and retrieved by an assigned name. Equivalent to a z/VSE data set.

f**irst in, first out.** A queuing technique in which the next item to be retrieved is the oldest item in the queue.

firewall. An intermediate server that functions to isolate a secure network from an insecure network.

fix. A correction of an error in a program, usually a temporary correction or bypass of defective code.

fixed-length record. A record having the same length as all other records with which it is logically or physically associated. Contrast with *variable-length record*.

FlashCopy. A point-in-time copy services function that can quickly copy data from a source location to a target location.

foreground. In multiprogramming, the environment in which high-priority programs are executed. Contrast with *background*.

fork. To create and start a child process. Forking is similar to creating an address space and attaching. It creates a copy of the parent process, including open file descriptors.

Fortran. A high-level language used primarily for applications involving numeric computations. In previous usage, the name of the language was written in all capital letters, that is, FORTRAN.

frame. For a mainframe microprocessor cluster, a frame contains one or two central processor complexes (CPCs), support elements, and AC power distribution.

FTP. File Transfer Protocol.

FULIST (FUnction LIST). A type of selection panel that displays a set of files or functions for the choice of the user.

fullword. A sequence of bits or characters that comprises four bytes (one word) and is referenced as a unit.

fullword boundary. A storage location whose address is evenly divisible by 4.

G

gateway node. A node that is an interface between networks.

GB. Gigabyte (1 073 741 824 bytes).

GETVIS space. Storage space within a partition or the shared virtual area, available for dynamic allocation to programs.

gigabyte. 2^{30} bytes, 1 073 741 824 bytes. This is approximately a billion bytes in American English.

Gregorian calendar. The calendar in use since Friday, 15 October 1582 throughout most of the world.

group. A collection of BSM users who can share access authorities for protected resources.

H

hardcopy file. A system file on disk, used to log all lines of communication between the system and the operator at the system console, to be printed on request.

hardware. Physical equipment, as opposed to the computer program or method of use
(for example, mechanical, magnetic, electrical, or electronic devices). Contrast with *software*.

Hardware Management Console (HMC). A console used to monitor and control hardware such as the mainframe microprocessors.

hardware unit. A central processor, storage element, channel path, device, and so on.

head of string. The first unit of devices in a string. It contains the string interfaces that connect to controller device interfaces.

hexadecimal. A base 16 numbering system. Hexadecimal digits range from 0 through 9 (decimal 0 to 9) and uppercase or lowercase A through F (decimal 10 to 15) and A through F, giving values of 0 through 15.

high-level language (HLL). A programming language above the level of assembler language and below that of program generators and query languages. Examples are C, C++, COBOL, Fortran, and PL/I.

HLL. High-level language.

HMC. Hardware Management Console.

host transfer file (HTF). Used by the Workstation File Transfer Support of z/VSE as an intermediate storage area for files that are sent to and from IBM personal computers.

I

I/O. Input/output.

I/O cluster. A sysplex that owns a managed channel path for a logically partitioned processor configuration.

I/O device. A printer, tape drive, hard disk drive, and so on. Devices are logically grouped inside units, which are in turn grouped into strings. The first unit, known as the head of string, contains string interfaces that connect to controller device interfaces and eventually to processor CHPIDs. Devices are represented as lines of text within the appropriate unit object in the configuration diagram.

IBM Support Center. The IBM organization responsible for software service.

IBM systems engineer (SE). An IBM service representative who performs maintenance services for IBM software in the field.

ICSF. Integrated Cryptographic Service Facility.

IDCAMS. An IBM program used to process access method services commands.

image. A single instance of the z/VSE operating system.

initial program load (IPL). The initialization procedure that causes the z/VSE operating system to begin operation. During IPL, system programs are loaded into storage and z/VSE is made ready to perform work. Synonymous with *boot*, *load*.

installation exit. The means by which an IBM software product may be modified by a customer's system programmers to change or extend the functions of the product.

interactive. Pertaining to a program or system that alternately accepts input and responds. In an interactive system, a constant dialog exists between user and system. Contrast with *batch*.

Interactive interface. A system facility that controls how different users see and work with the system by means of user profiles. When signing on, the interactive interface makes available those parts of the system authorized by the profile. The interactive interface has sets of selection-entry and data-entry panels through which users communicate with the system.

interactive partition. An area of virtual storage for the purpose of processing a job that was started interactively via VSE/ICCF.

interrupt. A suspension of a process, such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed.

IPL. initial program load.

IPv6. Internet Protocol Version 6.

J

JCL. Job control language.

JECL. Job entry control language.

job. A unit of work for an operating system. Jobs are defined by JCL statements.

job control language (JCL). A language that serves to prepare a job or each job step of a job to be run. Some of its functions are to identify the job, to determine the I/O devices to be used, set switches for program use, log (or print) its own statements, and fetch the first phase of each job step. In a job, such statements have the prefix '//'.

job entry control language (JECL). A language used to tell VSE/POWER how to manage the collection and scheduling of job input statements for partitions and the spooling of output from partitions. These statements begin with the prefix '* $$'.

job step. One of a group of related programs complete with the JCL statements necessary for a particular run. Every job step is identified in the job stream by an EXEC statement under one JOB statement for the whole job.

