SC33-0068-2

Customer Information Control System/Virtual Storage (CICS/VS) Version 1 Release 5

System/Application Design Guide

Program Product

Program Numbers 5740-XX1 (CICS/OS/VS)
5746-XX3 (CICS/DOS/VS)

IBM

| Third Edition (May 1980)

| This edition applies to Version 1 Release 5 (Version 1.5) of the IBM program product Customer Information Control System/Virtual Storage (CICS/VS), program numbers 5746-XX3 (for DOS/VS) and 5740-XX1 (for OS/VS). Until the OS/VS version is released, the information applicable to that version is for planning purposes only.

| This edition is based on the CICS/VS Version 1.4.1 edition, and changes | from that edition are indicated by vertical lines to the left of the | changes. Note, however, that the 1.4.1 edition remains current and | applicable for users of Version 1.4.1 of CICS/VS.

Information in this publication is subject to change. Changes will be published in new editions or technical newsletters. Before using this | publication, consult the latest IBM System/370 and 4300 Processors | Bibliography, GC20-0001, to learn which editions and technical newsletters are current and applicable.

Publications are not stocked at the addresses given below; requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication; if the form has been removed, comments may be addressed either to:

| International Business Machines Corporation,
| Department 812HP,
| 1133 Westchester Avenue
| White Plains, New York 10604.

or to:

IBM United Kingdom Laboratories Limited,
Programming Publications, Mail Point 095,
Hursley Park,
Winchester, Hampshire SO21 2JN, England.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Preface

This publication provides the system analyst and system administrator with guidelines which assist in the design of online applications to run under the control of CICS/DOS/VS or CICS/OS/VS. It assumes that the reader is familiar with the introductory information contained in the CICS/VS General Information manual.

The publication is directed mainly towards the inexperienced CICS/VS user, and assumes no prior CICS/VS knowledge apart from that presented in the CICS/VS General Information manual.

The generation of CICS/VS and the preparation of the system tables that describe the environment to be supported by CICS/VS are described in the appropriate (CICS/DOS/VS or CICS/OS/VS) CICS/VS System Programmer's Guide and the CICS/VS System Programmer's Reference Manual. The relationships between these publications, an overview of their contents, and suggestions regarding their use are given in Chapter 1 of this publication.

In this publication, the term VTAM refers to ACF/VTAM, to ACF/VTAME (CICS/DOS/VS only), and to the Record Interface of ACF/TCAM (CICS/OS/VS only). The term TCAM refers both to TCAM and to the DCB Interface of ACF/TCAM. The term BTAM refers to BTAM (CICS/OS/VS only) and to BTAM-ES (CICS/DOS/VS only). For further details of system requirements, refer to the publication CICS/VS General Information.

RELATED PUBLICATIONS

Telecommunications Access Method (TCAM):

- OS/VS TCAM Concepts and Applications, GC30-2049
- OS/VS TCAM System Programmer's Guide, TCAM Level 10, GC30-2051
- OS/VS TCAM Application Programmer's Guide, TCAM Level 10, GC30-3036
- OS/VS TCAM Macro Reference Guide, TCAM Level 10, GC30-2052

Virtual Telecommunications Access Method (VTAM):

- Introduction to VTAM, GC27-6987

Virtual Storage Access Method (VSAM):

- VSE/VSAM General Information, GC24-5743
- OS/VS VSAM Planning Guide, GC26-3799

Customer Information Control System/Virtual Storage (CICS/VS), Version 1 Release 5:

- General Information, GC33-0066
- Application Programmer's Reference Manual (Command Level), SC33-0077
- Application Programmer's Reference Manual (Macro Level), SC33-0079

Application Programmer's Reference Manual (RPG II), SC33-0085

System Programmer's Reference Manual, SC33-0069

Operator's Guide, SC33-0080

System Programmer's Guide (DOS/VS), SC33-0070

System Programmer's Guide (OS/VS), SC33-0071

Entry Level System User's Guide (DOS/VS), SC33-0086

Master Terminal Operator's Reference Summary, SX33-6011

Problem Determination Guide, SC33-0089

Program Debugging Reference Summary, SX33-6010

Messages and Codes, SC33-0081

Diagnosis Reference, LC33-0105

Data Areas (OS), LY33-6035

Data Areas (DOS/VS), LY33-6033

IBM 3270 Guide, SC33-0096

IBM 3600/3630 Guide, SC33-0072

IBM 3650/3680 Guide, SC33-0073

IBM 3767/3770/6670 Guide, SC33-0074

IBM 3790/3730 Guide, SC33-0075

Data Language/I (DL/I) DOS/VS:

System/Application Design Guide, SH12-5413

General Information, GH20-1246

Application Programming Reference Manual, SH12-5411

Utilities and Guide for the System Programmer, SH12-5412

High Level Programming Interface User's Guide, SH24-5009

Information Management System/Virtual Storage (IMS/VS):

System/Application Design Guide, SH20-9025

General Information, GH20-1260

System Programming Reference Manual, SH20-9027

Application Programming Reference Manual, SH20-9026

Utilities Reference Manual, SH20-9029

Display Management System/VS (DMS/VS):

General Information, GH20-1863

System Network Architecture:

Types of Logical Unit to Logical Unit Sessions, GC20-1869

Functional Description of Logical Unit Types, GC20-1868

VIDEO/370:

General Information, SC27-6960

Data Security Design Handbook, GB0F-7502

CICS Productivity Aids:

Installed User Programs

CICS Online Test/Debug II
Program Description Operations Manual, SH20-1877

Field Developed Programs

CICS Dynamic Map Program Description
Operations Manual, SB21-1075

CICS/VS Performance Analyzer II
Program Description Operations Manual, SB21-1697

CICS/3270 Simulator Program Description
Operations Manual, SB21-1036

CICS Plot
Program Description Operations Manual, SB21-1508

CICS Network Activity Simulator
Program Description Operations Manual, SB21-1505

CICS Source Program Maintenance Online II
Program Description Operations Manual, SB21-1700

Contents

Figures

Summary of Amendments for Version 1 Release 5

The new facilities and enhancements available with CICS/VS Version 1 Release 5 are:

- New intercommunication facilities, offering:
  - Multiregion operation (MRO) — a new mechanism that allows communication between multiple connected CICS/VS regions within the same processing system without the use of SNA networking facilities.
  - Distributed transaction processing (DTP) — direct transaction-to-transaction communication across systems. (This facility is not available on MRO.)
  - Intersystem Communication between CICS/VS and IMS/VS.
  - Improved throughput by support of SNA parallel sessions.
- Enhanced master terminal facilities for interactive control of CICS/VS
- Command-level interface enhancements:
  - an interactive command interpreter.
  - a new command-level interface with DL/I DOS/VS.
  - Execution Diagnostic Facility enhancements to support DL/I commands (CICS/DOS/VS only).
- Security enhancements, including support for an external security manager (for example, the Resource Access Control Facility (RACF) program product).
- Improved monitoring facilities
- Further device support, including:
  - additional 3270 support.
  - use of the OS/VS console as a CICS/VS terminal.
  - networking of TWX and WTTY terminals through the Network Terminal Option (NTO) program product.
- Usability and serviceability aids, including a new user exit mechanism and facilities in CICS/DOS/VS similar to those provided by the FERS service aid.

This manual has been reorganized since the previous release of CICS/VS. It is now divided into seven parts:

Part 1, Introduction
Part 2, Data Base Design
Part 3, Data Communication Design
Part 4, Application Design
Part 5, Recovery and Restart Design
Part 6, Performance Design
Part 7, Intercommunication Design

Summary of Amendments for Version 1 Release 4.1

This Technical Newsletter contains changes to the latest edition (SC33-0068-1 plus TNL SN33-6216) that reflect new features of CICS/VS introduced in Version 1, Release 4, Modification 1. These features are:

- Support for logical units type 4 (LUTYPE4)
- Message performance option of intersystem communication
- Fixed block architecture (FBA) storage devices

All changes are indicated by revision bars in the left margin.

Summary of Amendments for Version 1 Release 4

This publication replaces, for CICS/VS Version 1.4, the previous edition SC33-0068-0. A new chapter, Chapter 13, "Intersystem Communication", has been added describing the facilities available for the interconnection of CICS/VS systems.

Changes are also included to cover the following new facilities introduced in CICS/VS Version 1.4:

- Data Base Support
  - Shared Data Base (CICS/OS/VS)
  - Transaction Restart
- Extension to support of 3270 Devices
- Enhancements to the Command Level Interface
  - Assembler
  - RPG II (CICS/DOS/VS)
- CICS/DOS/VS Entry Level System (ELS)
- Execution (Command Level) Diagnostic Facility (EDF)

These additions and all other changes to the text originally contained in SC33-0068-0 are indicated by revision bars in the left margin.

Part 1. Introduction

Chapter 1.1. Introduction to CICS/VS

CICS/VS

The IBM Customer Information Control System/Virtual Storage (CICS/VS) is a general purpose data base/data communication system. The term data base/data communication (DB/DC) is descriptive of the type of processing carried out by online systems, as opposed to batch-processing systems. Generally online systems involve the transmission of information from a terminal to a computer, the use of that information to access data maintained by the computer (referred to as a data base), and the transmission of processed information back to the terminal. Hence the term data base/data communication system.

As a DB/DC system, CICS/VS provides support for online systems in much the same way as the operating system and access methods provide support for batch processing systems. However, CICS/VS is not a replacement for an operating system. It runs under the control of Virtual Storage Extended (VSE) or the Operating System/Virtual Storage (OS/VS1 or OS/VS2) and uses standard access methods.

A DB/DC system can be seen as consisting of two major components, application and environment.

While the application component varies from user to user, the environment component serves all users by controlling those elements of the DB/DC system involved in communicating with terminals, accessing data base information, and controlling the passing of that information to the application component for processing.

The development of the environment component of a DB/DC system often requires more effort than the development of the application component. To relieve the user of the need to develop the environment component of his DB/DC system, and to enable him to concentrate on the application component, CICS/VS is designed as a modular system. This modular structure allows the user to select and tailor a CICS/VS system at system generation or initialization to meet particular application processing needs. While some of the available CICS/VS functions and their associated services are essential to the system, many are optional and may be included in the system if and when required.

CICS/VS Pregenerated Systems

CICS/DOS/VS ENTRY LEVEL SYSTEM

The CICS/DOS/VS Entry Level System is a subset of CICS/VS. It provides fewer functions than those provided by CICS/VS. By providing fewer functions it is more suitable for small systems or first time users. The functions provided by the Entry Level System are described in the CICS/VS ELS User's Guide. The Entry Level System is supplied as a completely pre-generated system and thus no sysgen process is required.

CICS/VS STARTER SYSTEM (DOS/VS)

This system consists of a private core-image library and private relocatable library. The private core-image library contains pregenerated versions of all CICS/DOS/VS programs together with sample tables, maps, and applications. The private relocatable library contains those items that were needed to generate the private core-image library contents.

CICS/VS STARTER SYSTEM (OS/VS1)

A load library is supplied containing pregenerated versions of every CICS/OS/VS program, together with sample tables, maps, and applications.

INSTALLATION AND USE

The user can install the appropriate system and expand as the application needs dictate. Full details of installation and expansion are given in the CICS/VS ELS User's Guide, or the CICS/VS System Programmer's Guide (DOS/VS and OS/VS versions) and CICS/VS System Programmer's Reference Manual.

CICS/VS System Publications

The publications available with CICS/VS provide an extensive library containing information on the various aspects of CICS/VS. The CICS/VS System/Application Design Guide is one of a group of four publications concerned with CICS/VS design and programming. The other three publications in this group are:

CICS/VS Entry Level System User's Guide (DOS/VS)
CICS/VS System Programmer's Guide (DOS/VS or OS/VS)
CICS/VS System Programmer's Reference Manual

This group of publications provides information necessary for the design, installation, generation, execution, and efficient online performance of CICS/VS. Figure 1.1-1 gives an overview of the information in these publications. The notes following apply to the references given in the diagram.

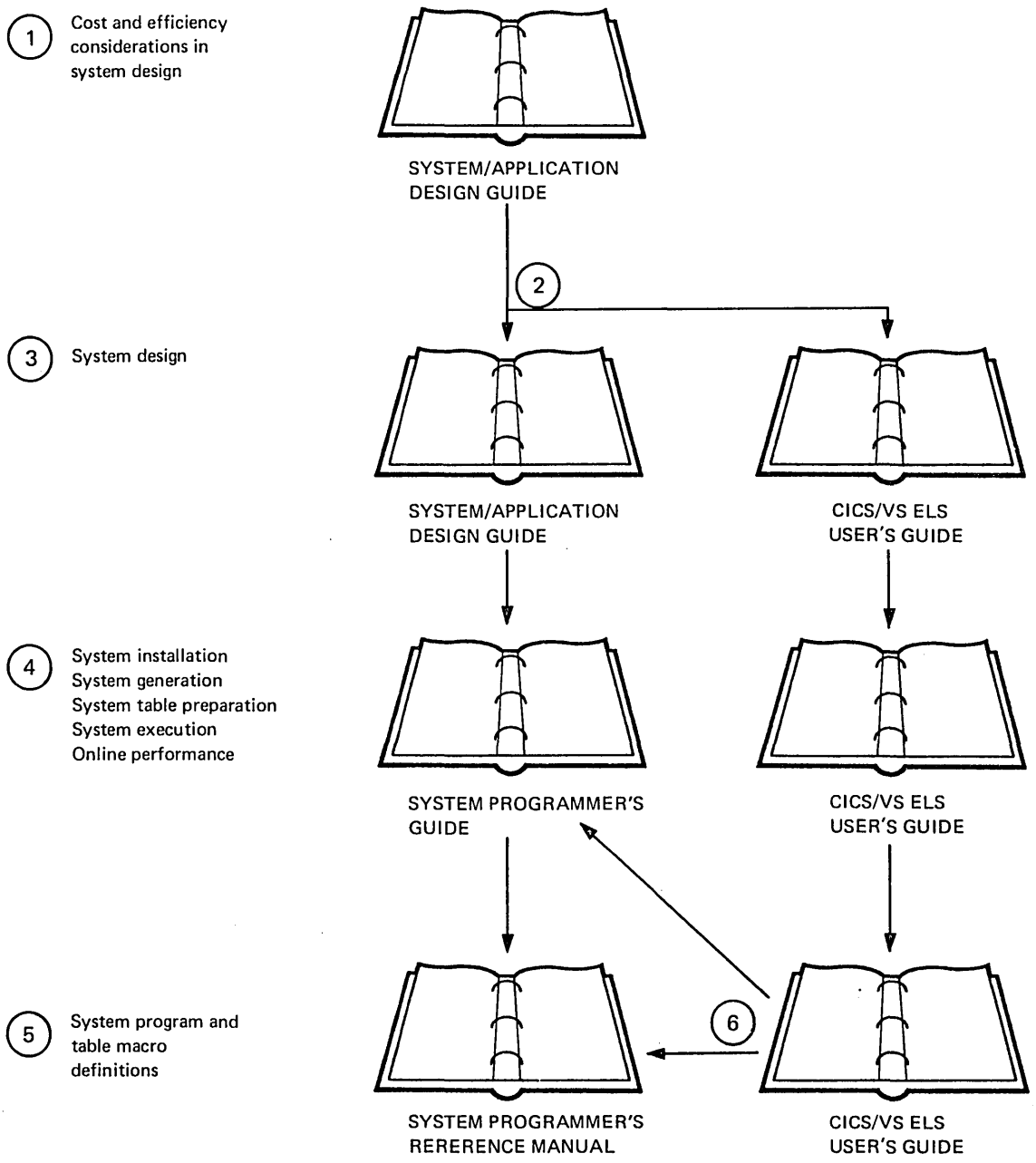


Figure 1.1-1. CICS/VS System Information Organization

Notes:

1. This publication provides information to enable a system design team to decide which CICS/VS functions and facilities would be best suited for a particular installation/application.

2. The CICS/DOS/VS Entry Level System is available to users who do not require the full range of CICS/DOS/VS functions. The limited set of functions available under the Entry Level System is described in the CICS/VS Entry Level System User's Guide, which contains information on the use of the CICS/DOS/VS Entry Level System.

3. Information relating to all aspects of system design (application programs, basic mapping support, data management and data base, is provided, as applicable, in this publication and the CICS/VS Entry Level System User's Guide.

4. Once the design of a particular CICS/VS system has been decided, the CICS/VS System Programmer's Guide, (or CICS/VS Entry Level System User's Guide if the CICS/DOS/VS Entry Level System is to be used), leads the system programmer through the various steps necessary to achieve and maintain efficient operation of an online system.

5. In order to generate the required CICS/VS programs and control tables, the system programmer must refer to either the CICS/VS System Programmer's Reference Manual (or the CICS/VS Entry Level System User's Guide) for detailed information about the macros that must be used.

Chapter 1.2. System Design

The installation of an online system involves a number of activities. These include, but are not limited to:

Development

- Feasibility study
- Network planning
- Equipment management
- Development of installation
- Education
- Development of standards
- Application design
- Documentation

Implementation

- CICS/VS generation
- Offline and online program writing and testing
- Preparation of computer and network operations
- System integration
- Documentation

Operations

Operation, maintenance, and evaluation of:

- CICS/VS
- Application programs
- Housekeeping routines
- Complete network

Continuous evaluation of all aspects of the installation could lead to modifications of the current system and expansion of the applications handled by the system.

The purpose of this publication is to discuss one main activity, namely, System Design of Online Applications, because of the effect of system design on the overall success or failure of the application. The publication presents the considerations involved in online system design in the same sequence as they might be encountered in an actual design situation. The factors to consider during each step of the design process are identified in terms of the application requirements. Some

design factors and application requirements are satisfied by CICS/VS-provided support. These facilities are explicitly defined and identified.

Many applications will not require additional user-developed support beyond that provided by CICS/VS. However, online applications may exhibit unique requirements. This publication presents these additional support requirements, outlines suggested design solutions, and discusses some of the potential problem areas that should be considered by the user.

To use CICS/VS facilities efficiently and satisfy design requirements, it is important that the system designer be aware of the manner in which CICS/VS implements these facilities. This information is presented at the conceptual level, assuming there is no prior knowledge beyond that covered in CICS/VS General Information. More detail can also be obtained, if necessary, by referring to other CICS/VS documentation.

The Need for Good System Design

The design of any system, whether it be a batch processing system or an online system, is a complex and involved procedure. A "cookbook" approach to system design cannot be followed because of the variety of ways the same application may be implemented in different organizations. However, guidelines can be recommended, which direct the designer to consider those functions or requirements that exist in the design of most online systems.

TURNAROUND OR RESPONSE TIME

The effect of poor system design in a batch processing environment increases the total processing time of applications, with consequent delays in turnaround time before results of that processing are available. With an infrequently run batch application, the effect of poor system design on the installation may not be great. However, with frequently run batch processing applications, poor system design and long run times may impact the ability of the installation to provide adequate turnaround for that and other applications. This will probably necessitate a change in the system design of the offending application.

In an online environment, the effect of poor system design is often immediately apparent, generally through the online system providing unacceptable response times for the particular applications concerned. The definition of an "acceptable response time" is generally very application-dependent. For example, in an online order entry application, where the terminal operator takes an order from a customer directly over the telephone, any response time that keeps that customer waiting can be regarded as unacceptable.

USER ACCEPTANCE

A factor that can affect the acceptability of an online application is the way in which it meets the needs of the users of that application. It is pointless for the user to design a system that provides fast response time if the information provided cannot be used. In this regard, measured by the usability of the system, an unusable system is therefore as inefficient as a "poor performance" system.