Julian date. A date format that contains the year in positions 1 and 2, and the day in positions 3 through 5. The day is represented as 1 through 366, right-adjusted, with zeros in the unused high-order position.

K

key-sequenced data set (KSDS). A VSAM file or data set whose records are loaded in ascending key sequence and controlled by an index. Records are retrieved and stored by keyed access or by addressed access, and new records are inserted in key sequence by means of distributed free space. Relative byte addresses can change because of control interval or control area splits.

KSDS. Key-sequenced data set.

L

LAN. Local area network.

Language Environment. Short form of Language Environment for z/VSE. A set of architectural constructs and interfaces that provides a common run-time environment and run-time services for C, COBOL, and PL/I applications compiled by Language Environment-conforming compilers.

last in, first out (LIFO). A queuing technique in which the next item to be retrieved is the item most recently placed in the queue.

LCSS. Logical channel subsystem.

LCU. Logical control unit.

LDAP. Lightweight Directory Access Protocol.

LE/VSE. Short form of Language Environment for z/VSE.

librarian. The set of programs that maintains, services, and organizes the system and private libraries.

library. A file that contains a collection of related files. See *VSE library* and *VSE/ICCF library*.

library member. The smallest unit of data that can be stored in and retrieved from a VSE sublibrary or a VSE/ICCF library.

Lightweight Directory Access Protocol (LDAP). An Internet protocol standard, based on the TCP/IP protocol, which allows the access and manipulation of data organized in a Directory Information Tree (DIT).

LIFO. Last in, first out.

linkage editor. A program used to create a phase (executable code) from one or more independently translated object modules, from one or more existing phases, or from both. In creating the phase, the linkage editor resolves cross references among the modules and phases available as input. The program can catalog the newly built phases.

link-edit. To create a loadable computer program by means of a linkage editor or binder.

little endian. A format for storage of binary data in which the least significant byte is placed first. Little endian is used by the Intel hardware architectures. Contrast with *big endian*.

local area network (LAN). A network in which communication is limited to a moderate-sized geographical area (1 to 10 km) such as a single office building, warehouse, or campus, and that does not generally extend across public rights-of-way. A local network depends on a communication medium capable of moderate to high data rate (greater than 1 Mbps), and normally operates with a consistently low error rate.

lock file. In a shared disk environment under VSE, a system file on disk used by the sharing systems to control their access to shared data.

logical control unit (LCU). A single control unit (CU) with or without attached devices, or a group of one or more CUs that share devices. In a channel subsystem (CSS), an LCU represents a set of CUs that physically or logically attach I/O devices in common.

logical partition (LP). A subset of the processor hardware that is defined to support an operating system. See also *logically partitioned (LPAR) mode*.

logical partitioning. A function of an operating system that enables the creation of logical partitions.

logical record. A user record, normally pertaining to a single subject and processed by data management as a unit. Contrast with *physical record,* which may be larger or smaller.

logical subsystem. The logical functions of a storage controller that allow one or more host I/O interfaces to access a set of devices. The controller aggregates the devices according to the addressing mechanisms of the associated I/O interfaces. One or more logical subsystems exist on a storage controller. In general, the controller associates a given set of devices with only one logical subsystem.

logical unit (LU). In SNA, a port through which an end user accesses the SNA network in order to communicate with another end user, and through which the end user accesses the functions provided by system services control points (SSCPs).

logical unit type 6.2. The SNA logical unit type that supports general communication between programs in a cooperative processing environment.

logically partitioned (LPAR) mode. A central processor complex (CPC) power-on reset mode that enables use of the PR/SM feature and allows an operator to allocate CPC hardware resources (including central processors, central storage, expanded storage, and channel paths) among logical partitions.

logoff. (1) The procedure by which a user ends a terminal session. (2) In VTAM, a request that a terminal be disconnected from a VTAM application program.

logon. (1) The procedure by which a user begins a terminal session. (2) In VTAM, a request that a terminal be connected to a VTAM application program.

loop. A situation in which an instruction or a group of instructions execute repeatedly.

LP. Logical partition.

LPAR. Logically partitioned (mode).

LU. Logical unit.

M

machine check interruption. An interruption that occurs as a result of an equipment malfunction or error.

machine readable. Pertaining to data a machine can acquire or interpret (read) from a storage device, a data medium, or other source.

macro. An instruction in a source language that is to be replaced by a defined sequence of instructions in the same source language.

main task. In the context of z/VSE multitasking, the main program in a multitasking environment.

master catalog. A catalog that contains extensive data set and volume information that VSAM requires to locate data sets, to allocate and deallocate storage space, to verify the authorization of a program or operator to gain access to a data set, and to accumulate usage statistics for data sets.

master console. In z/VSE, one or more consoles that receive all system messages, except for those that are directed to a particular console. The operator of a master console can reply to all outstanding messages and enter all system commands.