RESOURCE UTILIZATION

A final factor to consider is the utilization of resources such as the processor power, processor storage, and input/output devices. An online system that unnecessarily uses so much processor capability, or storage, or so many input/output devices that it impacts the ability of the installation to carry out other processing in other partitions or regions, may result in the complete installation becoming a "poor performance" system.

Thus, poor system design can have a significant impact on:

- Customer service (because of poor response time)
- Application usability
- Installation processing capability

Design Strategy

Generally, online systems cannot be designed in isolation. To ensure that the foregoing objectives are met, it is important that a design group comprise people with knowledge of:

- Application requirements
- CICS/VS facilities
- Installation requirements

Usually, the optimum size for the design group is three or four. Fewer than this number increases the probability that bad design decisions can slip through, while many more than four may affect the productivity of the design group as a whole.

The system design phase is an iterative process. Based on the decisions taken at one stage of the design, it may be necessary to change decisions that were made earlier in another area of the design. This change may in turn affect other decisions. Thus, the design group must be flexible in its approach and be prepared during the design phase to change its decisions if necessary. However, once the system design has been completed, it should be frozen at that point, and not changed unless serious errors or omissions are found, which will affect the ability of the system to run effectively.

During implementation of the design, there is always the temptation to incorporate improvements from an application point of view. While each improvement may not represent a great deal of extra implementation effort, all of these improvements may affect the project completion

date. Also, the effect of these improvements on the overall system performance must be evaluated. The danger is that this evaluation may not be carried out for those changes introduced after the system design phase has been completed.

These changes or enhancements must be controlled. The best way of achieving this control may be to incorporate all of these enhancements in a later version of the online application or system. These enhancements become a project in their own right, and must therefore go through the system design phase before implementation. In this way their effect on system performance can be readily evaluated.

A top-down approach to system design is possible, and such an approach should direct the design group to consider all of those areas of the online system which may require decisions to be taken. This top-down design approach is illustrated in Figure 1.2-1. This figure also illustrates some of the topics presented in this publication, and the description of each topic following the figure provides an overview of this publication.

APPLICATION DESIGN

The starting point for online system design is the application design. The initial application design steps require that the objectives to be achieved by an online application be defined and the requirements of the users of that application be identified. A broad system flow of the application is then developed as part of the initial design. This system flow and application design are an extremely important part of the overall design process, because they define the interface between the terminal user and the computer. Unless the online application meets the requirements of its users, it is destined to fail.

The online application should be designed initially to identify the broad input, processing, and output requirements of the application. The need for conversational and/or batch data transmission between the terminals and the processor can be identified. The terminal output requirements of the application can be determined, after which the broad processing logic and data set accessing necessary to produce that output can be designed. At this stage, the input data required for that processing and output can also be defined.

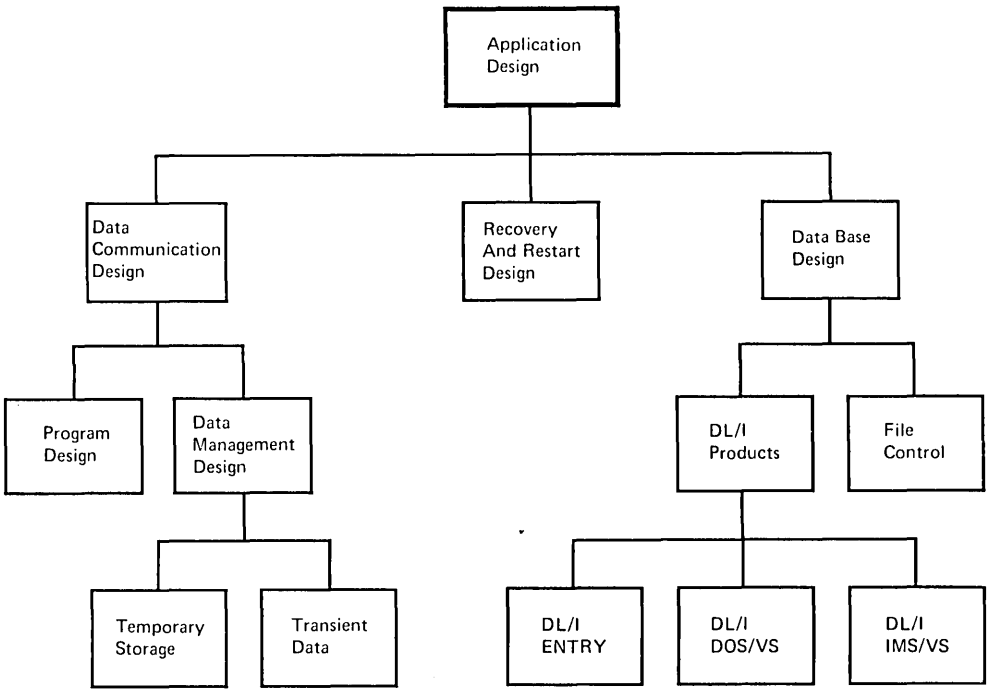


Figure 1.2-1. Top-down Systems Design

The result of this application design phase is a broad system flowchart showing, in application terms only, the flow of information to and from terminals, the broad processing to be carried out by the processor, and the file accessing necessary to allow that processing. Figures 1.2-2 and 1.2-3 illustrate two types of flowcharts, both representing the system flow of an Order Entry and Invoicing application in the Distribution industry.

DATA COMMUNICATION DESIGN

With the broad application design mapped out, design of transactions to be initiated from terminals and the responses to be sent back to the terminals can be developed. Also, during this phase the editing and validation of input messages can be defined in more detail.

Consideration should be given to the design of security procedures and the handling of high priority transactions. The effect of unrecoverable terminal and line errors should be considered, together with approaches which may be used to provide a communications backup capability (if required) to enable the online applications to continue to function, if possible, in the event of a communications equipment malfunction.

Data communication design is discussed in greater detail in Part 3 of this publication.

After determining system flow and broad processing to be carried out by the processor, this processing should now be broken down into particular functions. For example, the initial function on receiving a transaction identification code from a terminal would be that of editing or validation.

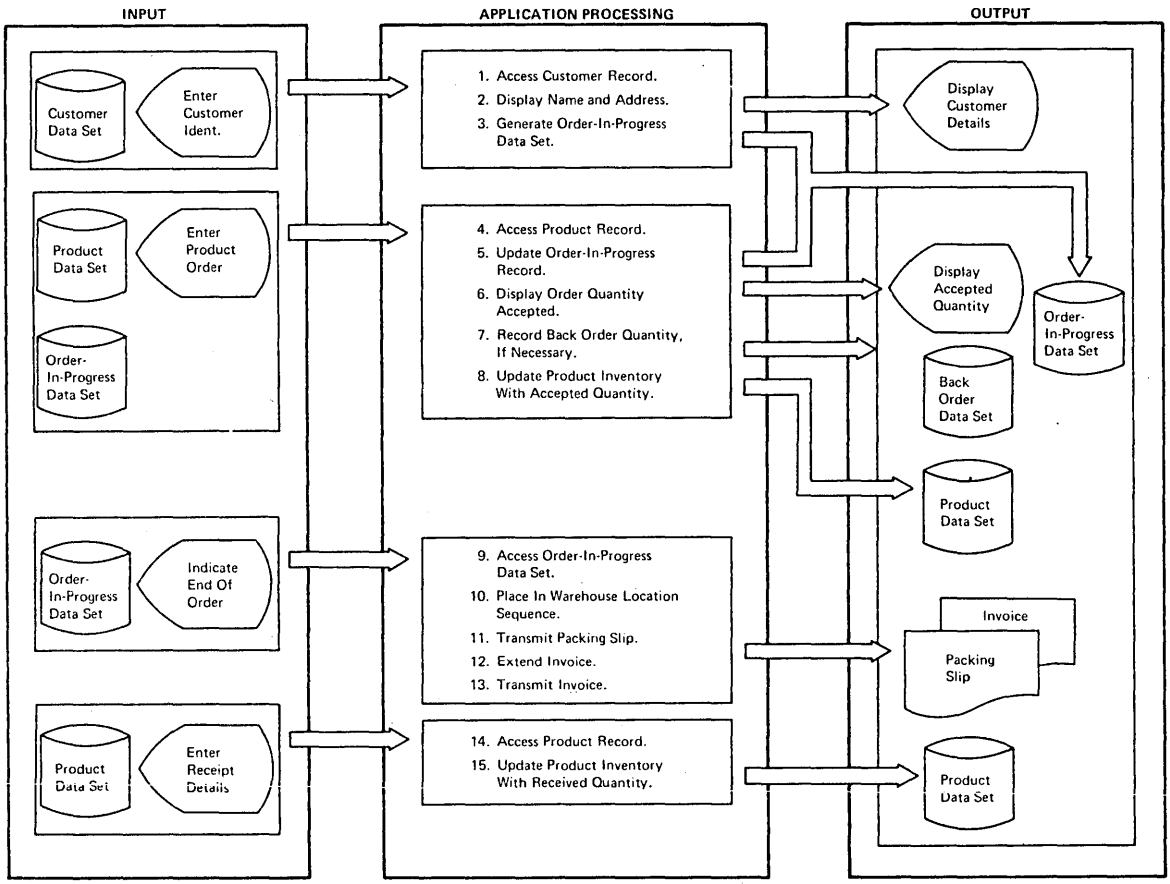


Figure 1.2-2. Order Entry and Invoicing Function Diagram

This validation may require access to various data sets. Following validation, it may be necessary to retrieve information from other data sets for processing, followed by possible updating of those data sets. Finally, it would be necessary to prepare a response to be sent to the terminal.

The processing for each type of transaction in the application should be broken down into logical sections in this manner. These logical sections may subsequently become separate CICS/VS application program modules, or can be incorporated into one module. Figure 1.2-4 illustrates the various modules in the program design for the Order Entry and Invoicing application shown in Figures 1.2-2 and 1.2-3.

Note that the separate programs and broad processing required, developed in Figure 1.2-4 from the flowchart in Figure 1.2-3, are described as part of the function diagram in Figure 1.2-2. In effect, the first three boxes in Figure 1.2-2 define the three separate programs in Figure 1.2-4.

A point to consider when defining program modules is the frequency of use of different modules. For example, exception routines or error routines that are infrequently used should be separated from the more frequently used main processing modules. In this way program design and subsequent implementation will be able to take best advantage of the dynamic storage capabilities of CICS/VS and the virtual storage capabilities of VSE, OS/VS1, or OS/VS2.

Application programs can be coded in assembler, COBOL, RPG II (CICS/DOS/VS only), or PL/I. The user can select the most appropriate language for each program. Programs written in one language can pass control to programs written in another language.

Application program design is discussed in greater detail in Part 4 of this manual.

DATA MANAGEMENT DESIGN

Application requirements for the temporary storage of information and the queuing of information should be defined. CICS/VS Temporary Storage management provides a "scratchpad" capability and allows information to be stored temporarily in main storage or, alternatively, on secondary storage.

The queuing, or sequential data set requirements, of the application can be defined. The need to pass information through sequential files to and from the CICS/VS partition and other batch partitions or regions using the CICS/VS Transient Data management facility can also be determined, together with broad recovery procedures.

It is also necessary to determine whether the application programs are to pass small sequential queues of information between each other in the CICS/VS partition, using CICS/VS Transient Data facilities.

Data management design is discussed in greater detail in Part 4 of this manual.

DATA BASE DESIGN

Particular application data base characteristics and requirements are considered when selecting the best data base support. This can be based on CICS/VS File Control facilities or on one of the DL/I products.

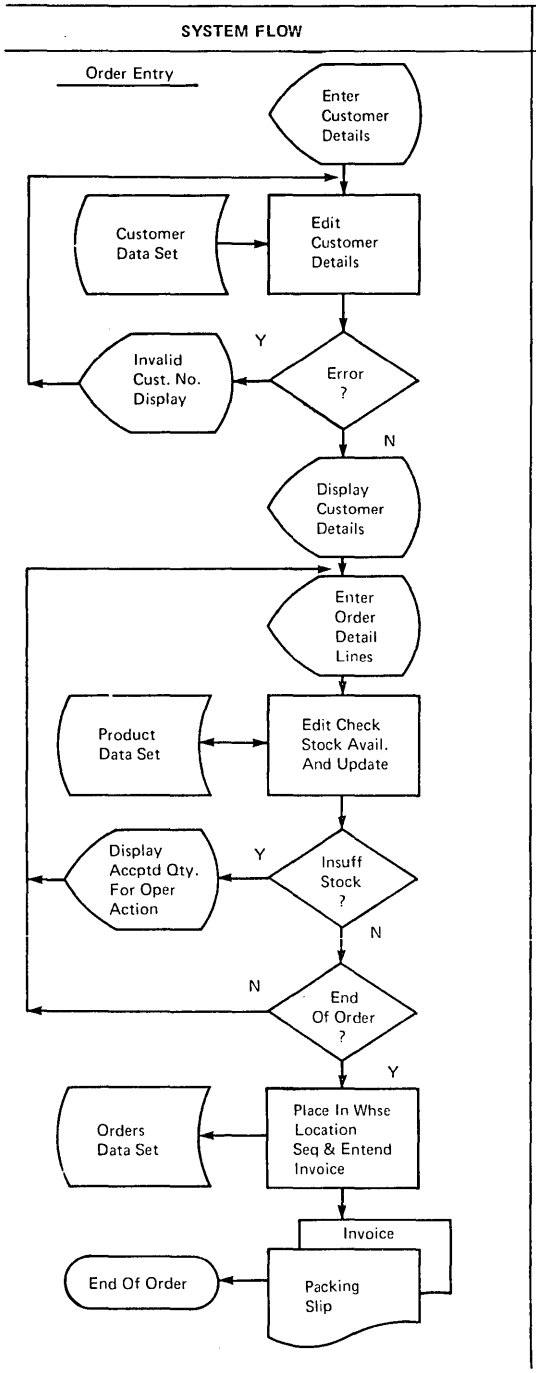


Figure 1.2-3. Order Entry and Invoicing Flowchart

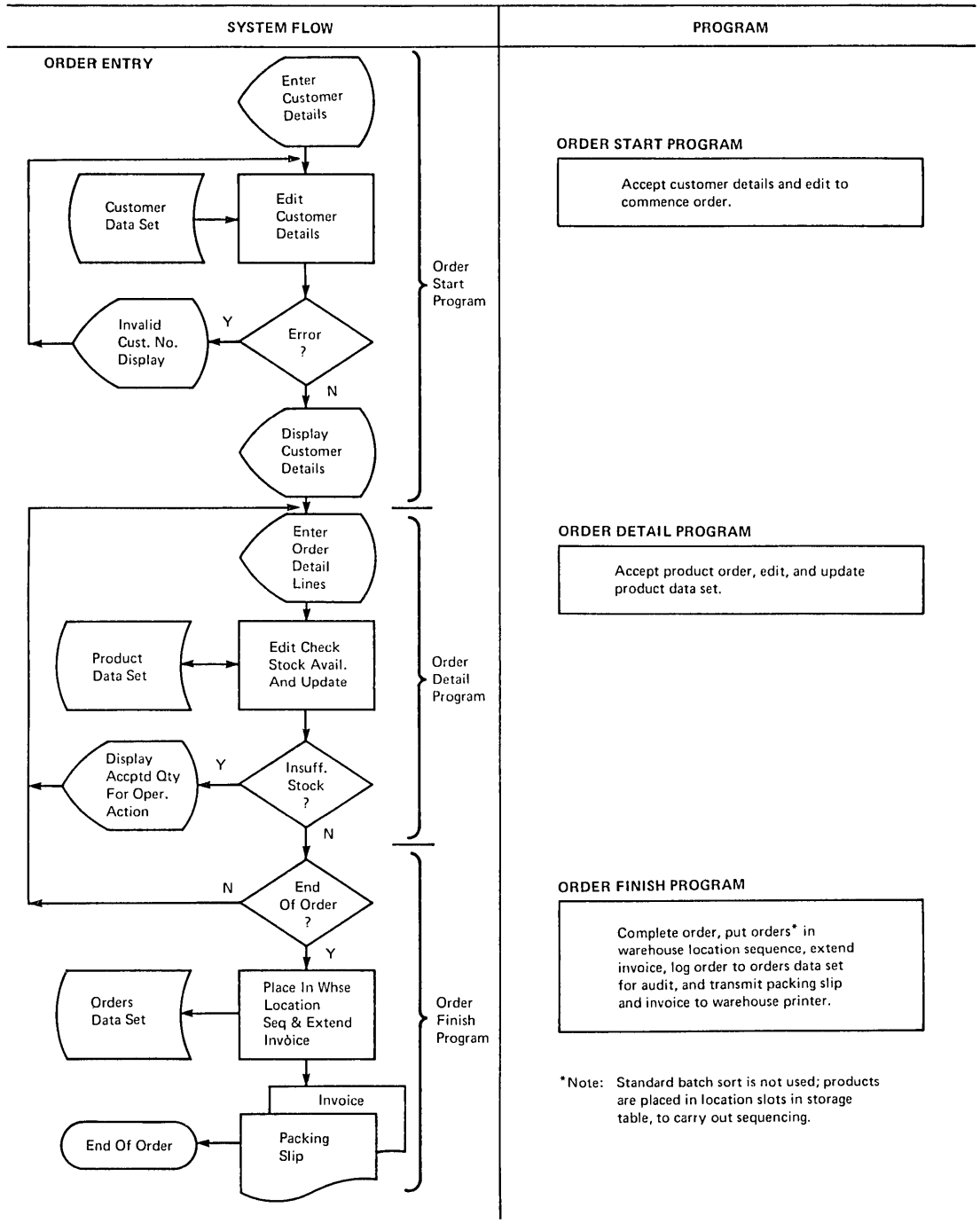


Figure 1.2–4. Order Entry and Invoicing Program Design

Factors to be considered in this decision include the need to access the data base from both online application programs and batch processing programs, and the number of ways in which information is to be retrieved, such as by the use of different record keys (for example, part number or part name in an inventory control application). Further factors in this decision are the number of times certain information occurs in each record, and the amount of information that may be absent in some records, yet present in others.

After selecting the appropriate data base support, the structure of the data base is designed, and how that data can be retrieved from application programs is defined. Figure 1.2-5 shows the design of a DL/I logical structure for the Order Entry application discussed above.

The effect of various errors and system failures on the integrity of the data base is considered, and a data base recovery and backup approach (if required) is defined.

Data base design is discussed in greater detail in Part 2 of this publication.

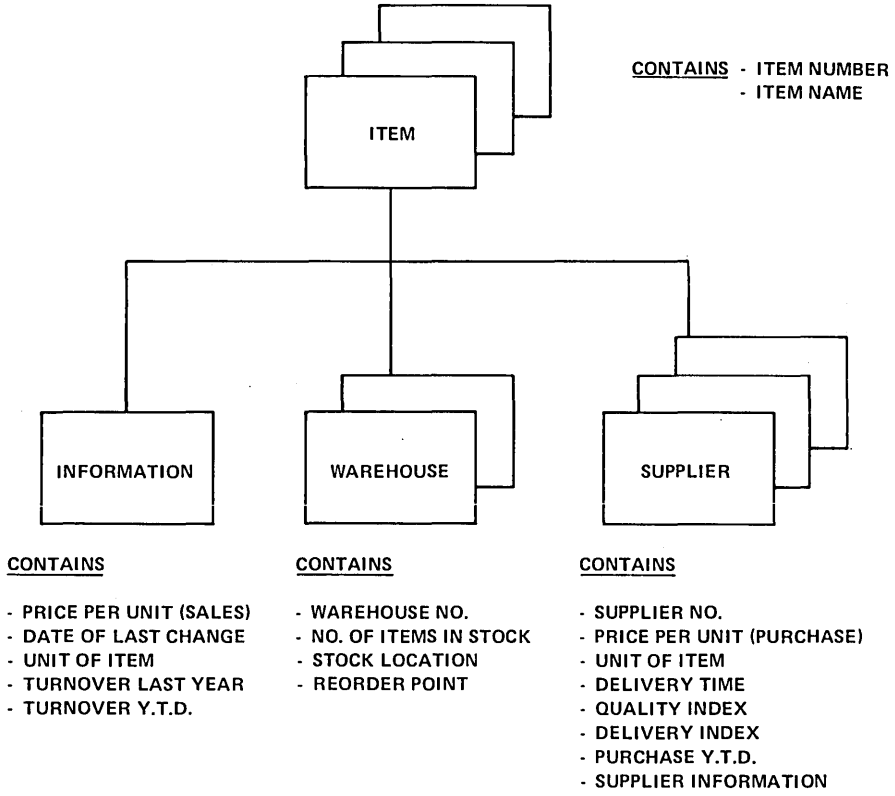


Figure 1.2-5. Order Entry Application Data Base Design

Part 2. Data Base Design

Chapter 2.1 Introduction

Application Requirements of Data Bases

DATA BASE DEFINITION

The term "data base" may have a different meaning to different online applications or installations. A general definition of a data base, which covers most considerations, is:

> "A structured nonredundant collection of interrelated information accessible to many users at the same time."