MB. Megabyte.

megabyte (MB). 2^{20} bytes, 1 048 576 bytes.,048,576 bytes.

member. The smallest unit of data that can be stored in and retrieved from a VSE sublibrary or a VSE/ICCF library.

message. (1) In VSE, a communication sent from a program to the operator or user. It can appear on a console, a display terminal, or a printout. (2) In telecommunication, a logical set of data being transmitted from one node to another.

microcode. Stored microinstructions, not available to users, that perform certain functions.

microprocessor. A processor implemented on one or a small number of chips.

migration. Refers to activities, often performed by the system programmer, that relate to the installation of a new version or release of a program to replace an earlier level. Completion of these activities ensures that the applications and resources on a system will function correctly at the new level.

modification level. A distribution of all temporary fixes that have been issued since the previous modification level. A change in modification level does not add new functions or change the programming support category of the release to which it applies. Contrast with *release* and *version*. Whenever a new release of a program is shipped, the modification level is set to 0. When the release is reshipped with the accumulated services changes incorporated, the modification level is incremented by 1.

module. The object that results from compiling source code. A module cannot be run. To be run, a module must be bound into a program.

multiprocessing. The simultaneous execution of two or more computer programs or sequences of instructions. See also *parallel processing*.

multiprocessor (MP). A CPC that can be physically partitioned to form two operating processor complexes.

multitasking. Mode of operation that provides for the concurrent, or interleaved, execution of two or more tasks, or threads. Synonymous with *multithreading*.

N

n-way. The number (n) of CPs in a CPC. For example, a 6-way CPC contains six CPs.

NCP. Network Control Program.

network. A collection of data processing products connected by communications lines for exchanging information between stations.

network job entry (NJE). An architecture that provides for the passing of jobs, system output data, operator commands, and messages between communicating systems connected by binary-synchronous communication lines, channel-to-channel adapters, VTAM-controlled SNA links, or TCP/IP links. Today, this architecture is used by VSE/POWER, RSCS, AS/400, JES2, and JES3 to exchange data between each other.

network operator. (1) The person responsible for controlling the operation of a telecommunication network. (2) A VTAM application program authorized to issue network operator commands.

next sequential instruction. The next instruction to be executed in the absence of any branch or transfer of control.

nonreentrant. A type of program that cannot be shared by multiple users.

nonstandard labels. Labels that do not conform to American National Standard or IBM System/370 standard label conventions.

null. Empty. Having no meaning.

O

object deck. A collection of one or more control sections produced by an assembler or compiler and used as input to the linkage editor or binder. Also called object code or simply OBJ.

object module. A module that is the output from a language translator (such as a compiler or an assembler). An object module is in relocatable format with machine code that is not executable. Before an object module can be executed, it must be processed by the link-edit utility.

offline. Pertaining to equipment or devices not under control of the processor.

offset. The number of measuring units from an arbitrary starting point in a record, area, or control block, to some other point.

online. Pertaining to a user's ability to interact with a computer.

operating system. Software that controls the running of programs. In addition, an operating system may provide services such as resource allocation, scheduling, I/O control, and data management. Although operating systems are predominantly software, partial hardware implementations are possible.

operator commands. Statements that system operators may use to get information, alter operations, initiate new operations, or end operations.

operator message. A message from an operating system directing the operator to perform a specific function, such as mounting a tape reel, or informing the operator of specific conditions within the system, such as an error condition.

overlay. To write over existing data in storage.

P

page. (1) In virtual storage systems, a fixed-length block of instructions, data, or both, that can be transferred between central storage and external page storage. (2) To transfer instructions, data, or both, between central storage and external page storage.

page data set (PDS). One or more extents of disk storage in which pages are stored when they are not needed in processor storage.

page fault. In z/VSE or System/390 virtual storage systems, a program interruption that occurs when a page that is marked *not in central storage* is referred to by an active page.

page frame. An area of processor storage that can contain a page.

paging. In z/VSE, the process of transferring pages between central storage and external page storage.

panel. The complete set of information shown in a single display on a terminal screen. Scrolling back and forth through panels is like turning manual pages. See also *selection panel* and *data entry panel*.

parameter. Data item that is received by a routine.

partitionable CPC. A CPC that can be divided into two independent CPCs. See also *physical partition*, *single-image mode*, *side*.

partition. A division of the virtual address area available for running programs. See also *dynamic partition* and *static partition*.

partitioning. The process of forming multiple configurations from one configuration.

password. A unique string of characters known to a computer system and to a user, who must specify the character string to gain access to a system and to the information stored within it.

patchport. A pair of fibre adapters or couplers. Any number of patchports can participate in a fiber link. To determine the total number of patchports in a cabinet, you must add the number of patchports of each defined panel of the cabinet.