STRUCTURES

The term "structured" in the definition refers to the organization of information in a manner by which it can be easily retrieved. The following two structuring approaches can be used:

- Physical structure
- Logical structure

To require an application program to be aware of the physical structure of the data base implies that any change to the organization of information on that data base might also necessitate modification of the application programs which access the data base.

A logically structured data base is one in which an application program can refer to information in that data base by name, without necessarily being aware of the physical organization or location of data on the data base. The physical structure or organization may be separately described by a data description table, while the application program can describe its logical accessing and usage of the data using a program description table. These tables provide an interface between the application program and the physical structure of the data base.

The advantage of logical structures is that a change in the data base generally only requires a change in the relevant tables, often without necessitating any change in the application programs. This is termed data independence, and results in reduced maintenance of programs following modification of a data base.

Data bases which are referenced by physical structure usually have limited (or no) data independence, and programs may require considerable modification following a data base change. However, programs that refer to data bases logically, exhibit a much higher degree of data independence. Any data base changes are reflected in the data base tables and program tables rather than in the program itself.

DATA REDUNDANCY

The term "nonredundant" in the above definition refers to the ability of a data base to record certain information (for example, a customer's name and address) once only, but make that information available to other programs that use it.

Traditionally, batch applications and programs are developed with their own data sets, often disregarding information that is recorded on separate data sets for separate applications. The result in the traditional batch environment is the existence of redundant information—that is, the same information is often recorded in many data sets. A change to that information must be propagated through all data sets to ensure that the information remains in step across all applications. One advantage in recording information only once, and yet making that one record of information available to all applications, is that once that information is changed, the change is reflected across all applications that use the information. A further advantage resulting from nonredundant storage of information is storage economy, either on disk or tape.

COLLECTION OF INTERRELATED INFORMATION

The term "collection of interrelated information" in the definition refers to the consolidation of information relating to applications at one common point. The advantages offered by such consolidation include:

- More readily available information
- More timely information
- Elimination of redundant information
- Saving in disk or tape storage requirements
- Easier maintenance of information
- Development of information relationships

The last advantage listed refers to a significant advantage of data bases: the determination of the logical relationship of all information referring to a particular entity. The identification of such logical relationships of information enables that information to be utilized for better management of an organization's activities. This information may have been available previously, but may not have been utilized effectively before implementing the data base.

Data Base Implementation for Applications

DATA BASE REQUIREMENTS SUMMARY

The most common requirements of data base support are:

- Ability to support the multiple occurrence of information, with the number of occurrences varying from zero to many

- Utilize disk storage most efficiently, without requiring storage space to be allocated for information which is not present for a particular record
- Handle variable-length information such as names, addresses, or textual information for better disk storage efficiency
- Add, change, or delete records in a data base
- Add, change, or delete multiple occurrences of information for a record
- Nonredundant storage of information
- Data independence
- Access to the data bases by batch and online programs

MULTIPLE OCCURRENCE IMPLEMENTATION

Before examining the various data base support techniques available to determine how these can satisfy the above requirements, it is particularly important to examine the way in which multiple occurrences of information for a particular data base record can be implemented. The two techniques are:

- Physically related occurrences
- Logically related occurrences

Physically related occurrences generally are implemented by utilizing separate data sets. The main "root" information is stored in one data set. This may be specific customer data in a customer information system, account information in a savings bank and loan system, or product information in an order entry system.

The multiple occurrences of related information are then stored in a separate data set or data sets, and are related back to the main root information in the root data set by means of pointers. Furthermore, the separate occurrences of information relating to a root can be chained by means of pointers.

For example, in the banking industry, all the accounts relating to a bank's customers may be recorded in savings and loan account data sets, with each account record containing pointers which refer back to the customer's root information, such as name and address. Each account record for that customer may also contain a pointer to the next account for that same customer in a chain of accounts. A further data set, a transaction data set, contains deposits and withdrawals for accounts. Each transaction refers back to its related account record by means of a pointer, and to the next transaction against the same account in a chain of transactions, using another pointer. This is illustrated in Figure 2.1-1.

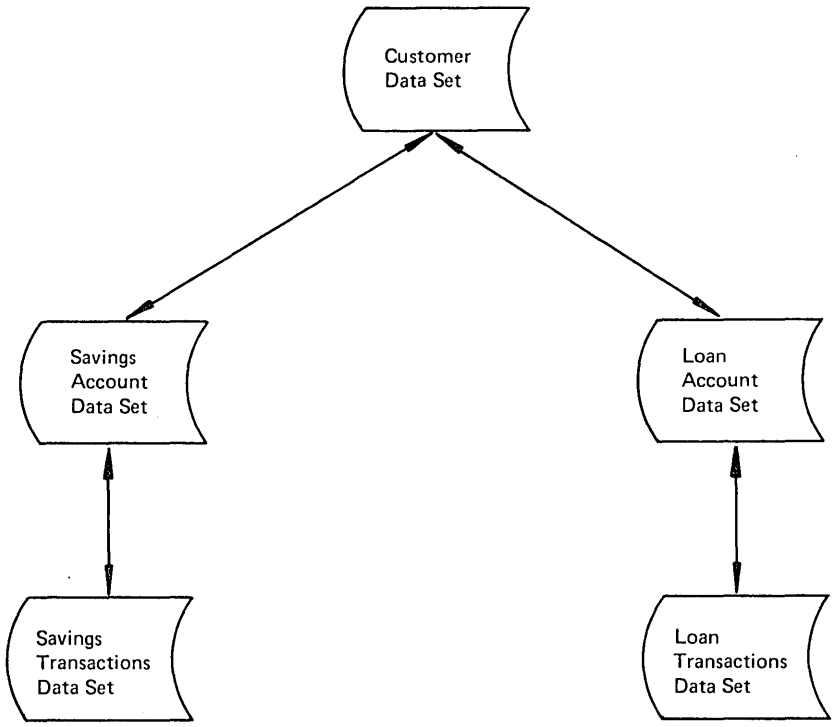


Figure 2.1-1. Savings and Loan Data Base Chaining

The separation of the root information in one data set, with the variable transaction information in other data sets chained logically to the root data set and also to other transactions for that same root, enables standard access methods to be utilized in providing data base support. The root information may be organized as a standard DAM (Direct Access Method), ISAM (Indexed Sequential Access Method), or VSAM (Virtual Storage Access Method) data set. Generally, the transaction data set would be organized as a DAM data set, or an entry-sequenced VSAM data set, to enable direct retrieval of transaction records. Retrieval of all the transactions relating to a particular root requires retrieval of the root information itself, followed by retrieval of each transaction in the chain—with a possible separate physical access for each transaction.

This physically related chaining technique may be supported by the CICS/VS file control indirect access feature, which is discussed in more detail later in this chapter.

The logically related technique for the multiple occurrence of information generally incorporates the multiple transactions in the same data set (or data base) with the root information. Most of the transactions relating to the root information are potentially accessible in fewer physical disk accesses than for physically related information. The data base support endeavors to place multiple transactions as close physically to their logically related root information as possible. For example, root information such as customer details is recorded immediately followed by multiple occurrences of information, each detailing a separate account for that customer and transaction activity against each particular account.

The data base support to implement a logically related technique must enable new information related to the root information to be added to other information for that root, existing information to be changed, or information to be deleted. This may require the utilization of internally controlled pointers and chains which are known only to the data base support and which are transparent to the application program. The application program may logically regard the multiple occurrences of information as if that information were physically adjacent to the root information.

Alternatively, the data base support may attempt to physically insert added information with the root and existing information, thus shifting along other information in the data base.

The data base support available for such logically related information is:

- CICS/VS file control segmented record feature
- DL/I products

These are discussed in outline in the next section, and in more detail in the next two chapters.

Data Base Selection Criteria

Several factors should be considered when choosing between CICS/VS file control and DL/I. The most important of these factors are outlined in the following sections. They should be borne in mind when reading the two subsequent chapters.

DATA BASE PERFORMANCE

CICS/VS File Control Accessing

Prime consideration in online application design should be given to the access time for retrieval of information from a data base. Depending upon the performance requirements of the application, this may dictate the selection of data base support. For example, if a data set may need to be accessed through other data sets, it may lend itself to the use of the CICS/VS file control indirect accessing feature. However, if several data sets have to be indirectly accessed to obtain the required information, these additional file accesses could have an adverse effect on online performance.

The particular access method selected with CICS/VS file control for the application may affect the performance. For example, the direct access method (DAM) generally provides excellent online performance. However, DAM support requires that records be identified by either their physical location on disk or their relative location in a data set. The application, on the other hand, may require that a record be accessed by a key. In this case, the indexed sequential access method (ISAM) may be suitable, but its use will involve at least two file accesses to retrieve each record. Furthermore, if many additions are made to an ISAM data set, the access time for a specific record may increase.

To overcome some of the above limitations, the virtual storage access method (VSAM) may be best suited. This enables records to be retrieved directly, based on relative location in the data set, or by key. It also enables rapid retrieval of information for applications with a high percentage of additions to the data set.

Another factor which should be considered is the serial scheduling of concurrently executing tasks, several of which may wish to update the same record in a data set at the same time (see "Exclusive Control During Update" in this chapter). CICS/VS will permit only one task to update a record at a time, and other tasks wishing to update that same record must wait for completion of the first update. (However, other records in the data set may be concurrently updated, if required.) This serialization of updates may affect performance, if application factors may cause concurrent updating of individual data set records to be attempted.

DL/I Accessing

DL/I provides a number of access methods which may be used for satisfactory performance depending upon the requirements of the application. These access methods are the: Hierarchical Sequential Access Method (HSAM), Hierarchical Indexed Sequential Access Method (HISAM), Hierarchical Direct Access Method (HDAM), and Hierarchical Indexed Direct Access Method (HIDAM). Refer to the System/Application Design Guide for the relevant DL/I product for further information about DL/I access method selection.

The CICS/VS-DL/I interface handles the data base activity from CICS/VS application programs on a multithread basis. Several CICS/VS application programs (tasks) may concurrently access the same, or different, data bases up to a maximum of 255 concurrent tasks for DL/I DOS/VS, 32 for DL/I ENTRY, or 15 for IMS/VS DL/I. CICS/VS allows concurrent access to DL/I data bases. To prevent double updating of a segment there are two scheduling methods available, Intent Scheduling and Program Isolation, the choice of which has a crucial effect on performance.

The CICS/DOS/VS-DL/I ENTRY interface permits multithread access to DL/I data bases, up to 32 tasks for DL/I ENTRY. In order to prevent double updating of a segment, DL/I ENTRY uses CICS/VS facilities to enqueue (between the GET HOLD and the REPLACE calls) on the logical record that contains the segment to be replaced.

BATCH PROGRAM ACCESS

If the online application data sets require further processing in a batch environment, this consideration should also be taken into account in selection of the data base support.

Factors which should be considered are that CICS/VS file control supports variable-length records within a fixed-length block for VSE ISAM data sets, standard OS/VS variable-length BISAM data sets, and blocked records for DAM data sets. VSE ISAM does not support these variable-length records for ISAM in a batch partition. They can be accessed in a batch environment by defining them to VSE ISAM as fixed-length unblocked records. However, the batch processing program must itself deblock the variable-length records from the fixed-length block returned to it by VSE ISAM.

Neither VSE DAM, OS/VS1, or OS/VS2 BDAM supports blocked records. If blocked DAM data sets are to be accessed in a batch environment sequentially, they may be defined as VSE SAM data sets or OS/VS BSAM or QSAM data sets. In this instance, the sequential access method will handle deblocking of records.

However, if the batch processing programs need to access these blocked records directly instead of sequentially, the responsibility rests with the batch program to define the data set as an unblocked DAM data set and provide its own deblocking of records within that physical block.

The use online of the CICS/VS file control indirect access and segmented record features requires that special coding to support these features in batch programs be developed by the installation.

The DL/I products support the same access methods and record formats both online and offline. No additional coding is required to enable batch DL/I programs to access online data bases.

SHARED DL/I DATA BASE (OS)

CICS/OS/VS has a shared DL/I Data Base feature that enables a batch region to simultaneously use the same DL/I data base as a CICS/VS transaction. The feature provides full data integrity by protecting against simultaneous update and by ensuring that if one of the regions fails then the data base is not left corrupted by the failing region.

If the batch program updates the data base then the CICS/VS system should make use of the program isolation scheduling facility. Long running batch updates should be split into small logical units of work by use of the CHKP call.

The batch program will appear to CICS/VS as another DL/I transaction so the systems programmer should be aware of overloading the CICS/VS system.

The CICS/DOS/VS user has a similar shared Data Base Facility provided by DL/I DOS/VS.

BATCH DATA BASE CREATION

CICS/VS file control provides no facility for creation of the online data bases, apart from that provided by standard SAM, VSAM, DAM, and ISAM support. The insertion of indirect access pointers in data sets, and the preparation and organization of segmented records, is the responsibility of the user. Generally, special data base creation programs must be written by the user.

Similarly, no facilities are provided for maintenance of the online file control data bases in a batch environment. To provide this, the user's data base creation program should also be designed to allow a maintenance capability.

DL/I allows creation and maintenance of data bases through the use of various utilities. Furthermore, because the program is independent of the physical data base organization and only refers to its logical organization, considerable flexibility is offered the installation in data base reorganization and maintenance.

INSTALLATION DATA BASE SUPPORT DIRECTION

In evaluating each of the selection criteria described above, the system designer must keep in mind the future direction for his installation in the use of particular data base support.

CICS/VS file control may be utilized if desired, because CICS/VS has been identified by IBM as one of its standard data base/data communications program products. However, the CICS/VS installation may wish to take full advantage of the extensive data base support provided by DL/I, by using the appropriate CICS/VS-DL/I Interface feature.

Chapter 2.2. DL/I

DL/I Products

This chapter is intended as an introduction to some of the significant features of DL/I, as used in the CICS/VS environment. It should not be considered as a substitute for the DL/I product documentation. No attempt is made to describe the operation of the various DL/I products in depth, nor to describe in detail the design of DL/I data bases. DL/I is discussed only in sufficient detail to help the CICS/VS system designer to evaluate the various DL/I products and the CICS/VS File Control facilities as data base support for online applications to run under control of CICS/VS. Refer to the appropriate DL/I General Information manual, System/Application Design Guide, and other DL/I manuals listed in the preface of this publication for further information about these DL/I products and the design of DL/I data bases. Additional information on DL/I, as used by CICS/VS, is given in the CICS/VS Application Programmer's Reference Manuals.

CICS/VS enables data bases created and maintained by the following DL/I products to be accessed by CICS/VS application programs:

- DL/I/DOS/VS ENTRY (CICS/DOS/VS only)
- DL/I/DOS/VS (CICS/DOS/VS only)
- IMS/VS DL/I (CICS/OS/VS only)

DL/I ENTRY DOS/VS

DL/I ENTRY DOS/VS provides a subset of the data base facilities offered by DL/I DOS/VS. It utilizes VSE SAM and VSAM on which DL/I access methods HSAM, HISAM, and HDAM are organized. Data bases may be created, maintained, and operated upon by batch processing programs. In addition, CICS/VS application programs may retrieve, update, add or delete information in DL/I ENTRY HISAM and HDAM data bases online. DL/I ENTRY does not support logging of DL/I activity, and does not provide data base recovery utilities that are available with DL/I DOS/VS and IMS/VS. However, since in an online environment DL/I ENTRY uses CICS/VS File Control services to map the DL/I access methods HISAM and HDAM onto VSE SAM and VSAM, the normal CICS/VS recovery functions can be used where appropriate.

DL/I DOS/VS

DL/I DOS/VS provides a subset of the data base facilities offered by IMS/VS (though not necessarily a compatible subset) and utilizes VSE SAM and VSAM as its standard access methods, on which are organized the DL/I access methods, HSAM, HISAM, HIDAM, and HDAM. These access methods are described later in this chapter. Data bases may be created, maintained, and operated upon by batch processing programs. In addition, CICS/VS application programs may retrieve, update, add, and delete information in DL/I DOS/VS data bases online. Data base recovery utilities are supplied for data base backout in the event of an uncontrolled shutdown and for data recovery following an unrecoverable I/O error.

IMS/VS DL/I

IMS/VS DL/I operates under control of OS/VS1 or OS/VS2. It utilizes VSAM as its standard access method (and also BISAM and a BDAM access method called OSAM). DL/I access methods HSAM, HISAM, HIDAM, and HDAM are organized on VSAM or ISAM/OSAM (see later in this chapter). Data bases may be created, maintained, and operated upon by batch processing programs. In addition, CICS/VS application programs may retrieve, update, add, and delete information in IMS/VS DL/I data bases online. Data base recovery utilities are supplied for data base backout in the event of an uncontrolled shutdown and for data recovery following an unrecoverable I/O error.

Unless specifically stated otherwise, the following discussion of DL/I support applies to each of the DL/I products previously described.

Full details of the individual DL/I products can be found in the appropriate IBM manuals, as listed in the Preface.

DL/I ACCESS FROM CICS/VS

Access to DL/I data bases online from CICS/VS application programs is achieved using the multitasking facilities of CICS/VS. Any CICS/VS application programs may concurrently access the same data base, up to the maximum number of active DL/I tasks specified for the CICS/VS partition, or the maximum number of concurrent DL/I tasks specified during initialization of the relevant DL/I product with CICS/VS.

DL/I ENTRY permits concurrent data base access up to a maximum of 32 tasks, DL/I DOS/VS up to 255 tasks, and IMS/VS DL/I up to 15 tasks.

DL/I utilities are used to describe the physical organization of data bases and the way in which programs will logically access the data bases defined. In addition, a number of DL/I utilities are provided to allow recovery of data bases in the event of I/O errors or system failures.

Introduction to DL/I

DL/I is a general-purpose data base control system that executes in a virtual storage environment under VSE, OS/VS1, or OS/VS2. It has been designed to simplify the user's task of creating and maintaining large common data bases to be accessed by various applications. Its design is open-ended, which allows future DL/I functions to be added without affecting existing functions. DL/I also allows growth to online applications through an interface with CICS/VS, and, on OS/VS only, through the data communications feature of IMS/VS.

DL/I has been developed by IBM to serve two application areas:

- Batch processing
- Online processing

In batch processing, single data base transactions requested by applications are accumulated and processed periodically against the data base. Because of the elapsed time, data in the data base is not always current. The use of batch processing should depend on how current the user's information must be, viewed in relation to the cost of other methods of processing data.