PC. Personal computer.

PCHID. Physical channel identifier.

PE. See *program error PTF*.

percolate. The action taken by the condition manager when the returned value from a condition handler indicates that the handler could not handle the condition, and the condition will be transferred to the next handler.

performance administration. The process of defining and adjusting workload management goals and resource groups based on installation business objectives.

PFK capability. On a display console, indicates that program function keys are supported and were specified at system generation.

physical channel identifier (PCHID). The physical address of a channel path in the hardware. Logical CHPIDs have corresponding physical channels. Real I/O hardware is attached to a processor through physical channels. Channels have a physical channel identifier (PCHID) that determines the physical location of a channel in the processor. The PCHID is a three hexadecimal digit number and is assigned by the processor.

physical partition. Part of a CPC that operates as a CPC in its own right, with its own copy of the operating system.

physical unit (PU). (1) The control unit or cluster controller of an SNA terminal. (2) The part of the control unit or cluster controller that fulfills the role of a physical unit as defined by systems network architecture (SNA).

physically partitioned (PP) mode. The state of a processor complex when its hardware units are divided into two separate operating configurations or sides. The A-side of the processor controller controls side 0. The B-side of the processor controller controls side 1. Contrast with *single-image (SI) configuration*.

PL/I. A general-purpose scientific/business high-level language. PL/I is a powerful procedure-oriented language especially well suited for solving complex scientific problems or running lengthy and complicated business transactions and record-keeping applications.

platform. The operating system environment in which a program runs.

pointer. An address or other indication of location.

portability. The ability to transfer an application from one platform to another with relatively few changes to the source code.

preprocessor. A routine that examines application source code for preprocessor statements that are then executed, resulting in the alteration of the source.

preventive service. The mass installation of PTFs to avoid rediscoveries of the APARs fixed by those PTFs.

primary key. One or more characters within a data record used to identify the data record or control its use. A primary key must be unique.

procedure. A set of self-contained high-level language (HLL) statements that performs a particular task and returns to the caller. Individual languages have different names for this concept of a procedure. In C, a procedure is called a function. In COBOL, a procedure is a paragraph or section that can only be performed from within the program. In PL/I, a procedure is a named block of code that can be invoked externally, usually through a a call.

processor. The physical processor, or machine, has a serial number, a set of channels, and a logical processor associated with it. The logical processor has a number of channel path IDs, or CHPIDs, which are the logical equivalent of channels. The logical processor may be divided into a number of logical partitions.

processor storage. See *central storage*.

program fetch. A program that prepares programs for execution by loading them at specific storage locations and readjusting each relocatable address constant.

program library. A VSE sublibrary that always contains named members.

program management. The task of preparing programs for execution, storing the programs, load modules, or program objects in program libraries, and executing them on the operating system.

processor controller. Hardware that provides support and diagnostic functions for the central processors.

Processor Resource/Systems Manager (PR/SM). The feature that allows the processor to use several z/VSE images simultaneously and provides logical partitioning capability. See also *LPAR*.

profile. Data that describes the significant characteristics of a user, a group of users, or one or more computer resources.

program error PTF (PE-PTF). A PTF that has been found to contain an error.

program function key (PFK). A key on the keyboard of a display device that passes a signal to a program to call for a particular program operation.

program interruption. The interruption of the execution of a program due to some event such as an operation exception, an exponent-overflow exception, or an addressing exception.

program level. The modification level, release, version, and fix level.

program management. The functions within the system that provide for establishing the necessary activation and invocation for a program to run in the applicable run-time environment when it is called.

program mask. In bits 20 through 23 of the program status word (PSW), a 4-bit structure that controls whether each of the fixed-point overflow, decimal overflow, exponent-overflow, and significance exceptions should cause a program interruption. The bits of the program mask can be manipulated to enable or disable the occurrence of a program interruption.

program number. The seven-digit code (in the format xxxx-xxx) used by IBM to identify each licensed program.

program status word (PSW). A 64-bit structure in central storage used to control the order in which instructions are executed, and to hold and indicate the status of the computing system in relation to a particular program. See also *program mask*.

program temporary fix (PTF). A temporary solution or bypass of a problem diagnosed by IBM as resulting from a defect in a current unaltered release of the program.

PSP. Preventive service planning.

PSW. Program status word.

PTF. Program temporary fix.

Q

QSAM. Queued sequential access method.

queue. A line or list formed by items in a system waiting for processing.

queue file. A direct access file maintained by VSE/POWER that holds control information for the spooling of job input and job output.

queued sequential access method (QSAM). An extended version of the basic sequential access method (BSAM). Input data blocks awaiting processing or output data blocks awaiting transfer to auxiliary storage are queued on the system to minimize delays in I/O operations.

R

RACF. Resource Access Control Facility.

read access. Permission to read information.

reader. A program that reads jobs from an input device or database file and places them on the job queue.

real address. In virtual storage systems, the address of a location in central storage.

real storage. See *central storage*.

reason code. A return code that describes the reason for the failure or partial success of an attempted operation.

record. (1) A group of related data, words, or fields treated as a unit, such as one name, address, and telephone number. (2) A self-contained collection of information about a single object. A record is made up of a number of distinct items, called fields. A number of shell programs (for example, awk, join, and sort) are designed to process data consisting of records separated by newlines, where each record contains a number of fields separated by spaces or some other character. awk can also handle records separated by characters other than newlines. See *fixed-length record* and *variable-length record*.

record data. Data sets with a record-oriented structure that are accessed record by record. This data set structure is typical of data sets on z/VSE and other mainframe operating systems. See also *byte stream*.

recording format. For a tape volume, the format of the data on the tape, for example, 18, 36, 128, or 256 tracks.

recovery. The process of rebuilding data after it has been damaged or destroyed, often by restoring a backup version of the data or by reapplying transactions recorded in a log.

recovery system. A system that is used in place of a primary application system that is no longer available for use. Data from the application system must be available for use on the recovery system. This is usually accomplished through backup and recovery techniques, or through various DASD copying techniques, such as remote copy.

recursive routine. A routine that can call itself or be called by another routine that it has called.

redundant array of independent disk (RAID). A disk subsystem architecture that combines two or more physical disk storage devices into a single logical device to achieve data redundancy.

reenterable. The reusability attribute that allows a program to be used concurrently by more than one task. A reenterable module can modify its own data or other shared resources, if appropriate serialization is in place to prevent interference between using tasks. See *reusability* and *reentrant*.

reentrant. The attribute of a routine or application that allows more than one user to share a single copy of a load module.

refreshable. The reusability attribute that allows a program to be replaced (refreshed) with a new copy without affecting its operation. A refreshable module cannot be modified by itself or any other module during execution. See *reusability*.

register. An internal computer component capable of storing a specified amount of data and accepting or transferring this data rapidly.

register save area (RSA). Area of main storage in which contents of registers are saved.

related installation materials (RIMs). In IBM custom-built offerings, task-oriented documentation, jobs, sample exit routines, procedures, parameters, and examples developed by IBM.

release. A distribution of a new product or new function and APAR fixes for an existing product. Contrast with *modification level* and *version*.

remote job entry (RJE). Submission of job control statements and data from a remote terminal, causing the jobs described to be scheduled and executed as though encountered in the input stream.

remote operations. Operation of remote sites from a host system.

reserved storage allocation. The amount of central and expanded storage that you can dynamically configure online or offline to a logical partition.

residency mode (RMODE). The attribute of a program module that specifies whether the module, when loaded, must reside below the 16 MB virtual storage line or may reside anywhere in virtual storage.

Resource Access Control Facility (RACF). An IBM security manager product that provides for access control by identifying and verifying the users to the system, authorizing access to protected resources, logging the detected unauthorized attempts to enter the system, and logging the detected accesses to protected resources.

restore. To write back onto disk data that was previously written from disk onto an intermediate storage medium such as tape.

restructured extended executor (REXX). A general-purpose, procedural language for end-user personal programming, designed for ease by both casual general users and computer professionals. It is also useful for application macros. REXX includes the capability of issuing commands to the underlying operating system from these macros and procedures.

return code. A code produced by a routine to indicate its success or failure. It may be used to influence the execution of succeeding instructions or programs.

reusability. The attribute of a module or section that indicates the extent to which it can be reused or shared by multiple tasks within the address space. See *refreshable*, *reenterable*, and *serially reusable*.

RIM. Related installation material.

RJE. Remote job entry.

RMODE. Residency mode.

rollback. The process of restoring data changed by an application to the state at its last commit point.

routine. (1) A program or sequence of instructions called by a program. Typically, a routine has a general purpose and is frequently used. CICS and programming languages use routines. (2) A database object that encapsulates procedural logic and SQL statements is stored on the database server, and can be invoked from an SQL statement or by using the CALL statement. The three main classes of routines are procedures, functions, and methods. (3) In REXX, a series of instructions called with the CALL instruction or as a function. A routine can be either internal or external to a user's program. (4) A set of statements in a program that causes the system to perform an operation or a series of related operations.

routing. The assignment of the communications path by which a message will reach its destination.

routing code. A code assigned to an operator message and used to route the message to the proper console.

RSA. Register save area.

run. To cause a program, utility, or other machine function to be performed.

run time. Any instant at which a program is being executed. Synonymous with *execution time*.

run-time environment. A set of resources that are used to support the execution of a program. Synonymous with *execution environment*.

S

SAF. System authorization facility.