For online processing, IMS/VS DL/I may be used in conjunction with either the IMS/VS DC feature, or with CICS/OS/VS. With DL/I ENTRY and DL/I DOS/VS, data communications support is provided only through CICS/DOS/VS. The use of online processing, as opposed to batch processing, enables a response to be generated for each transaction as it is requested. This reduces the elapsed time inherent in batch processing systems and allows the user to maintain current data for his applications.

Traditionally, data used by application programs is organized in data sets. Each data set is physically structured to present data in the physical sequence and format required by the particular application program, and each program contains a description of the data set organization and record format as an integral part of the program (see Figure 2.2-1). When the same data is shared by many applications (common data), the data is duplicated on different data sets so that it can be presented to each application program in the physical sequence and format required. This duplication uses additional storage space and results in increased maintenance time and cost, since the same data has to be maintained simultaneously in many locations. Futhermore, when the data set organization or record format must be changed, each program which accesses that data set must be modified to reflect the changes. This traditional data set approach is illustrated in Figure 2.2-1.

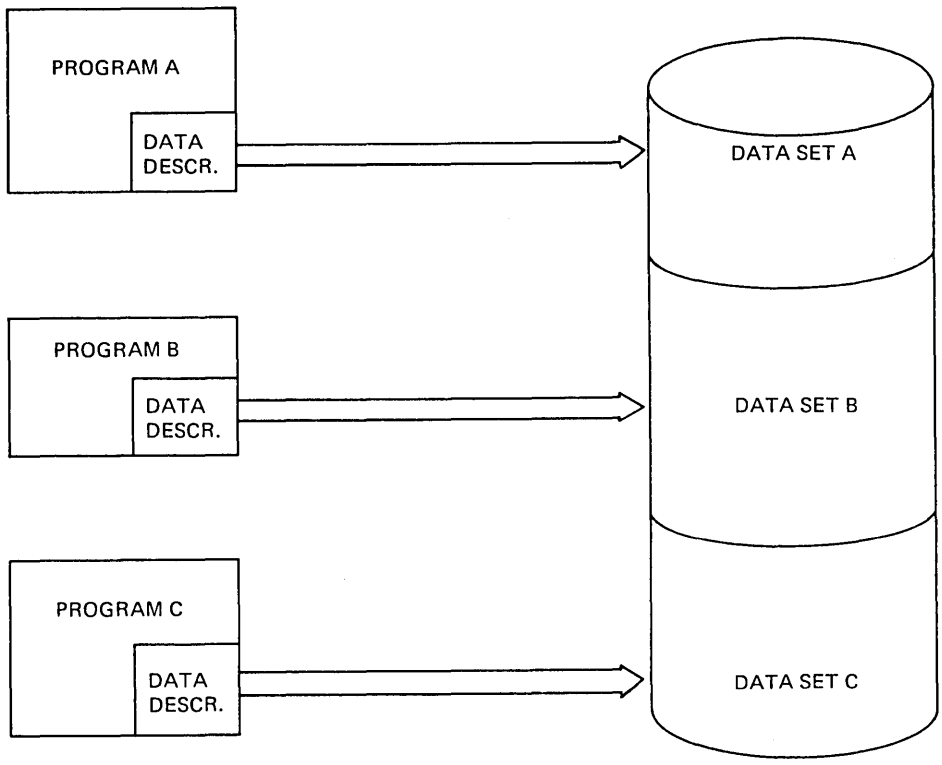


Figure 2.2-1. Traditional Data Set Approach

DL/I enables programs to be freed from their dependence on data set organization and record format. The description of the physical organization of a data base is removed from programs and contained in a separate data description table. Each program utilizes this data description. DL/I extracts the requested information from the data base to present to the program. This is illustrated in Figure 2.2-2. Because programs using DL/I are no longer dependent upon the physical organization of data, when the data base organization must be changed, generally only the data description table need be changed. Programs which access the data base using the data description in most cases need not be aware that the data base has changed, and generally need no modification.

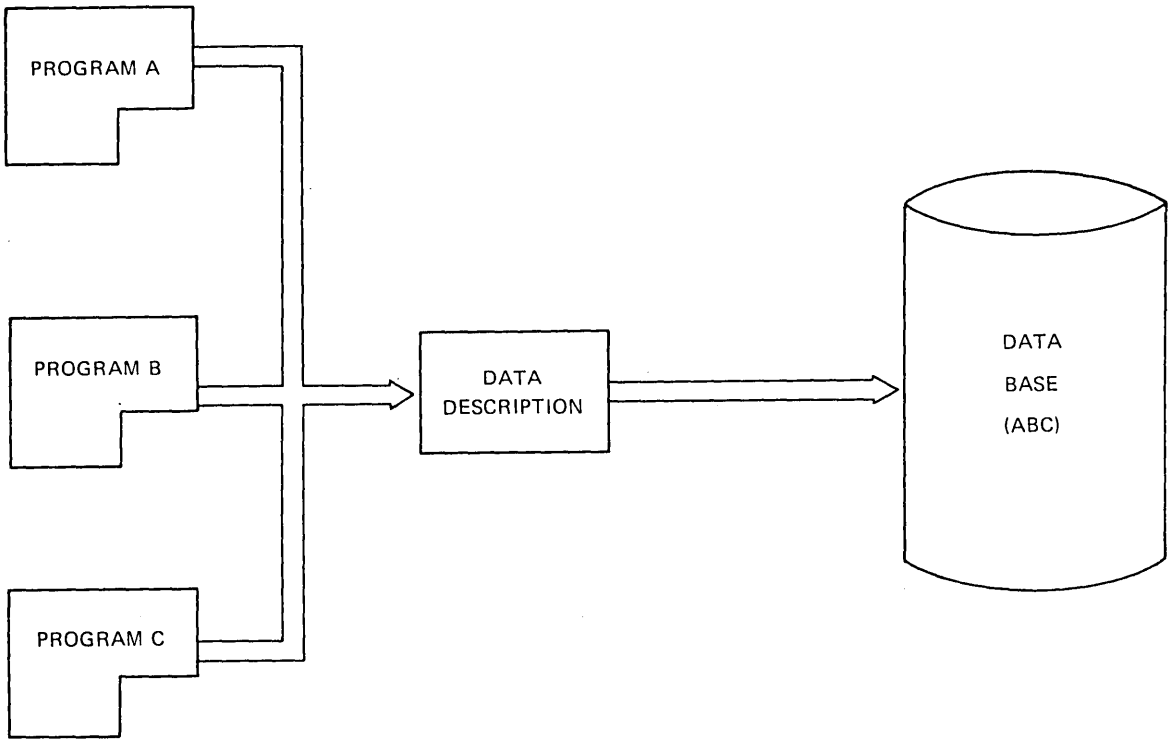


Figure 2.2-2. DL/I Data Base Approach

All application data is stored in one or more data bases in a hierarchical fashion; that is, the most significant data resides on hierarchically higher levels, while less significant but related data (dependent data) appears on hierarchically lower levels, as illustrated in Figure 2.2-3. This hierarchical approach enables programs to view data in a data base apart from its physical organization. Through the use of a concept called "sensitivity," each application program views only that data in the data base which it uses. (Sensitivity is discussed further, in "Logical Data Structures," later in this chapter.)

DL/I accesses data in the data base and presents only the information requested by the program. The data presented to the program by DL/I is called a "segment." A program requests a segment from DL/I by issuing a DL/I call.

In practice, a system designer reviews the data requirements of all applications (as illustrated at the start of this chapter), then defines the data base or bases. To create a data base, the user defines to DL/I a common data structure and format that serve his applications and loads his application data into that data base.

The definition of the data base is provided by a data base description (DBD), and a DBD is required for each data base (see Figure 2.2-3). This is generated prior to the loading of data into the data base, by assembling a set of DBD macro instructions which define the data base.

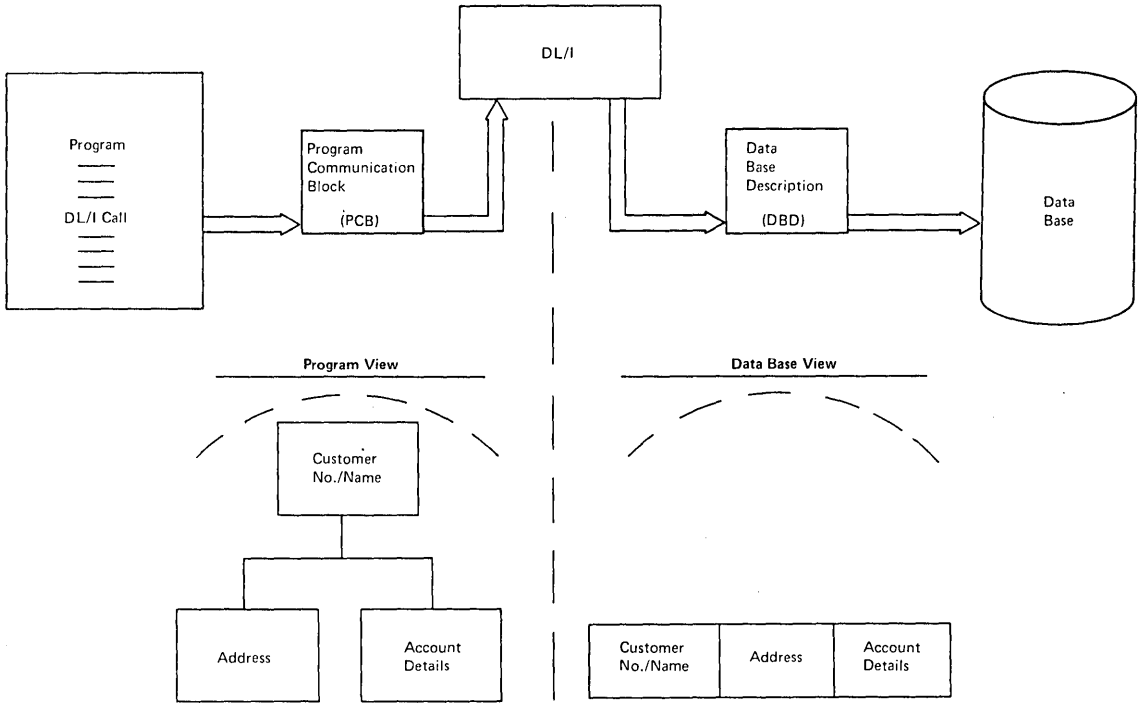


Figure 2.2-3. DL/I Data Base Access

The second definition required is the program specification block (PSB). The PSB defines the data base processing requirements of an application program. It identifies those sensitive data elements (segments) in the data base that are available to the application program which uses the PSB, and the way in which the program views the data base. Although a transaction may only reference one PSB at a time, CICS/VS permits a transaction to dynamically change from one PSB to another through the PSB schedule and termination mechanisms.

The PSB is generated by assembling a set of PSB macro instructions which define the above factors.

Through DL/I's use of the DBD and PSB, application programmers can write their programs without much regard to the physical structure of data. Instead, they refer only to segments of data as needed by the program, without consideration for the physical location of that data in the data base.

APPLICATION PROGRAMMING INTERFACE

An application program can retrieve, replace, delete or insert data in a DL/I data base by means of either a high level programming interface (HLPI) or a CALL interface. The HLPI uses EXEC DLI program statements in a way similar to the use of EXEC CICS statements for interfacing with CICS/VS. Further information is given in the DL/I High Level Programming Interface User's Guide .

The HLPI is available to PL/I and COBOL users under VSE. The CALL interface is available to PL/I, COBOL and assembler users under VSE and OS/VS.

A special interface (RQDLI) is available for RPG users.

ADVANTAGES OF DL/I

DL/I provides application independence from access methods, from physical storage organization, and from the characteristics of the devices on which the data of the application is stored. This independence is provided by a common symbolic program linkage and by data base descriptions external to the application program. A reduction in application program maintenance is generally realized.

DL/I provides for the reduction, and possible elimination of, redundant data or sharing of common data. The majority of the data utilized by any company has many interrelationships that can cause significant redundant storage of data if conventional organization and access methods are used. For example, manufacturing and engineering departments work with subset data which is also useful to quality control.

The storage organization and access methods employed by DL/I facilitate data integration with a minimum of data redundancy. However, if an analysis of a company's data shows that all of the data cannot be placed in a single common data base, DL/I allows the user the additional capability of physically structuring the data across more than one data base. Before DL/I, application programmers frequently did not have the time or ability to integrate other data with their own data to eliminate redundancies without the necessity of a major rewrite of the application programs involved.

Chapter 2.3. CICS/VS File Control Facilities

Introduction to CICS/VS File Control

The CICS/VS file control program provides data base support for application programs executing under its control. It uses the standard access methods available under VSE and OS/VS1 or OS/VS2 — namely the Indexed Sequential Access Method (VSE ISAM or OS/VS BISAM), Direct Access Method (VSE DAM or OS/VS BDAM), and Virtual Storage Access Method (VSAM). For the remainder of this chapter, "DAM" will be used to refer both to VSE DAM and OS/VS BDAM, and "ISAM" will refer to both VSE ISAM and OS/VS BISAM.

The facilities provided by the standard access methods are extended in some cases by CICS/VS file control to provide additional support. For example, file control supports the following data sets:

- Fixed-length and variable-length records
- Blocked and unblocked data sets
- ISAM, DAM, and VSAM

Note: Fixed block architecture (FBA) devices are supported by CICS/VS file control using VSAM only.

Extensions provided by CICS/DOS/VS file control enable the support of variable-length VSE ISAM data sets, which are not part of standard support provided by VSE ISAM. Similarly, file control provides support for blocked fixed-length or variable-length DAM data sets, which are not included in the standard support provided by VSE DAM or OS/VS BDAM. The support of blocked direct access data sets is particularly useful if those data sets are processed sequentially by CICS/VS programs, as discussed below in "Sequential Access (Browsing)." CICS/VS file control allows both direct access and sequential access to ISAM, DAM, and VSAM data sets.

A VSAM function provides the ability to reuse a VSAM data set without redefinition. CICS/VS support allows reusable files to be defined, via the SERVREQ= operand of the DFHFCT TYPE=DATASET macro (refer to the CICS/VS System Programmer's Reference Manual). These reusable files are available to application and system programmers as temporary files and/or for test purposes. The user should be aware, however, that there is no recovery support for reusable data sets; if the system is taken down for any reason then the file is closed and the data is lost.

Direct Access

Direct access, sometimes referred to as random access, is supported by file control for ISAM, DAM, and VSAM data sets. The following services are provided by CICS/VS file control for DAM, ISAM, and VSAM:

- Random record retrieval
- Random record update

- Random record addition
- Random record deletion (VSAM only)
- Logically open/close data sets
- Exclusive control of records during update operations
- Variable-length ISAM records (both VSE and OS/VS)
- Blocked DAM records
- LOCATE mode, read-only retrieval (VSAM only)
- Mass record insertion (VSAM only)
- Segmented records
- Indirect access

These services enable CICS/VS file control to provide data management support that surpasses OS/VS or VSE data management support in many areas.

Direct access to data sets is made on the basis of record identification of the particular logical record to be retrieved. The record identification may be either a record key in the case of ISAM or key-sequenced VSAM data sets, or a record location within the data set for DAM or entry-sequenced VSAM data sets. The use of record keys or locations for direct access is discussed in more detail under "Record Identification."

Based upon presentation of the appropriate record identification by the application program, CICS/VS file control will access the data set requested by the program to carry out the services listed above, and described in detail in the following sections.

RANDOM RECORD RETRIEVAL

File control will directly access the record identified by the application program using either key or record location (depending upon the type of data set) from the specified data set. The application program issues a file control READ command, identifying by name the data set to be accessed, the data area within the program into which the record is to be read, the location in the program which contains the record identification. The data set name is used by CICS/VS to locate the relevant entry for that data set in the file control table (FCT). This entry contains specifications for that data set, such as:

- Access method used
- Record length
- Block length
- Key length (if applicable)
- Key location (if applicable)

This information is not contained within the application program. In the event of a change to the data set, the relevant changes may be made

to the FCT, without affecting the application program. This provides a limited degree of data independence.

When the application program issues a READ command, the input operation begins. The application program waits until the requested operation is completed. Any I/O errors on completion, which cannot be recovered by the access method or by file control, are then returned to the application program for action.

Although an application program does not continue processing while a requested I/O operation is being carried out, CICS/VS utilizes the available processing time during the I/O for other concurrently executing tasks. Consequently, all tasks are given an equal opportunity to process, based upon their respective task priorities, while I/O is in progress. The net result is improved overall performance of all concurrently executing tasks in the system, even though the full processing overlap potential of the single task issuing the I/O operation request is not utilized.

RANDOM RECORD UPDATE

A record can be directly accessed using a file control READ command, as described above, for potential subsequent update. An indication that this record may subsequently be updated is made by the application program at the time that the READ command is issued, by specifying the UPDATE operand.

In this case, the record is retrieved as described above for the READ command. After the application program has updated the record, it issues a REWRITE command, supplying to CICS/VS the name of the data area holding the record. The logical record then replaces the original record on disk.

If the application program does not wish to update the record which was retrieved, it does not issue a WRITE command. The application program should issue an UNLOCK command.

EXCLUSIVE CONTROL DURING UPDATE

If the exclusive control feature was specified when the file control routines were generated then file control will protect record integrity during updating.

- Exclusive Control and ISAM Files

  CICS/VS maintains exclusive control over an ISAM logical record that is read for update. If another task attempts to access the record before exclusive control is released then a lockout will occur.

- Exclusive Control and DAM Files

  If a DAM logical or physical record is to be updated, CICS/VS will maintain exclusive control over the physical record. If another task attempts to access that physical record or if the same task attempts to access another logical record within that physical record before exclusive control is released then a lockout will occur.

- Exclusive Control and VSAM Files

  VSAM maintains exclusive control over a control interval that contains a record that is to be updated. If the same task or another task tries to read or add a new record within that control interval before exclusive control is released then a lockout will occur.

RANDOM RECORD ADDITION

Records may be added to a data set through the use of a WRITE command. The record identification supplied by the program is used to determine where the new record will be added.

If the record identification provided is a record key for addition of new records to ISAM or key-sequenced VSAM data sets, the record is placed in sequence in the data set based upon that key. For DAM data sets, the new record is inserted as close as possible to the specified record location as described below.

For fixed-length unblocked DAM records, such data sets must be initially generated with a number of dummy records interspersed throughout the data set. A dummy record is one containing hexadecimal FF in the first byte of the record. The record to be added is inserted in the first available dummy record location following the specified record location. If no dummy records are available in the same cylinder (for VSE), the application program is notified; it may then reissue the WRITE request for the new record to another part of the data set until a dummy record is found. When the new record replaces the dummy record, file control returns the record location where the new record is stored to the application program in the record identification field.

Notes:

1. Record addition is not possible for VSAM files that have been specified as ICIP. Such files should revert to normal VSAM if record addition is required.

2. CICS/VS cannot be used to extend (add a record with a key higher than any existing record) an ISAM dataset.

For variable-length record DAM data sets, CICS/VS file control attempts to add the new record at the end of the specified track, for CICS/DOS/VS, providing there is sufficient space on that track to contain it. For CICS/OS/VS, a specified number of tracks may be searched to locate a track on which to add the record. If there is not sufficient space, the application program is notified, and may reissue the WRITE request for the new record, indicating another track to be used. When the new record has been successfully written at the end of the specified track, its record location is returned to the application program in the record identification field.@@@@@@@

For entry-sequenced VSAM data sets, new records are always added to the end of the data set regardless of whether they are fixed or variable-length. The relative byte address of the added record in the data set is returned to the application program.

RANDOM RECORD DELETION (VSAM ONLY)

The file control DELETE command is used to specify the deletion of records in a VSAM key-sequenced data set. The specified record is physically deleted. The space occupied by that record is reclaimed and added to the available free space in the particular control interval which contained that deleted record.

LOCATE MODE PROCESSING (VSAM READ-ONLY)

The normal mode of processing for file control operations is move mode. With mode processing of blocked data sets, the logical record is moved from the block into a FWA, and the address of that FWA is presented to the application program.

For VSAM data sets, locate mode processing may be specified for read-only operations. With locate mode processing, the address of the logical record in the control interval is stored in a virtual storage work area (VSWA). The additional processing required to move the logical record from the control interval is therefore avoided. However, locate mode is invalid if a read for update is specified and/or segmented records are being retrieved.

BLOCKED DAM RECORDS

CICS/VS file control provides for the deblocking of logical records in a blocked direct access (DAM) data set. This service is provided for both fixed-length and variable-length records. When creating or adding to blocked DAM data sets, the application program must work with entire blocks.

The advantage in supporting blocked DAM records is to enable both direct and sequential access of the data set. The block size should be such that the physical record retrieved for direct access is maintained as small as possible, while still providing sufficient blocking to enable satisfactory performance for sequential retrieval.

CICS/VS will support the update of variable length records within fixed or variable length blocks provided that the length of the records is not changed.

VSE ISAM VARIABLE-LENGTH RECORDS

CICS/DOS/VS supports the retrieval and static update (that is, no length variation) of variable-length records within a fixed-length block under ISAM organization. These pseudovariable blocks must contain the block length in the first four bytes in the standard form LLbb. Since all blocks are fixed-length, this value is the same for all blocks. Each logical record within the block must also reflect the length of the record in the first four bytes (LLbb). A logical record may not be continued into the next block. The first byte of any unused portion of a block must contain a hexadecimal FF.

The addition and deletion of records for a VSE ISAM variable-length record data set must be handled by the user in an offline batch

environment. When creating the data set, it must be defined as fixed unblocked, and the key for each block must be the same as the last logical record in that block. The block size must be an even number of bytes. All records must reside in the prime data area; no overflow records are allowed.

However, the use of key-sequenced VSAM data sets instead of ISAM allows the support of both fixed-length and variable-length records, with the added advantage that the record length can be either increased or reduced as a result of a record update, addition, or deletion.

DYNAMIC OPEN/CLOSE OF DATA SETS

When the CICS/VS system is initialized, data sets may be specified in the file control table (FCT) as either open or closed. Closed data sets may be dynamically opened for accessing at a later time, by means of a master terminal command. The design techiques described below utilize dynamically opened or closed data sets.

Data sets may be dynamically closed at certain times of the day, to prevent access to information from terminals, and may be dynamically opened when access is to be permitted. In this way, support for certain online applications may be provided only when desired. If an application program attempts to access a data set which has been closed, an error indication is returned to the program.

MASS RECORD INSERTION (VSAM ONLY)

If many records are to be added to a VSAM data set and those records have keys that are in ascending sequence then significant performance gains can be achieved by using mass record insertion. This is specified with the MASSINSERT option of the WRITE command.

VSAM SHARED RESOURCES

VSAM shared resources enable a pool of I/O related blocks, channel programs, and buffers to be shared among several VSAM data sets. This permits efficient utilization of storage in an environment in which many VSAM data sets are open and it is difficult to predict the amount of activity against a given data set, or in a situation where each transaction may access several VSAM data sets.

The user indicates in the FCT which VSAM data sets are to share resources. CICS/VS calculates the maximum amount of resources required by using the number of strings specified in the FCT for each of the VSAM data sets that are to share resources and the control interval sizes for these data sets from the VSAM catalog. CICS/VS then requests VSAM to build a resource pool large enough for a certain percentage of maximum amount of resources required. The user can override this percentage and resource calculation if desired. For example, the user may wish to override the CICS/VS calculation to reflect specific data set activity known only to the user.

Storage utilization efficiency obtained by sharing VSAM resources must be evaluated against the effect on performance. If insufficient resources are available to satisfy a specific I/O request against a

shared resource data set, the requesting task is placed in a CICS/VS wait until the necessary resources become available. CICS/VS provides statistics (number of strings, buffer sizes, and number of buffers of each size) to identify the resources allocated. Statistics are also provided to aid in the optimization of these resources to ensure that sufficient buffers and VSAM strings are available to avoid excessive task wait time.

If the activity against specific data sets is higher than can be managed using shared resources, those data sets should be defined in the FCT as not sharing resources.

Sequential Access (Browsing)

The operations discussed above refer to direct access. CICS/VS file control enables DAM, ISAM, and VSAM data sets to be sequentially as well as directly accessed. This sequential access is sometimes referred to as a "browse" operation. Data sets to be browsed may be either fixed-length or variable-length, blocked or unblocked data sets.

A browse operation using CICS/VS file control is analogous to the sequential retrieval of records from ISAM data sets, sometimes called SETL retrieval, in a batch environment. However, a batch program can only sequentially retrieve records from one logical section of a data set at a time. On the other hand, CICS/VS enables many browse operations to be concurrently executed on the same data set, either from the one task or several tasks. This is referred to as multiple browsing, and is discussed further below.

BROWSE INITIATION

To specify a browse operation, the application programmer identifies the data set to be browsed, and provides the record identification of the logical starting point in the data set for the browse operation. This logical starting point can be either a specified record location, or key, or a generic key. For example, if it is desirable to browse an orders data set, containing orders for products placed by different branches, a generic key may indicate that browsing is to start with the first order recorded from a specified branch. The initiation of a browse operation is achieved by the application program issuing STARTBR command.

BROWSE RETRIEVAL

Each record is sequentially retrieved for the browse operation when the application program issues a READNEXT command. Each READNEXT presents the next sequential logical record to the application program for processing. In the case of an ISAM data set or a key-sequenced VSAM data set, the records are presented in ascending key sequence (except for a browse operation using relative byte address (RBA) for a key-sequenced data set, when records may be presented in physical sequence). For a DAM data set, or an entry-sequenced VSAM data set, the records will be presented in the sequence in which they are physically stored on the data set.

BROWSE TERMINATION

The browse operation continues with each subsequent READNEXT, until the end of the data set is reached or it is desired to terminate the browse operation. This termination is achieved by issuing an ENDBR command.

MULTIPLE BROWSING

A task can issue one or more STARTBR commands to initiate one or more browse operations. Each one must have a unique halfword binary value specified in the REQID operand. This value is then specified in the REQID operand of corresponding READNEXT commands.

There is no logical limit to the number of browse operations that may be executed concurrently for the same data set, either from the same task or many tasks. The only limitation is the availability of data areas and buffers. This is a factor of the block length, record length, degree of multitasking, and amount of dynamic storage allocated in the CICS/VS partition, and number of VSAM strings specified for a VSAM data set.

The multiple browse technique introduces a number of very useful system design solutions. For example, an orders data set may contain orders that are in sequence according to product number as placed from several branches within a company. If it is desired to retrieve all product orders from a specific branch, this can be achieved by issuing a browse operation, starting the browse at the first product number ordered from that branch. Subsequent READNEXT commands will retrieve the next product ordered from the branch, until the end of all products ordered from that branch is reached. At this time the browse operation may be terminated by an ENDBR command.

However, if the product orders received from several branches are reported using a terminal, this may imply that the orders data set should be sorted into the sequence of branch number within product number.

Generally, online sorting is impractical. Records should be retrieved in the sequence of branch within product, while still maintaining the orders data set in the sequence of product within branch. This can be achieved by issuing multiple browse operations, having each browse initiated from the first product order record from each branch. Each browse operation in effect logically breaks up the orders data set into a number of separate order data sets, one for each branch.

A READNEXT command can be issued for each branch browse operation to retrieve the orders placed for the first product number in the small logical data set for each branch. The second READNEXT issued for each browse operation then retrieves the next product order record for each branch.

This can continue, retrieving all the information from each branch relating to a specified product until the application program has constructed an entire terminal page. At this time, the browse operations for the products and branches contained on that terminal page may be terminated by issuing an ENDBR command for each browse.

As previously stated, the technique of multiple browsing enables the sequential retrieval of information in a sequence different from that in

which a data set is organized. This multiple browsing design technique may open up powerful data inquiry possibilities for online data sets.

SKIP SEQUENTIAL BROWSING (VSAM ONLY)

Skip sequential refers to the ability to sequentially browse through a logical section of a data set, and then skip to another logical section of a data set to continue the same browse operation. In effect, it provides a direct access capability in the middle of a sequential retrieval operation. The record identification of the next logical section of the data set may be moved into the record identification field set up in the program, and another READNEXT command can be issued. This will position the browse to the new section of the data set, thus effecting a skip sequential operation. This technique cannot be used for DAM or ISAM data sets.

BROWSING BACKWARDS (VSAM ONLY)

The CICS/VS support of this VSAM function allows the user to browse backwards through a VSAM file.

Once a browse operation against a VSAM data set has been initiated, the next sequential record toward the beginning of the data set can be requested by issuing a READPREV command.

The definition of the previous record depends on the type of browse operation. For browses by key, the previous record is the one which has a key which is next in descending order. If, however, the records have identical (non-unique) keys, they are returned in the order in which they were added to the data set.

Record Identification

As discussed above, data sets supported by CICS/VS file control can be accessed either directly or sequentially. Records are accessed based upon the record identification supplied by the application program. The record identification utilized depends upon the particular data set being accessed. There are two types of record identifications:

- Record key
- Record location

Record identification based upon a key is used to access ISAM data sets and key-sequenced VSAM data sets. The key may be either a full key for retrieval of a particular logical record, or a partial (generic) key, to indicate a logical point in a data set from which a browse operation is to commence. This generic key contains sufficient information in the high-order bytes of the key to uniquely identify the logical section of the data set. The remaining low-order bytes of the key may be either binary zeros or blanks. For key-sequenced VSAM, a truncated generic key may be utilized, with the first byte of the key specifying (in binary) the number of significant bytes in the generic key which follows.

For instance, the orders data set discussed above for browsing can utilize a key containing a branch number in the high-order bytes of the key, and specific product numbers in the low-order bytes of the key. For example, orders for product number 1016 from branch number 12 may be contained in a record which utilizes the key of 121016. The generic key to enable the first product record to be accessed for branch number 12 would then be the generic key 120000 for ISAM, or 212 for VSAM. The "2" indicates (in binary) that a generic key of length two bytes follows, for branch number 12 in this example.

When a full record key is used to access an ISAM data set, it must locate a record on that data set with the identical key; otherwise, an error indication is returned to the application program.

However, when a full record key is used to access a key-sequenced VSAM data set, any search for relevant VSAM records must be specified as:

- Full Key Equal - indicates that the key provided by the application program is a full key, and failure to locate a record with this exact key will result in an error indication being returned to the application program.

- Full Key Greater or Equal - specifies that the record key is a full key, and that the first data record with a key equal to or greater than the supplied record key is to be retrieved. This is equivalent to using a generic key in ISAM.

- Generic Key Equal - indicates that the record key is a generic key with a specified generic length. A record whose key is equal to the supplied generic key for the number of bytes indicated is then retrieved. If one cannot be found, a "no record found" condition is returned to the program.

- Generic Key Greater or Equal - indicates that a generic key is provided, and the first data record with a key equal to or greater than this generic key for the number of bytes indicated is to be retrieved.

An additional advantage in the utilization of record keys is in the addition of records. When new records are added to the data set, they are inserted in sequence in the data set based upon their record key value.

Relative Record Number (VSAM Only): This VSAM support allows users to define a type of VSAM data set, called a Relative Record Data Set (RRDS), which can be accessed by a relative record number. In the case of a RRDS, the key to a record is its relative record number, and access is by this number only.

Alternate Indexes (VSAM Only): This VSAM support allows users to construct and maintain alternate indexes to a data set thus permitting access to it via multiple keys. The major difference with an alternative index is that there can be several records in the data set which have the same alternative key. The CICS/VS support enables the user to access his data set via an alternative path and to handle duplicate keys. The user should be aware that if the data set is going to be updated then any other indexes may no longer be valid. Equally, if he has defined his alternative indexes as part of an update group, the several indexes will automatically be updated by VSAM. These factors will affect performance.

There is a database integrity exposure in VSE when updating a path (or paths) and the base dataset at the same time. The VSE Share options cannot be used to provide such integrity for files accessed in the same partition.

Also if transactions update a path and the base dataset at the same time CICS/VS logging will no longer guarantee data base integrity. Logging will only function correctly if all updates are made via a single path or via the base dataset.

It is strongly recommended that transactions get read-only access via paths and perform all updates on the base dataset. Failure to do this may result in loss of integrity of the database. Also, it should be noted that, under VSE, records retrieved by read-only access via a path may be invalid if concurrent update activity is taking place, because the path buffers are not refreshed when an update is performed on the base data set.

Spanned Records (VSAM Only): CICS/VS does not support VSAM spanned records.

Reusable Dataset (VSAM Only): This VSAM feature causes the contents of the dataset to be deleted when the dataset is opened. When the reusable dataset has been closed, the data in it can be accessed with another File Control Table entry which is not specified as reusable.

RECORD LOCATION

To facilitate retrieval from DAM or VSAM data sets, records are identified by their locations in the data set. VSAM record identification is based on relative byte address (RBA) within the data set. In the case of DAM, the physical block (record) identification can be on the basis of:

- Actual disk address (MBBCCHHR)
- Relative track and record within the data set
- Relative block number (for CICS/OS/VS only)

If a physical key is recorded for the physical record, it may be appended to each of the record identifications detailed above. Figure 2.3-1 shows some representative record identification field formats.

DAM data sets with or without physical keys can be accessed. If a physical key is recorded on disk preceding the data record, the record identification can indicate the relative track within the data set, and the key which is physically recorded with the data record to be retrieved.

Both blocked and unblocked DAM data sets are supported by CICS/VS file control. In the case of blocked DAM data sets, additional information may be provided to identify the logical record within the physical block. This logical record identification immediately follows the physical block identification (as detailed above) in the record identification field provided by the application program. Logical records may be selected from a physical block based upon:

- Record number within block
- Record key within block (as illustrated in Figure 2.3–2)

where the location of the record key within each logical record is defined in the file control table.

CICS/VS file control uses this logical record number or key to deblock the relevant logical record from the physical block and present it to the application program.

For VSAM data sets, the record location utilized is a relative byte address (RBA). VSAM data sets use this relative byte address to identify the location within the entire data set of information (such as a logical record) to be retrieved.

PHYSICAL RECORD (BLOCK) LOCATION	PHYSICAL KEY (IF PRESENT)	DEBLOCKING ARGUMENT (IF BLOCKED)	COMMENTS
Relative Block No.	-	Record No.	CICS/OS/VS Only
Relative Block No.	Key	Record Key	
TTR	-	Record No.	Relative Track and Record (binary)
TTR	-	Record Key	
TTR	Key	Record Key	
TTTTTTRR	-	Record No.	Relative Track and Record (zoned decimal)
TTTTTTRR	-	Record Key	
TTTTTTRR	Key	Record Key	
MBBCCHHR	-	Record No.	Actual Disk Address
MBBCCHHR	-	Record Key	

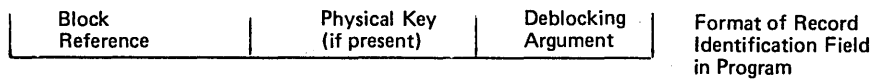


Figure 2.3–1. DAM Data Set Record Location

Initially, this appears to restrict the size of the data set. However, the relative address is maintained in a fullword, enabling a

data set to be maintained, containing 2 raised to the power of 32 bytes. This is equivalent to a data set of approximately 43 billion bytes, extending over more than forty-three 3330 disk drives.

Records which are written to an entry-sequenced VSAM data set are never moved until the data set is reorganized. Any additions to the data set are made at the end of the data set, and the relative byte address by which that added record may be subsequently retrieved is returned to the application program.

Using the relative byte address for record identification of a VSAM data set provides the following advantages:

- Operates equally well with fixed-length or variable-length records
- Provides rapid file access, with full rotational position sensing support even when variable-length records are utilized

The record identification provided by an application program to access a VSAM data set by RBA is a four-byte relative byte address. This may be calculated using techniques similar to that used to calculate a relative record number or relative block number for DAM data sets, or the relative byte address may be stored as a pointer in logically related records.

Indirect Access

CICS/VS file control enables data bases to be constructed. This is achieved by the use of the indirect access feature of file control, enabling various data sets to be constructed to identify logically related records in other data sets. The indirect access feature utilizes pointers from a record in the data set to logically related records in other data sets. The pointers can contain the actual disk address of a logically related record, the relative location of that record in its data set, or the key of that record. This enables identification and retrieval of information logically related to the record being processed.

Indirect access is supported, at the CICS/VS command level interface, for VSAM data sets only. Support for other access methods is provided at the macro level; it is not described in this manual, but further information is given in the Application Programmer's Reference Manual (Macro Level).

INDIRECT ACCESS APPLICATION EXAMPLES

Figure 2.3-2 illustrates a product record in a product data set and the supplier of that product through a supplier number. This supplier number is used as a pointer to access a separate supplier data set to obtain further information about the supplier of the product in question. The supplier number in the product record becomes a pointer to the supplier data set and can be indirectly accessed from the product data set.

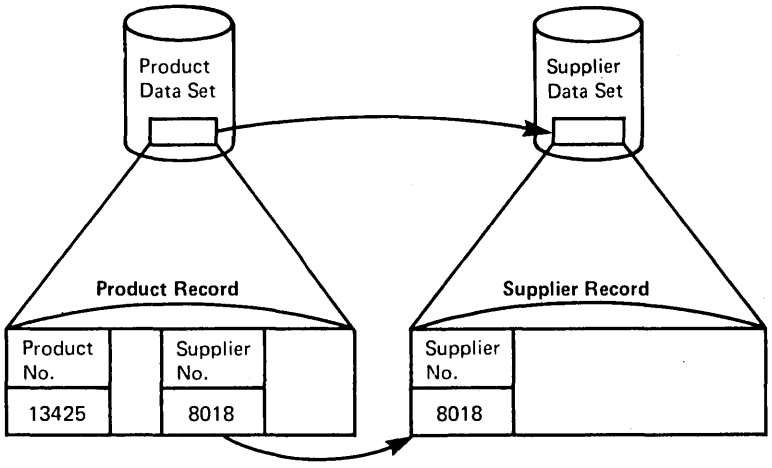


Figure 2.3-2. Product Data Set Indirect Access

Another example of indirect access is in an insurance policy information system. In this case, a policy record in a policy data set contains information relating to the policyholder (for example, customer number or name). If the customer data set is organized in customer name sequence, the name may be used as a key to retrieve the customer record relating to that particular policy. In this way, the customer name in the policy record is used as a pointer for indirect access to further customer details in the customer data set (see Figure 2.3-3).

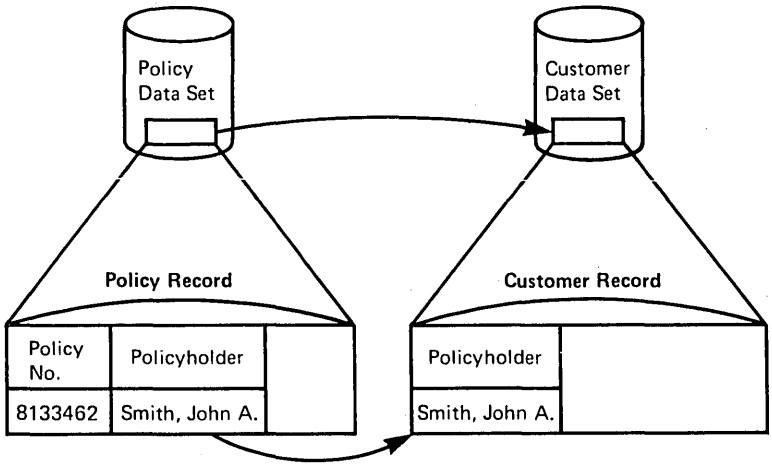


Figure 2.3-3. Policy Data Set Indirect Access

An indirectly accessed data set may also contain pointers to other logically related records in other data sets. The customer record, indirectly accessed from the policy record, may in turn have a field which identifies that customer's insurance agent. The identification of this agent (agent number) may be used to access the related agent record in an agent data set (see Figure 2.3-4).