SAP. System Assistance Processor.

save area. Area of main storage in which contents of registers are saved.

search chain. The order in which chained sublibraries are searched for the retrieval of a certain library member of a specified type.

security administrator. A programmer who manages, protects, and controls access to sensitive information.

selection panel. A displayed list of items from which a user can make a selection. Synonymous with menu.

sequential access. The serial retrieval of records in their entry sequence or serial storage of records with or without a premeditated order. Contrast with *direct access.*

sequential data set. (1) A data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape. Contrast with direct data set. (2) A data set in which the contents are arranged in successive physical order and are stored as an entity.

serially reusable. The reusability attribute that allows a program to be executed by more than one task in sequence. A serially reusable module cannot be entered by a new task until the previous task has exited. See *reusability*.

server. (1) On a network, the computer that contains programs, data, or provides the facilities that other computers on the network can access. (2) The party that receives remote procedure calls. Contrast with *client*.

service. PTFs and APAR fixes.

service level agreement (SLA). A written agreement of the information systems (IS) service to be provided to the users of a computing installation.

service processor. The part of a processor complex that provides for the maintenance of the complex.

session. (1) The period of time during which a user of a terminal can communicate with an interactive system. Usually, the elapsed time from when a terminal is logged on to the system until it is logged off the system. (2) The period of time during which programs or devices can communicate with each other. (3) In VTAM, the period of time during which a node is connected to an application program.

shared area. An area of storage that is common to all address spaces in the system. z/VSE has two shared areas:
1. The shared area (24 bit) is allocated at the start of the address space and contains the supervisor, the SVA (for system programs and the system GETVIS area), and the shared partitions.
2. The shared area (31 bit) is allocated at the end of the address space and contains the SVA (31 bit) for system programs and the system GETVIS area.

shared virtual area (SVA). A high address area that contains a system directory list (SDL) of frequently used phases, resident programs that can be shared between partitions, and an area for system support.

side. One of the configurations formed by physical partitioning.

SIGP. Signal processor.

Simple Object Access Protocol (SOAP). A protocol for exchanging XML-based messages over computer network.

simultaneous peripheral operations online (spool). The reading and writing of input and output streams on auxiliary storage devices, concurrently while a job is running, in a format convenient for later processing or output operations.

single-image (SI) mode. A mode of operation for a multiprocessor (MP) system that allows it to function as one CPC. By definition, a uniprocessor (UP) operates in single-image mode. Contrast with *physically partitioned (PP) configuration*.

single-processor complex. A processing environment in which only one processor (computer) accesses the spool and comprises the entire node.

single system image. The characteristic a product displays when multiple images of the product can be viewed and managed as one image.

skeleton. A set of control statements, instructions, or both, that requires user-specific information to be inserted before it can be submitted for processing.

SLA. Service level agreement.

small computer system interface (SCSI). A standard hardware interface that enables a variety of peripheral devices to communicate with one another.

SMF. System management facilities.

SNA. Systems Network Architecture.

SOAP. Simple Object Access Protocol.

software. (1) All or part of the programs, procedures, rules, and associated documentation of a data processing system. (2) A set of programs, procedures, and, possibly, associated documentation concerned with the operation of a data processing system. For example, compilers, library routines, manuals, circuit diagrams. Contrast with *hardware*.

sort/merge program. A processing program that can be used to sort or merge records in a prescribed sequence.

source code. The input to a compiler or assembler, written in a source language.

source program. A set of instructions written in a programming language that must be translated to machine language before the program can be run.

spool. Simultaneous peripheral operations online.

spooled data set. A data set written on an auxiliary storage device and managed by VSE/POWER.

spooling. The reading and writing of input and output streams on auxiliary storage devices, concurrently with job execution, in a format convenient for later processing or output operations.

standard label. A fixed-format record that identifies a volume of data such as a tape reel or a file that is part of a volume of data.

startup. The process of performing IPL of the operating system and of getting all subsystems and application programs ready for operation.

static partition. A partition defined at IPL time and occupying a defined amount of virtual storage that remains constant. Contrast with *dynamic partition*.

status-display console. An MCS console that can receive displays of system status but from which an operator cannot enter commands.

string. A collection of one or more I/O devices. The term usually refers to a physical string of units, but may mean a collection of I/O devices that are integrated into a control unit.

subarea. A portion of the SNA network consisting of a subarea node, attached peripheral nodes, and associated resources. Within a subarea node, all links and adjacent link stations in attached peripheral or subarea nodes that are addressable within the subarea share a common subarea address and have distinct element addresses.

sublibrary. A subdivision of a z/VSE library. Members can only be accessed in a sublibrary.

sublibrary directory. An index for the system to locate a member in the accessed sublibrary.

submit. A VSE/POWER function that passes a job to the system for processing.

subpool storage. All of the storage blocks allocated under a subpool number for a particular task.

subsystem. A secondary or subordinate system, or programming support, usually capable of operating independently of or asynchronously with a controlling system. Example is CICS.

subtask. In the context of z/VSE multitasking, a task that is initiated and terminated by a higher order task (the main task). Subtasks run the parallel functions, those portions of the program that can run independently of the main task program and each other.

supervisor. The part of z/VSE that coordinates the use of resources and maintains the flow of processing unit operations.

supervisor call (SVC). An instruction that interrupts a program being executed and passes control to the supervisor so that it can perform a specific service indicated by the instruction.

support element. A hardware unit that provides communications, monitoring, and diagnostic functions to a central processor complex (CPC).