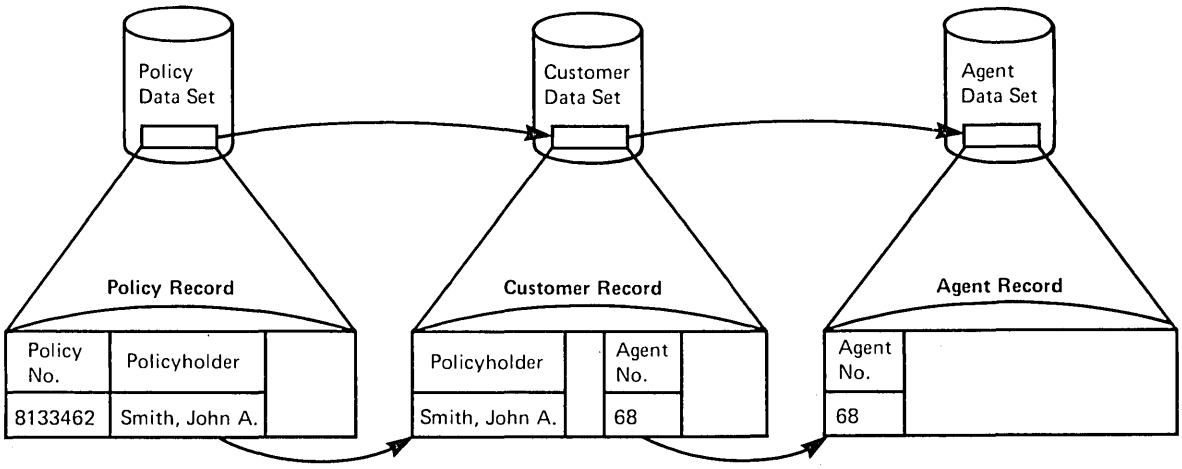


Figure 2.3-4. Indirect Access to Insurance Agent Data Set

Indirect Access Implementation

The CICS/VS file control indirect access feature enables fields in a record to be utilized as pointers to logically related records in other data sets. There is no limit to the number of indirect accesses to other data sets which may be made through the use of these pointers. Depending upon the type of data set, the pointer will be either a record location or a record key.

The data set which contains a pointer field to a logically related record in another data set is referred to as the index data set. The logically related data set is referred to as the object data set. An index data set may utilize several fields in a record to point to logically related records in several object data sets. These indirectly accessed object data sets may in turn utilize a field in their records to point to logically related records in other data sets. These original object data sets become index data sets for the next level of indirectly accessed data set.

INDIRECT ACCESS INITIATION

Indirect access enables a chain to be constructed through logically related records in many different data sets. Figure 2.3-5 illustrates a parts data set, which may be organized in part name sequence. This data set is accessed by means of a part name. The part name record is utilized as an index to a part number record in the parts data set. This part number record may in turn contain a supplier number utilized as a pointer to the supplier data set. In turn, the supplier record may contain a relative block address (RBA) pointing to an associated accounts payable record for that supplier.

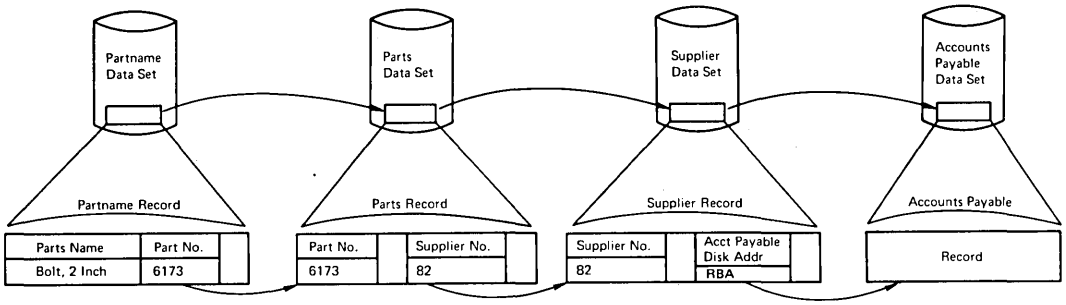


Figure 2.3-5. Indirect Access Chain in a Parts Data Base

To initiate an indirect access retrieval, the application program issues a READ command indicating the name of the data set from which a logically related record is to be retrieved. In the example illustrated in Figure 2.3-5, the application program may provide a part name key and specify that a record is to be retrieved from the supplier data set, to obtain further information about the supplier of that part name. This is shown in Figure 2.3-6.

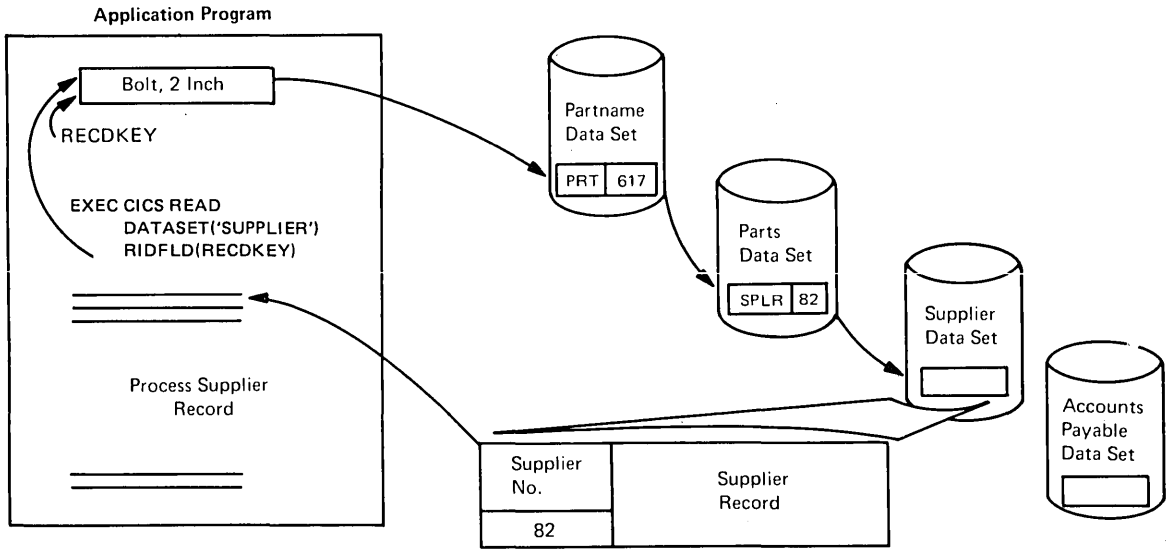


Figure 2.3-6. Indirect Access Operation

CICS/VS file control automatically retrieves the part name record for the part name key provided by the program, extracts the part number, and uses it as a key to retrieve the part number record from the parts data set. Also, the supplier number is extracted from that parts record and is used as a key by file control to retrieve the related supplier record from the supplier data set. This supplier record is the record

requested by the application program and is returned to it for processing.

In one READ request, CICS/VS file control follows the necessary indirect access chain, accessing as many data sets as required, to retrieve the record requested and present it to the application program. However, the application program is not aware of the number of data sets indirectly accessed. It appears to the program as if the supplier data set in the above example is in fact organized in part name sequence, rather than in supplier number sequence.

By following an indirect access chain in this way, file control must be aware of the logical relationship between data sets. This is achieved at system generation when the file control table (FCT) is generated.

SPECIFICATION OF INDIRECT ACCESS LOGICAL RELATIONSHIPS

As characteristics of each data set are specified in the FCT entry for that data set during FCT generation, an indirect access relationship between that data set and another data set may be specified. Information required to define an indirect access relationship is:

- Location of the field in the record to be used as a pointer to the indirectly accessed data set
- Length of that field or pointer
- Name of the object data set

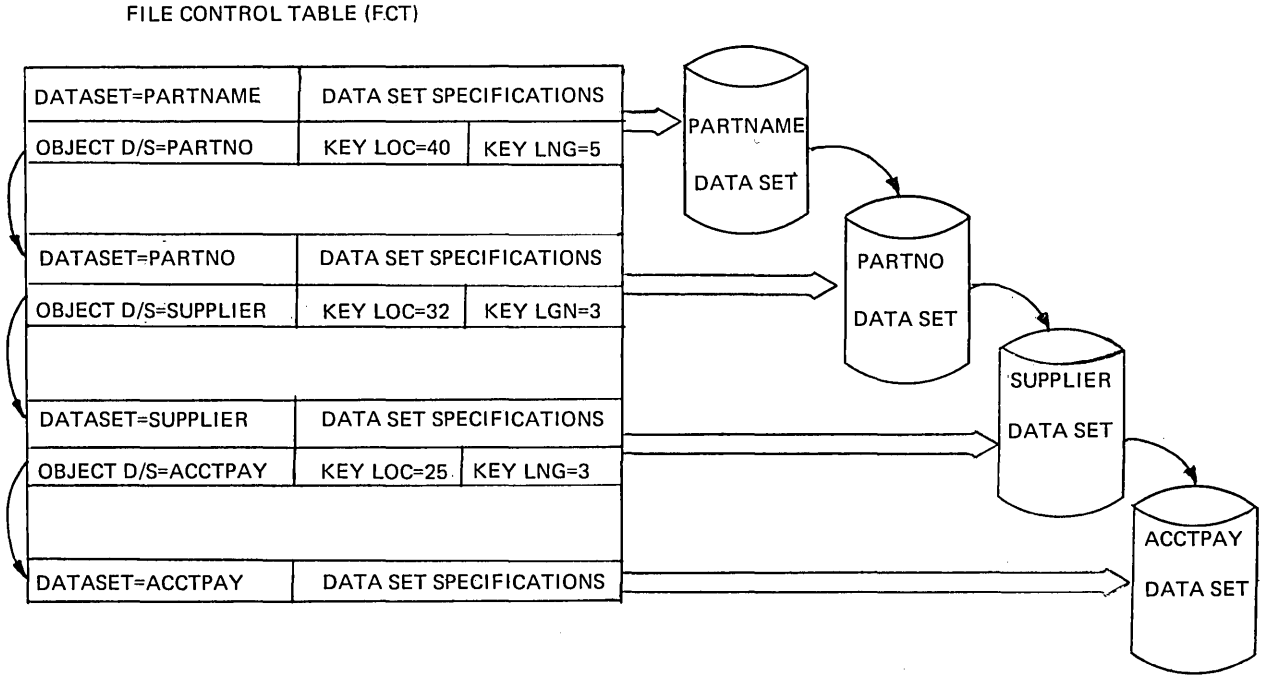


Figure 2.3-7. Specification of Indirect Access Logical Relationships

All fields in the record which are to be utilized as pointers to indirectly accessed data sets are identified, together with the data sets to which they refer, as shown in Figure 2.3-7. This information defines the data set in question as an index data set and identifies the indirectly accessed data set as an object data set. These data sets, when they are subsequently defined in the FCT, are also identified as index data sets which refer to other object data sets. This is done by defining the record fields which are to be used as pointers in those data sets, and the names of the object data sets to which they refer.

A chain of logically related data sets is defined in the FCT during system generation. When an application program requests indirect access retrieval, file control identifies a chain on which the object data is located. It then retrieves each related record in the data sets on the identified chain in the sequence specified by the FCT, until the related record in the object data set is retrieved. It is then presented to the application program for processing.

Indirect access retrieval enables data bases to be constructed utilizing logically related information in a number of data sets. One READ command causes all the required indirect accesses to be carried out until the requested record is retrieved for presentation to the task. This indirect accessing is carried out asynchronously, enabling other concurrently executing tasks to continue processing.

UPDATING INDIRECTLY ACCESSED RECORDS

Indirect access retrieval may be carried out with the intention of subsequently updating the object data set record, if required. This is indicated by the application program specifying that this indirect access retrieval is also part of an update operation. When the object record is retrieved, exclusive control is placed on the VSAM control interval. The application program issues either a REWRITE command to write the updated record back, or an UNLOCK command to indicate that the record is not to be updated, but that exclusive control is to be released.

DUPLICATES DATA SET

In following a direct access chain through several data sets, the pointer field in an index data set record can identify a number of separate records in its relevant object data set. As shown in Figure 2.3-4, a policy record identifies the policyholder by name. The customer data set in this case may be organized in customer name sequence, with the name used as a key to access relevant records. However, there may be several customers with the same name, such as Jones. To develop a unique key for each policyholder with the name of Jones, additional information covering first and second names, birth date, or address must be added to the key. However, adding extra information to the record key reduces the amount of disk storage available, and wastes disk storage when additional identifying information is not used.

To overcome this problem, CICS/VS file control provides an additional capability with the indirect access feature, to enable duplicate records to be identified. This is achieved by utilizing the first byte of a pointer field in an index data set record. This first byte contains a unique code which cannot otherwise occur as part of the key. In the case of customer Jones, the first byte of the customer name field in the

policy record can contain a unique code, for example, hexadecimal FF, as shown in Figure 2.3-8. This is immediately followed by the customer name (Jones in this case). The hexadecimal code FF identifies this as a pointer to several records with the same key. The key is utilized to access a separate duplicates data set, rather than the normal object data set. In this example, the key Jones is used to access a "duplicate customer" data set that contains one record with the name key of Jones. This duplicate record may in turn contain information enabling further identification of customers with the name Jones.

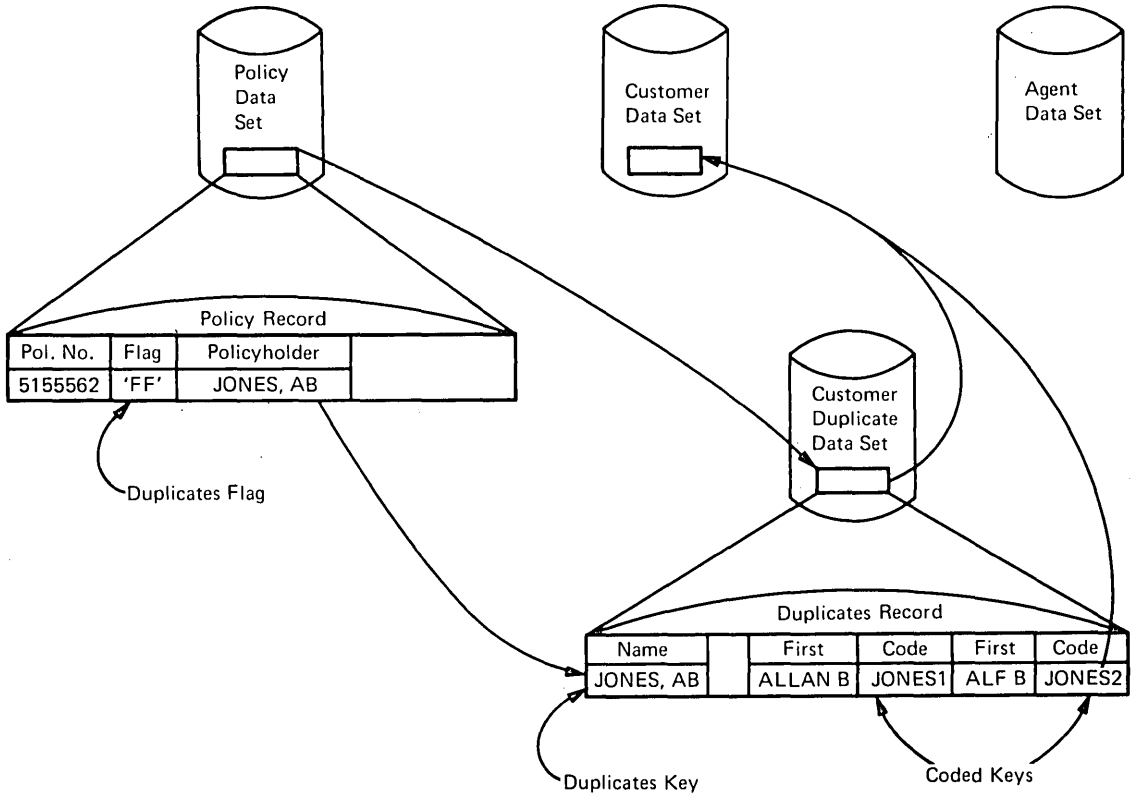


Figure 2.3-8. Duplicates Data Set for Indirect Access

DUPLICATES DATA SET IMPLEMENTATION

The definition of a duplicates data set associated with a particular index data set is specified in the FCT as part of the index data set FCT entry. This indirect access FCT entry identifies the location and length of the pointer field in the index record and the name of the object data set. In addition, the user-specified duplicates code in the first byte of the pointer is defined, together with the name of the duplicates data set.

Even though an indirect access chain is followed through several data sets, the first byte of the pointer field to be used in a record is examined by file control to determine whether it is a duplicates code defined for that index data set. If it is, the duplicates data set is accessed using that pointer, instead of the defined object data set for

that index record. The indirect access chain is broken at that point, and the duplicates record is returned to a user-supplied duplicates routine in the application program for further processing, instead of returning the requested object data set record. The function of the user-supplied duplicates routine is to examine the information presented in that duplicates record, and uniquely identify the pointer to be used to retrieve the next record in the indirect access chain. This identification can be made either by the application program itself, or by a request for further information from the terminal operator if necessary (see Figure 2.3-8).

The duplicates data set feature becomes a powerful capability, enabling unique record identification problems to be resolved so that an indirect access chain can be maintained through the various logically related data sets.

ADDITIONS TO INDIRECT ACCESS DATA SETS

Records can be retrieved using the indirect access data set, either for processing only (READ), or for subsequent update (READ with UPDATE). The updated object record can be written back with a REWRITE command, or ignored by issuing an UNLOCK command.

However, when adding new records to data sets which are indirectly accessed, care must be exercised to ensure that the indirect access chains are correctly maintained.

Records can be added to data sets which are part of an indirect chain, using the procedures described in the topic "Direct Access." When constructing new records, the indirect access pointer fields must be correctly positioned, and must contain the correct information for record identification to access the object data set referenced by that particular index pointer.

The order in which new records are added to indirect data sets is significant, depending upon whether a record key or a record location pointer is utilized. If all the indirect access pointers are record key pointers (for example, to be used for accessing key-sequenced VSAM data sets), the order in which additions should be made to the indirect data sets is not particularly significant. However, if some or all of the pointers are record location fields for use with VSAM data sets, the order becomes significant, because the actual locations of the added records are not known until the addition is completed.

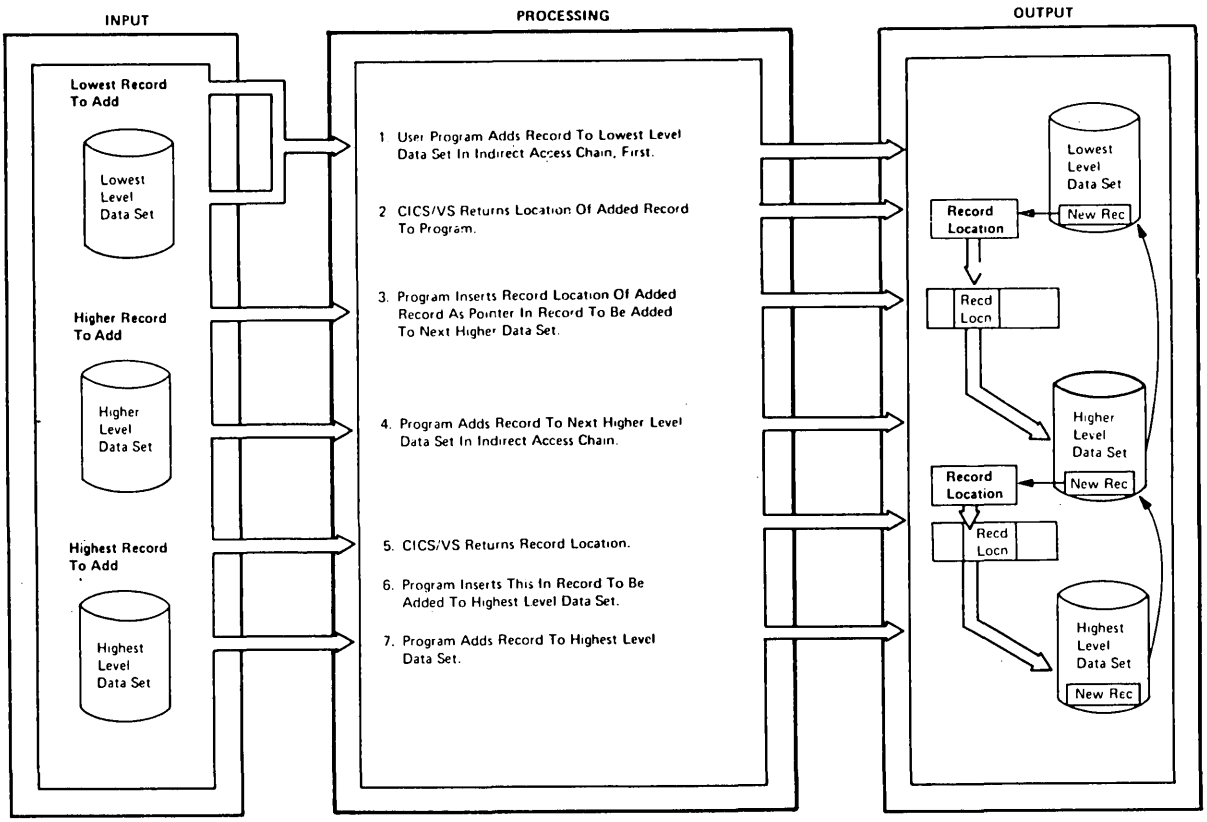


Figure 2.3-9. Addition of Records to Indirect Accessed Data Base

Additions to an entry-sequenced VSAM data set are made at the end of the data set, as discussed under the topic "Direct Access." The record location (RBA) of the added record is returned to the application program. This RBA may be utilized as a pointer from an index record to that new added record. Figure 2.3-9 shows how indirect access chains for entry-sequenced VSAM data sets may be built up. Records are added at the lowest indirect access level first, and the RBA pointers returned to the application program are utilized as indirect access pointers in the next higher indirect access level. This progresses upward through successively higher indirect access levels until the highest indirect access level record has been added to its associated data set.

INDIRECT ACCESS CHAIN INTEGRITY

Special consideration should be given to the possibility of system failure during the addition of an indirect access chain. If a system failure occurs before all the logically related records are added to their relevant data sets, an incomplete direct access chain will result. Accordingly, techniques discussed in Part 5 should be utilized for journaling any indirect access additions, to enable incomplete indirect access chains to be backed out on restart if necessary.

Normally, new indirect access chain records should be added offline to the data sets. However, adding these indirect access record chains online using the preceding technique is required to reduce the

vulnerability of those data sets to system failure, and it is advisable to ensure that only one direct access chain be added at a time. This may be achieved by all application programs that generate new indirect access chains enqueuing on the same single user resource prior to commencing an indirect access chain addition. On successful completion of the entire indirect access chain addition, the application program can dequeue itself from the single user resource. By utilizing CICS/VS ENQ and DEQ commands, only one task at a time will be able to carry out an indirect access chain addition. While this may have an effect on online performance, depending upon the frequency of adding new chains, it will result in a definite improvement in the integrity and safety of the various data sets in the event of a system failure.

Segmented Records

The CICS/VS file control segmented record feature enables data sets to be constructed for efficient use on disk and dynamic storage space, including those data sets containing a considerable amount of variable-length information and which have various fields that are either present or absent in specific records.

The segmented record feature considers a record to be comprised of a number of segments - each segment containing one or more related fields. For example, in a customer data set (see Figure 2.3-10), the customer name may be defined as one segment and each address line as another segment. A number of customer account history fields containing the balance outstanding (current, one month, two months, three months, and over three months) may comprise another single segment.

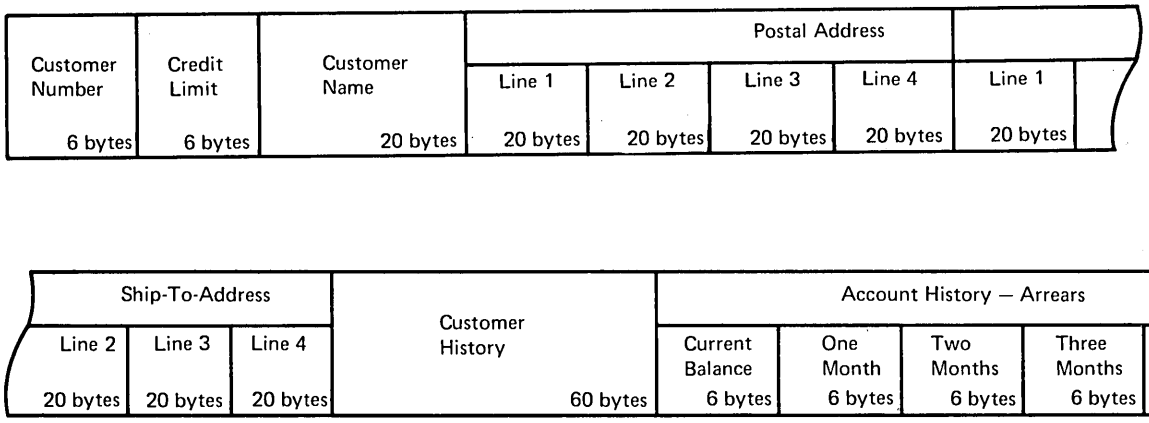


Figure 2.3-10. Typical Customer Record Format

Another example of segmented records is shown in Figure 2.3-11, which illustrates a savings account data set used in the banking industry. The account master information may be defined as one segment. Each deposit or withdrawal transaction made previously against the account may also be defined as a segment.

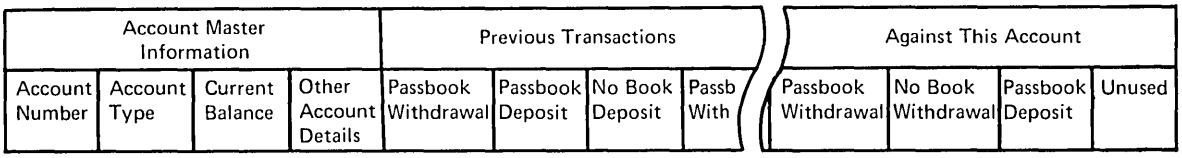


Figure 2.3-11. Typical Savings Account Record Format

SEGMENT DESIGN

Generally, fields are grouped within one segment if they have some similar relationship, such as similar information which will be operated upon together by various programs, or fields which are either all present or all absent for a particular record, or single fields which may contain variable-length information such as name or address lines.

A segment may contain any number of fields up to a total segment length of 255 bytes. Segments may be further defined as either fixed-length segments, or variable-length segments whose length is indicated by a one-byte binary field at the start of the segment.

Consider the storage of a customer name and address in the record format shown in Figure 2.3-10. How many bytes should be allocated for the name field? Should each name field be allocated 20 bytes or 30 bytes or 15 bytes? Depending upon the characteristics of various customer names, a 30-byte name field could contain every possible name, whereas a 20-byte name field may contain 95% of customer names, and a 15-byte name field only 80% of names. Using the 30-byte name field will avoid the necessity of abbreviating or reducing the lengths of names, as would be necessary using 20-byte or 15-byte fields. However, the great majority of names may be less than 15 bytes. In this case, 15 bytes or more of disk space in each customer record is wasted if a 30-byte name field is allocated.

How many bytes should be allocated to each address line? Allocation of 30 bytes for each address line may enable every possible address line to be stored in full, while 20-byte or 15-byte address lines may require some form of abbreviation. Again, the use of 30-byte address line fields may result in wasting approximately 15 bytes or more per address line, because the majority of address lines may fit within 15 bytes.

Another consideration is the number of address lines to allocate for a customer record. Many customer addresses have two or three address lines, while some may require six or more address lines. Should six address lines be allocated for each customer record? Should instead three address lines be allocated, reducing longer addresses to three lines, but wasting an address line field in the case of a two-line address?

The segmented record feature enables record format definition problems such as these to be easily resolved. The customer name and each customer address line may be defined as separate segments. Furthermore, they may be defined as variable-length segments such that for one additional length byte at the start of each segment, the exact length of that segment can be indicated. Thus, a five-character

customer name would occupy five bytes plus one byte for the length indication — a total of six bytes. However, a 25-byte customer name would occupy a total of 25 plus one or 26 bytes, while a 14-character name would occupy a total of 15 bytes. Only the amount of storage required for each individual name need be allocated, for more efficient disk storage utilization.

Figure 2.3-12 shows that each address line segment may be defined as variable-length, with the actual length of each address line occupying only that many bytes, together with an additional byte per address line as a length indication. Names and addresses need not be abbreviated, but may occupy as little or as much disk storage as required.

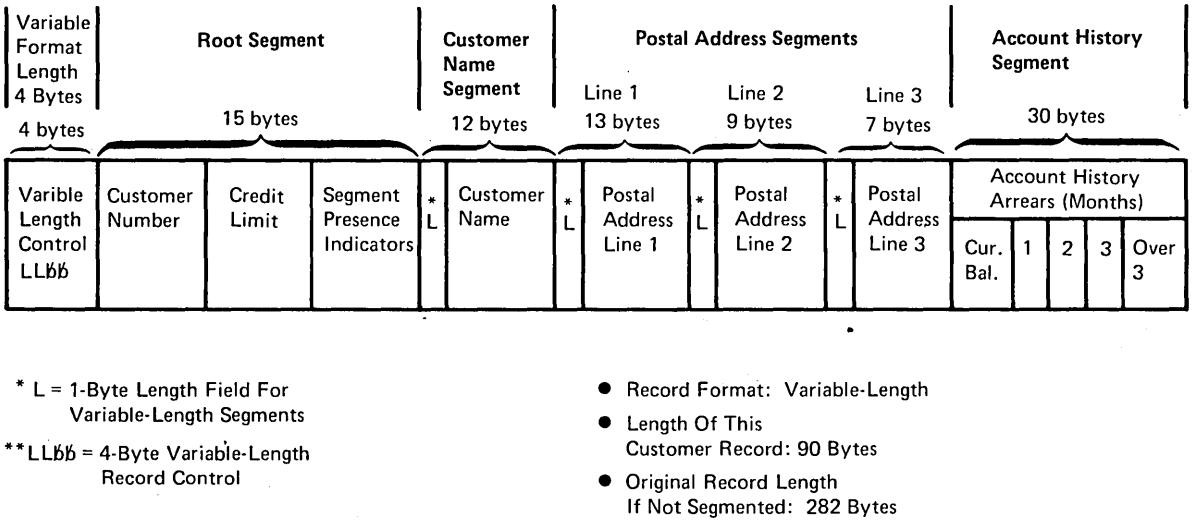


Figure 2.3-12. Segmented Customer Record Format

PRESENCE OR ABSENCE OF SEGMENTS

A further advantage of variable-length segments is that segments may be defined as being either present or absent on an individual record basis. In Figure 2.3-11, eight address line segments are defined for each customer record. However, if only three address lines are required, only three address line segments need be present for that record (see Figure 2.3-12). The remaining five possible address line segments are not present, and do not occupy any disk space.

ROOT SEGMENT

Fields which are always present in every record, such as a customer number, credit rating, and other required information for each record, are generally grouped together into one segment. This is called the root segment and precedes all other segments in the record. This segment is always present in a segmented record.

SEGMENT INDICATOR FLAGS

The presence or absence of other segments in the record is indicated by segment indicator flags. These indicator flags are contained within the root segment, and may comprise either bit indicators or halfword indicators.

Bit indicators use a separate bit to indicate the presence or absence of each specific segment. If the segment associated with a bit is present, that bit is turned on. However if the segment associated with the bit is absent, that bit is turned off.

Halfword indicators utilize two bytes to indicate the presence or absence of each segment. The two bytes are used as a zero/nonzero switch. If the two bytes are nonzero, the associated segment is present; if the two bytes are binary zero, the associated segment is absent.

COBOL programs must use halfword indicators.

Figure 2.3-18 illustrates the use of segment indicator flags, and relates to the customer record shown in Figure 2.3-12.

SEGMENT DEFINITIONS IN FCT

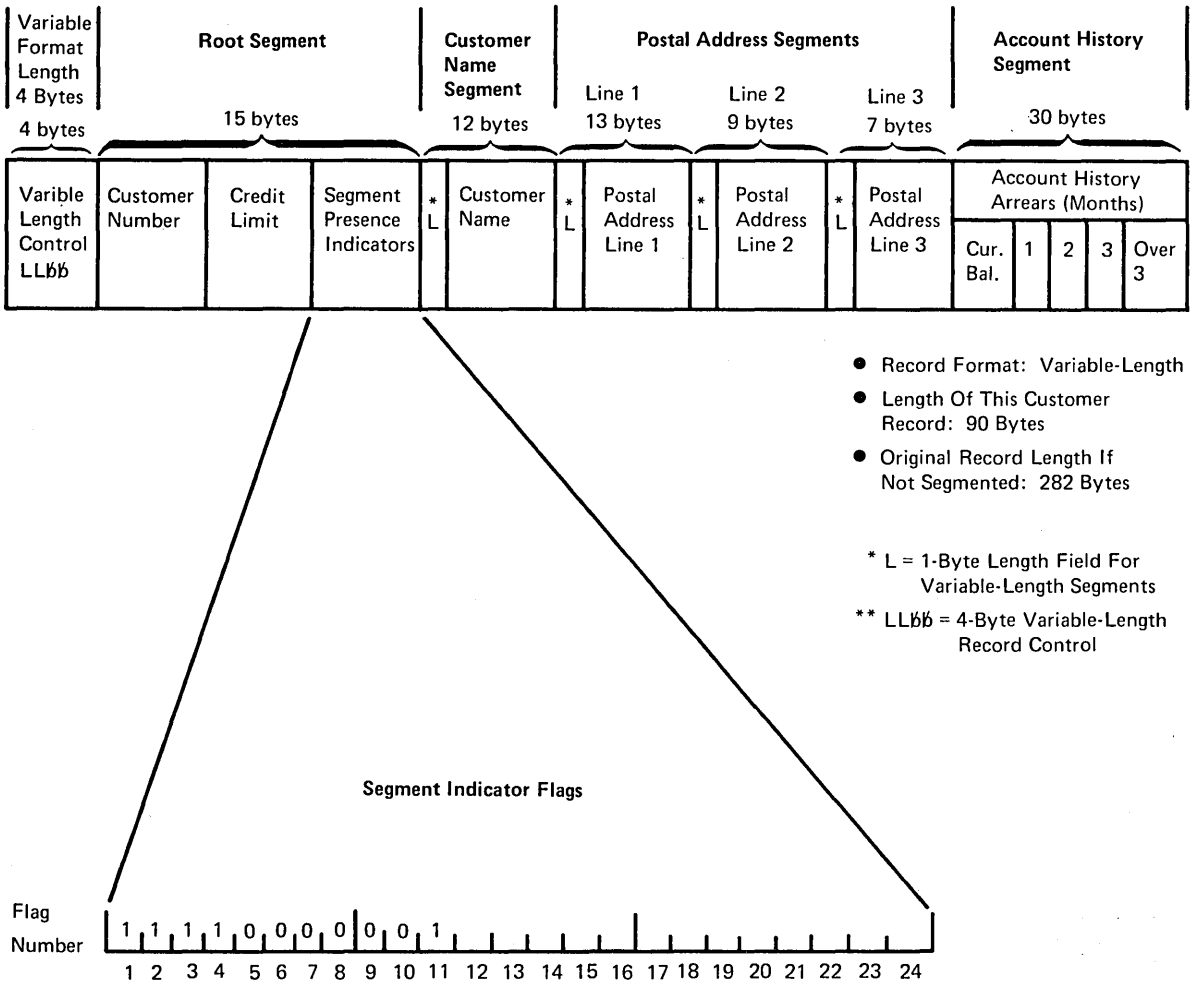
While it is the responsibility of the system programmer to specify segments in the FCT, it is important that the system designer have a general understanding of how segments are defined, to better understand the operation of the CICS/VS segmented record feature. This will enable him to take advantage of facilities offered by this feature when initially designing the various data bases which will be utilized by the application.

Either bit indicators or halfword indicators may be used with a particular segmented record data set, but not both. The selected type of indicators is specified in the file control table (FCT) entry for that segmented record data set. As many bytes as required to contain all of the necessary segment indicator flags (up to a maximum of 99 segment flags) must be defined within the root segment. The starting location, and length in bytes, of the segment indicator flags field in the root segment is also defined in the FCT entry for a segmented record data set. See Figure 2.3-13.

Each separate segment is then defined in the FCT entry (see Figure 2.3-14). Every segment is allocated a segment name up to 8 characters long, and is specified as a fixed-length segment or a variable-length segment of a defined maximum length. In addition, any required boundary alignment (byte, halfword, fullword, or doubleword) necessary when the segment is read into storage is identified. As each segment is presented to the application program for processing, the segment is aligned on the specified boundary. However, the extra bytes required to ensure specific boundary alignment are not recorded on disk; they are inserted when the record has been read from disk and before the segment is passed to the application program.

An application program may utilize several segments in a record for processing. These segments may be grouped together and called a segment set. By identifying a segment set by name, all the segments comprising that segment set are also identified.

Consider a customer data set in the utilities industry, as shown in Figure 2.3-15. An application program which requires access only to the customer name and address may request a segment set comprised only of the customer name segment and each of the postal address line segments. This may be uniquely identified as a segment set which is given a specific name, such as NAMEADDR.



- Segment Indicator Flags Indicate The Presence Or Absence Of Segments.
- 1 Bit or 2 Bytes May Be Used To Indicate The Presence Or Absence Of Each Segment. Here, Bit Indicators Are Used.
- Flag 1 Refers To The First Segment After The Root, That Is, Customer Name. Flag 2 Refers To Postal Address Line 1, Flag 3 Is Postal Address Line 2, And So On, Up To Flag 11, Which Is Account History.
- A Bit ON Indicates The Relevant Segment Is Present; A Bit OFF Indicates The Segment Is Absent.
- The Example Here Shows That Customer Name, A 3-Line Postal Address, And Account History Are Present In This Particular Customer Record.
- Bits 12 Through 26, In This Example, Are Not Used. They Have Been Allocated To Enable Another 13 Segments To Be Added To This Record Later, For Other Applications, Without Necessitating Any Modification To Existing Application Programs.