SVC. supervisor call instruction.

SVC interruption. An interruption caused by the execution of a supervisor call instruction, causing control to be passed to the supervisor.

SVC routine. A control program routine that performs or begins a control program service specified by a supervisor call instruction.

SWA. scheduler work area.

switch. A device that provides connectivity capability and control for attaching any two ESCON or FICON links together.

syntax. The rules governing the structure of a programming language and the construction of a statement in a programming language.

SYSRES. For z/OS: system residence disk. For z/VSE: system residence file.

system. The combination of a configuration (hardware) and the operating system (software). Often referred to simply as the z/VSE system.

system console. In z/VSE, a console attached to the processor controller used to initialize a z/VSE system.

system control element (SCE). Hardware that handles the transfer of data and control information associated with storage requests between the elements of the processor.

system directory list (SDL). A list containing directory entries of frequently-used phases and of all phases resident in the SVA. The list resides in the SVA.

system file. A file used by the operating system, for example, the hardcopy file, the recorder file, the page data set.

system management facilities (SMF). A z/OS component that provides the means for gathering and recording information for evaluating system usage.

system recorder file. The file used to record hardware reliability data. Synonymous with recorder file.

system residence file (SYSRES). The z/VSE system sublibrary IJSYSRS.SYSLIB that contains the operating system. It is stored on the system residence volume DOSRES.

Systems Network Architecture (SNA). A description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of networks.

T

tape volume. Storage space on tape, identified by a volume label, which contains data sets or objects and available free space. A tape volume is the recording space on a single tape cartridge or reel. See also *volume*.

task. In a multiprogramming or multiprocessing environment, one or more sequences of instructions treated by a control program as an element of work to be accomplished by a computer.

task control block (TCB). A data structure that contains information and pointers associated with the task in process.

TCB. task control block.

TCP/IP. Transmission Control Protocol/Internet Protocol.

temporary data set. A data set that is created and deleted in the same job.

terminal. A device, usually equipped with a keyboard and some kind of display, capable of sending and receiving information over a link.

terminal owning region (TOR). A CICS region devoted to managing the terminal network.

timeout. The time in seconds that the storage control remains in a “long busy” condition before physical sessions are ended.

transaction. A unit of work performed by one or more transaction programs, involving a specific set of input data and initiating a specific process or job.

Transmission Control Protocol/Internet Protocol (TCP/IP). A hardware independent communication protocol used between physically separated computers. It was designed to facilitate communication between computers located on different physical networks.

Transport Layer Security (TLS). A protocol that provides communications privacy over the Internet.

trunk cable. Cables used to make permanent connections between cabinets and which remain in place even when not in use.

U

Unicode Standard. A universal character encoding standard that supports the interchange, processing, and display of text that is written in any of the languages of the modern world. It can also support many classical and historical texts and is continually being expanded.

uniprocessor (UP). A processor complex that has one central processor.

unused cable. Physical cables that have been recently disconnected, but not yet placed in inventory.

upwardly compatible. The ability for applications to continue to run on later releases of z/OS, without the need to recompile or relink.

user abend. A request made by user code to the operating system to abnormally terminate a routine. Contrast with *system abend*.

user catalog. An optional catalog used in the same way as the master catalog and pointed to by the master catalog. It also lessens the contention for the master catalog and facilitates volume portability.

user exit. A routine that takes control at a specific point in an application. User exits are often used to provide additional initialization and termination functions.

user ID. user identification.

user identification (user ID). A 1-8 character symbol identifying a system user.

V

variable-length record. A record having a length independent of the length of other records with which it is logically or physically associated. Contrast with *fixed-length record*.

vendor. A person or company that provides a service or product to another person or company.

version. A separate licensed program that is based on an existing licensed program and that usually has significant new code or new functions. Contrast with *release* and *modification level*.

VIO. virtual input/output.

virtual address space. In virtual storage systems, the virtual storage assigned to a job, terminal user, or system task. See also *address space*.

virtual input/output (VIO). The allocation of data sets that exist in paging storage only.

Virtual Storage Access Method (VSAM). An access method for direct or sequential processing of fixed-length and varying-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by relative-record number.

virtual storage. (1) The storage space that can be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of auxiliary storage available, not by the actual number of main storage locations. (2) An addressing scheme that allows external disk storage to appear as main storage.

virtual telecommunications access method (VTAM). A set of programs that maintain control of the communication between terminals and application programs running under z/VSE.

VisualAge Generator EGL plug-in for VSE. A successor product for VisualAge Generator. The purpose of the plug-in is to generate COBOL applications that can be executed with VisualAge Generator Server for VSE (the “Server”). Currently the plug-in executes with Rational Application Developer (RAD) and Rational Web Developer (RWD).

VM. Virtual Machine.

volume. (1) The storage space on DASD, tape or optical devices, which is identified by a volume label. (2) That portion of a single unit of storage which is accessible to a single read/write mechanism, for example, a drum, a disk pack, or part of a disk storage module. (3) A recording medium that is mounted and demounted as a unit, for example, a reel of magnetic tape or a disk pack.

volume backup. Backup of an entire volume to protect against the loss of the volume.

volume serial number. A number in a volume label that is assigned when a volume is prepared for use in the system.

volume table of contents (VTOC). A table on a direct access storage device (DASD) volume that describes the location, size, and other characteristics of each data set on the volume.

VPN. virtual private network.

VSAM. virtual storage access method.

VSE (Virtual Storage Extended). A system that consists of a basic operating system and any IBM supplied and user-written programs required to meet the data processing needs of a user. VSE and the hardware it controls form a complete computing system. Its current version is called z/VSE.

VSE/ICCF (VSE/Interactive Computing and Control Facility). An IBM program that serves as interface, on a time-slice basis, to authorized users of terminals linked to the system’s processor.

VSE/ICCF library. A file composed of smaller files (libraries) including system and user data which can be accessed under the control of VSE/ICCF.

VSE library. A collection of programs in various forms and storage dumps stored on disk. The form of a program is indicated by its member type such as source code, object module, phase, or procedure. A VSE library consists of at least one sublibrary which can contain any type of member.

VSE/POWER. An IBM program primarily used to spool input and output. The program's networking functions enable a VSE system to exchange files with or run jobs on another remote processor.

VSE/VSAM. VSE/Virtual Storage Access Method.

VSE/VSAM catalog. A file containing extensive file and volume information that VSE/VSAM requires to locate files, to allocate and deallocate storage space, to verify the authorization of a program or an operator to gain access to a file, and to accumulate use statistics for files.

VTAM. Virtual Telecommunications Access Method.

VTOC. volume table of contents.

W

WAP. wireless access point.

wait state. Synonymous with *waiting time*.

waiting time. (1) The condition of a task that depends on one or more events in order to enter the ready condition. (2) The condition of a processing unit when all operations are suspended.

wild carding. The use of an asterisk (*) as a multiple character replacement in classification rules.

workload. A group of work to be tracked, managed and reported as a unit.

wrap mode. The console display mode that allows a separator line between old and new messages to move down a full screen as new messages are added. When the screen is filled and a new message is added, the separator line overlays the oldest message and the newest message appears immediately before the line.

write-to-operator (WTO) message. A message sent to an operator console informing the operator of errors and system conditions that may need correcting.

write-to-operator-with-reply (WTOR) message. A message sent to an operator console informing the operator of errors and system conditions that may need correcting. The operator must enter a response.

WTO. write-to-operator.

WTOR. write-to-operator-with-reply.

X

XA. Extended Architecture.

Z

zAAP. System z Application Assist Processor.

z/Architecture. An IBM architecture for mainframe computers and peripherals. The System z family of servers uses the z/Architecture.

zIIP. System z Integrated Information Processor.

z/OS. A widely used operating system for IBM mainframe computers that uses 64-bit central storage.

zAAP. System z Application Assist Processor. A specialized processing assist unit configured for running Java programming on selected System z machines.

z/VM. z/Virtual Machine.

z/VSE (z/Virtual Storage Extended). The most advanced VSE system currently available.

z/VSE Language Environment. An IBM software product that provides a common run-time environment and common run-time services for conforming high-level language compilers.

Numerics

3270 pass-through mode. A mode that lets a program running from the z/OS shell send and receive a 3270 data stream or issue TSO/E commands.

Index

Introduction to the New Mainframe: z/VSE Basics

Introduction to the New Mainframe: z/VSE Basics

Basic mainframe concepts and architecture

z/VSE fundamentals for students and beginners

Mainframe hardware and peripheral devices

This IBM Redbooks publication provides students of information systems technology with the background knowledge and skills necessary to begin using the basic facilities of a mainframe computer. It is the first in a planned series of textbooks designed to introduce students to mainframe concepts and help prepare them for a career in large systems computing.

For optimal learning, students are assumed to have successfully completed an introductory course in computer system concepts, such as computer organization and architecture, operating systems, data management, or data communications. They should also have successfully completed courses in one or more programming languages, and be PC literate.

This textbook can also be used as a prerequisite for courses in advanced topics or for internships and special studies. It is not intended to be a complete text covering all aspects of mainframe operation, nor is it a reference book that discusses every feature and option of the mainframe facilities.

Others who will benefit from this course include experienced data processing professionals who have worked with non-mainframe platforms, or who are familiar with some aspects of the mainframe but want to become knowledgeable with other facilities and benefits of the mainframe environment.

SG24-7436-00 ISBN 0738486248