Figure 2.3-13. Segment Indicator Flags

6 bytes	6 bytes	20 bytes					60 bytes				6 bytes				
			Postal Address					Ship-To-Address				Account History – Arrears				
Customer Number	Credit Limit	Customer Name	Line 1	Line 2	Line 3	Line 4	Line 1	Line 2	Line 3	Line 4	Customer History	Current Balance	One Month	Two Months	Three Months	Over Three Months

Data Set = Customer

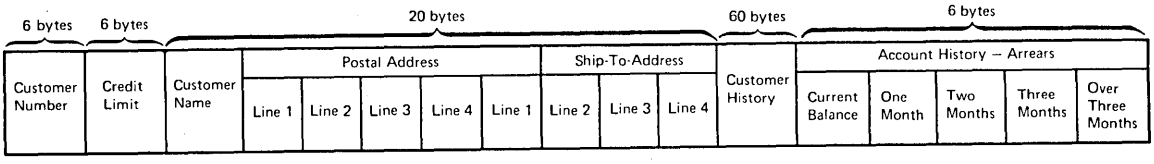
- Record Format: Fixed-Length
- Record Length: 282 Bytes

File Control Table (FCT)

DATASET=CUSTOMER	DATA SET SPECS			
SEGMENT=ROOT	LENGTH=15	BITINDICS	START=12	LNG=3
SEGMENT=NAME	VARIABLE	20 BYTES (MAX)		BYTE ALIGN
SEGMENT=ADDR1	"	20 "		"
SEGMENT=ADDR2	"	20 "		"
SEGMENT=ADDR3	"	20 "		"
SEGMENT=ADDR4	"	20 "		"
SEGMENT=SHIP1	"	20 "		"
SEGMENT=SHIP2	"	20 "		"
SEGMENT=SHIP3	"	20 "		"
SEGMENT=SHIP4	VARIABLE	20 "		BYTE ALIGN
SEGMENT=HISTORY	FIXED	60 BYTES		WORD ALIGN
SEGMENT=ARREARS	FIXED	60 BYTES		WORD ALIGN

Segment Definitions For Dataset = Customer (See Record Format Above)

Figure 2.3–14. Segment Definition in FCT



Dataset = Customer

- Record Format: Fixed-Length
- Record Length: 282 Bytes

File Control Table (FCT)

DATASET=CUSTOMER	DATA SET SPECS			
SEGMENT=ROOT	LENGTH=15	BITINDICS	START=12	LNG=3
SEGMENT=NAME	VARIABLE	20 BYTES (MAX)		BYTE ALIGN
SEGMENT=ADDR1	"	20 "		"
SEGMENT=ADDR2	"	20 "		"
SEGMENT=ADDR3	"	20 "		"
SEGMENT=ADDR4	"	20 "		"
SEGMENT=SHIP1	"	20 "		"
SEGMENT=SHIP2	"	20 "		"
SEGMENT=SHIP3	"	20 "		"
SEGMENT=SHIP4	VARIABLE	20 "		BYTE ALIGN
SEGMENT=HISTORY	FIXED	60 BYTES		WORD ALIGN
SEGMENT=ARREARS	FIXED	60 BYTES		WORD ALIGN
SEGSET=NAMEADDR (ROOT) NAME ADDR1 ADDR2 ADDR3 ADDR4				
SEGSET=ACCOUNT (ROOT) NAME ARREARS				

Segment Definitions For Dataset = Customer

Segment Set Definitions

Figure 2.3–15. Segment Set Definition in FCT

Another application program may require access only to the customer name and account history segments in Figure 2.3–15. These segments may be defined in a separate segment set which may be given a unique name such as ACCOUNT. The application program in this way indicates that it is sensitive only to segments in the specified segment set.

SEGMENT RETRIEVAL

When the application program wishes to retrieve a specified segment set from a segmented record, it provides information for the identification of that record, such as a record key or record location field, identifies the data set to be accessed by name, and then identifies the segment set by name to be presented to the application program (see Figure 2.3-16).

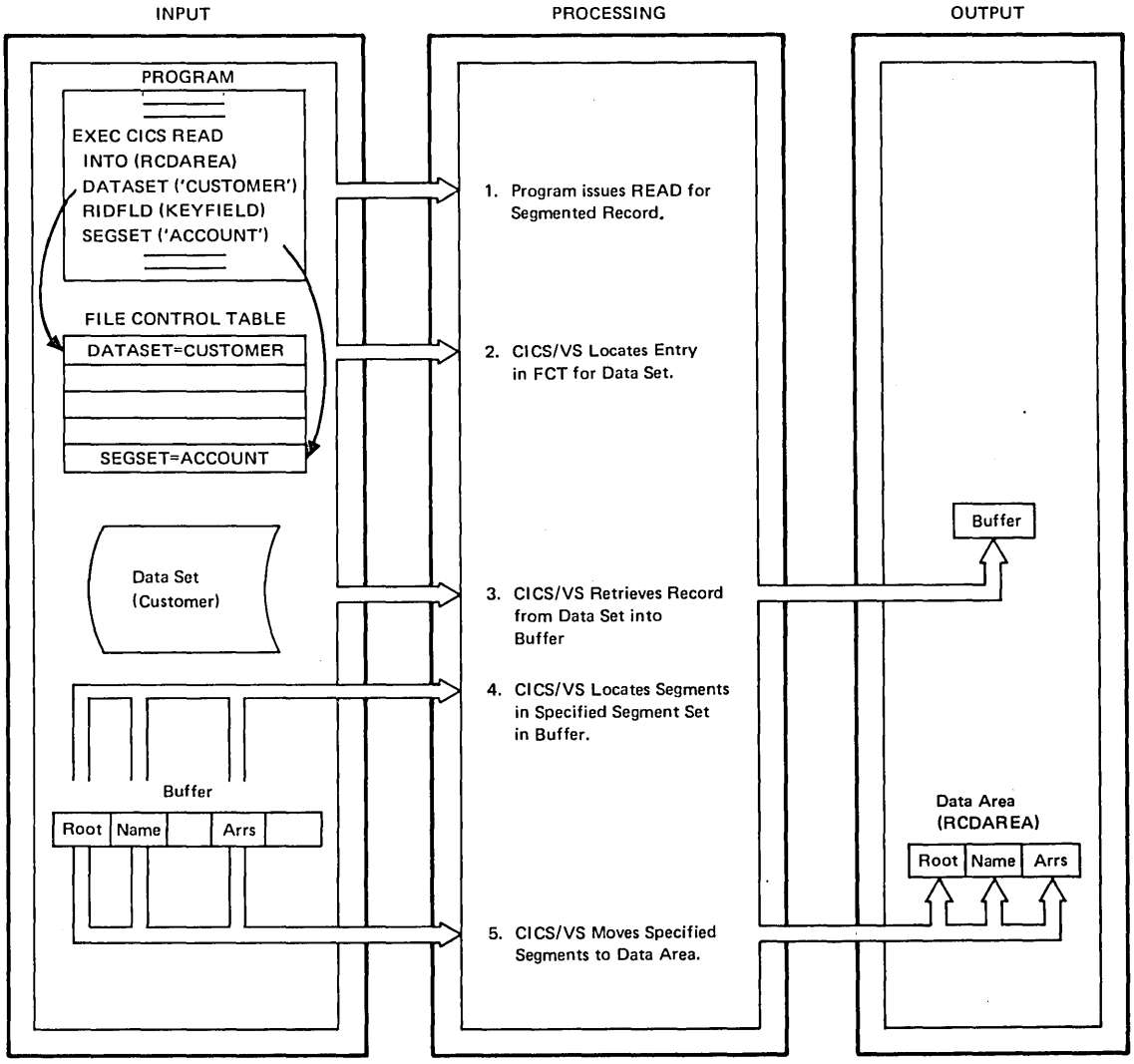


Figure 2.3-16. Segment Retrieval

File control uses the record identification field to access the required record from the specified data set. The segment set name is then used to identify the particular segments making up that set. Each of the segment names in the segment set is defined in the FCT entry as being fixed-length or variable-length, of a specified maximum length, and requiring specified boundary alignment. Segments are assumed to be in the same sequence in the record as the sequence in which they are defined in the FCT.

The segments in the segment set are extracted from the record by file control. These segments are presented to the application program together with the root segment, as if only those segments were present in the record on disk (see Figure 2.3-18). Other segments which are not

part of the segment set are ignored, and are not presented to the application program.

The presence or absence of segments in the record passed to the program is indicated by the status of the segment indicator flags (see Figure 2.3-13). These segment indicator flags are located within the root segment, and each flag is tested by the application program to determine whether it is on or off, and hence whether its associated segment is present or absent. The first segment indicator flag (which may be a bit flag, for example) indicates the presence or absence of the first segment in the record following the root segment. The second indicator flag specifies presence or absence of the second segment in the record following the root segment, and so on. These indicator flags specify the presence or absence of all segments within the record, not only those segments in the segment set passed to the program.

The use of DSECTs or structures in application programs can simplify the testing and manipulation of segment indicator flags. If the appropriate DSECT or structure defining all indicators is copied into the application program when it is compiled, the use of installation-controlled labels for each indicator may be achieved. The application programs may then be less sensitive to changes in segments, a segment change being made to the DSECT or structure and the programs then recompiled.

SEGMENT UPDATING (KEY-SEQUENCED VSAM)

Application programs may replace segments, delete segments, or add new segments. However, the extent to which new segments may be added depends upon the particular access method used for the data set. If DAM, ISAM, or entry-sequenced VSAM data sets are used for the segmented record data set, the total segmented record length may not be increased, but may be reduced if required. (An exception is with OS/VS variable-length ISAM records, where the segmented record length may be increased if necessary.) With key-sequenced VSAM data sets, the segmented record length may be increased, either through the addition of new segments to the record or by a change in the length of existing segments. The increased record length is regarded by file control as a replacement of the previous segmented record and is handled in much the same way as the addition of a new record to the data set. That is, records which follow the lengthened record in the same control interval are shifted to the right to enable the increased-length record to be inserted.

If it is necessary to increase the length of DAM, ISAM, or entry-sequenced VSAM segmented records, a different technique, segment updating, must be utilized.

SEGMENT UPDATING (DAM, ISAM, AND ENTRY-SEQUENCED VSAM)

When it is determined that a segmented record in a DAM, ISAM, or entry-sequenced VSAM data set will be increased in length because of the addition of a new segment, or the increase in length of an existing segment, the original segmented record retrieved must be updated by the application program to indicate that that record is now obsolete. This may be specified by the program placing a logical delete flag in the root segment as shown in Figure 2.3-17.

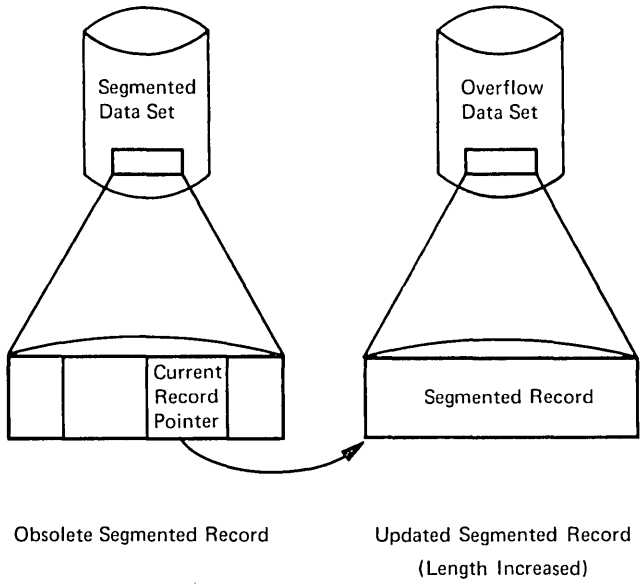


Figure 2.3-17. Segment Updating, with Length Increase in DAM, ISAM, and Entry-Sequenced VSAM Data Sets

The increased-length record may be written by the user program to a separate user-defined "overflow" data set. The location of that increased-length record in the overflow data set may be inserted as a record pointer field in the root segment of the original record (see Figure 2.3-17). The logical delete flag in the root segment may be utilized as the record pointer field. If this record pointer field is zero, it indicates that the segmented record is current. However, if the record pointer field is nonzero, it indicates that the segmented record has been logically deleted and replaced by another record in the overflow data set. The record pointer may be utilized by the user program as a record key, or record location pointer (depending upon the particular data set it points to). It directs the application program to the new current segment in the overflow data set.

With this technique, an increase in length of a segmented record may be logically supported using DAM, ISAM, or entry-sequenced VSAM data sets. The increased-length record is first written by the user program to the overflow data set. The record location field returned by CICS/VS as a result of that addition, or the record key for ISAM, is then inserted into the record pointer field in the root segment of the record in the segmented data set (See Figure 2.3-17).

On subsequent retrieval, the application program first tests the record pointer field in the root segment. If the pointer field is nonzero, the application program uses that pointer as a record identification to the more current segmented record in the overflow data set, and retrieves that current segmented record for processing. The overflow data set must be specified during FCT generation as having the same segments, and segment sets, as the original segmented data set.

If a segment is to be updated with no change in length, that update is effected and the segments are presented by the program to CICS/VS to be written back to the data set by means of a REWRITE command. If the length of a variable-length segment is to be reduced, the length byte is modified to reflect the new length of the updated segment. This segment is then written back to the data set by issuing a REWRITE command. File

control blocks up each segment in the segmented record, removes any boundary alignment bytes, and uses the length byte in variable-length segments to allocate a sufficient number of bytes to contain contents of that segment.

To allow for a potential increase in length after a READ with UPDATE command of variable-length segments, or the addition of a new segment which was missing in the segmented record, CICS/VS file control will present the application program with a segmented record containing space set aside for each segment, whether present or absent, based upon either the fixed length of that segment, or the maximum variable length of that segment.

SEGMENT DELETION

A segment may be deleted if the application program turns off the relevant segment indicator in the root segment. File control then physically deletes that segment when it writes the segmented record back to disk.

FIXED- AND VARIABLE-LENGTH SEGMENTED RECORDS

CICS/VS segmented records may be either fixed-length or variable-length. For fixed-length records, the actual segmented record may be less than the allocated fixed-length record size. Slack bytes are therefore used at the end of the segmented record to make up the specified fixed-length record. Segmented records which are variable in length because of variable-length segments, or the absence of particular segments, may be specified as variable-length records (see Figure 2.3-18). In this case, variable-length records enable more efficient disk storage utilization.

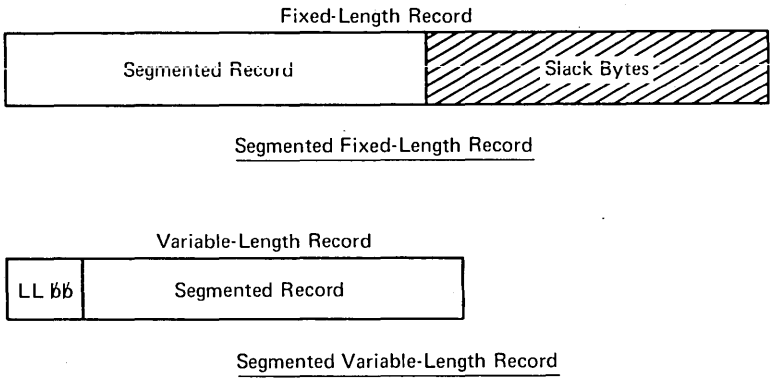


Figure 2.3-18. Segmented Record Disk Utilization

CREATION AND MAINTENANCE OF SEGMENTED RECORDS

CICS/VS file control provides no facilities for the creation or maintenance of segmented records. These records must be created offline, with the various segments (either fixed- or variable-length) blocked by user programs. The offline user program must set the relevant indicator flags, to specify the presence or absence of each segment in that segmented record. Once segmented records have been created offline, they may be retrieved online using the CICS/VS segmented record feature. In the event of addition of new segments or deletion of existing segments online, the online application program is responsible for determining that the appropriate bit or byte indicator flag is on or off, and turning on the appropriate indicator for a segment which has been added online, or turning off the indicator for a segment which is to be deleted online. CICS/VS will examine these indicator flags when the record is to be written back to disk, and take the appropriate action to either delete the segment or increase the segmented record length if necessary to allow for an added segment.

With the availability of the built-in bit manipulation function, Assembler, PL/I, and American National Standard (ANS) COBOL application programs may test bit indicator flags and set bit indicator flags on or off. Consequently, the use of halfword indicator flags for ANS COBOL programs is not necessary, and all indicator flags may be maintained as bit flags.

ADVANTAGES OF SEGMENTED RECORDS

The principal advantages in the use of the segmented record feature are described below.

Disk Storage Utilization

Through the use of variable-length records, variable-length segments, and omitting unused segments, only that amount of disk storage required for storage of the information relevant to each record need be allocated.

Dynamic Storage Efficiency

On an application program READ command, the entire segmented record is read into the a buffer area. The segments making up the segment sets requested by the program are extracted by CICS/VS from the record in the buffer, and are moved into the data area specified in the READ command. At this time, if the application program did not indicate that the record will subsequently be updated and written back, CICS/VS file control releases the buffer storage, and transfers only the requested set of segments to the data area. Consequently, the amount of main storage used in processing those segments is only as large as the segments themselves. No additional dynamic storage is utilized for segments which are not being processed.

Limited Data Independence

Because an application program identifies only that set of segments which are significant to it, the modification of segments in the data set which are not relevant to the program concerned requires no modifications to that program. Limited data independence is thus achieved with a consequent reduction in the amount of program modification following a change to the record format or data set organization.

A further consideration of data independence is that additional segment indicator flags may be allocated in the root segment to allow for the addition of future segments. If bit indicators are used, each additional byte of bit flags will enable up to eight segments to be added to the end of the segmented record. When these segments are eventually added to the records, some reorganization of the data set will of course be necessary. However, application programs which utilize segment sets which do not contain the new added segments are not affected.

CICS/VS File Control Design Considerations

As can be seen from the above discussion of the indirect access feature and segmented record feature of CICS/VS file control, data bases may be constructed. A number of factors should be taken into account in deciding the appropriate data base support technique, either indirect access, or segmented records, or both.

ACCESS TO ONLINE DATA BASE BY OFFLINE PROGRAMS

If data sets utilized online in a data base must also be processed offline, the indirect access feature may be found to be more suitable than segmented records. Use of this feature requires each offline application program to provide its own support for the extraction of segment sets from segmented records, and the insertion of segment sets back into segmented records for updating. Alternatively, the code supporting the segmented record feature in the CICS/VS file control program may be extracted by the user and modified for use by batch programs.

If the batch processing requirements of the installation dictate that standard access method support be used for batch programs accessing online data sets, the indirect access feature rather than the segmented record feature should be used. This enables logically related information to be maintained across a series of different data sets. However, this additional file accessing will have an effect on the performance of the online applications, and consequently on response time. This is a further consideration in the design of a CICS/VS file control data base.

Provided that suitable support can be developed to enable batch application programs to access segmented records in online data sets, the segmented record feature is superior to the indirect access feature when one considers the reduced file accessing necessary and the efficient utilization of disk storage. With one file access, a segmented record with all its related segments may be retrieved. With indirect access, these related segments may have been defined in separate indirectly accessed data sets, each requiring a separate file

access. Furthermore, only the object or target data set is returned to the user.

SEGMENTED RECORDS

A consideration with segmented records is the extent to which additions to segments may be made. If segment additions result in an increase in the segmented record length, then the segmented record data set should ideally be defined as a key-sequenced VSAM data set, or a variable-length OS/VS ISAM data set. Alternatively, DAM, ISAM, or entry-sequenced VSAM data sets may be utilized using the dummy segment and overflow data set techniques described above. However, this will result in less efficient disk storage utilization, more user programming, and more file accesses.

A further consideration is the extent to which information may be added to data set records at a future time. If there is a strong possibility of this occurring, it may be advisable to use the segmented record feature, allocating additional segment indicator flags in the root segment to allow for a specified number of additional future segments. At that time, additional segments may be added, without requiring modification of programs which utilize segment sets not associated with the newly added segments.

MULTIPLE OCCURRENCE OF SEGMENTS

The use of the segmented record feature enables a large number of multiple segments to be stored for a particular logical record within a block, to a maximum of 99 segments in each logical record. The number of multiple occurrences of segments may be further limited by user data set creation and maintenance factors.

REAL STORAGE AVAILABILITY

While the segmented record feature does not provide the capability of the DL/I products, and introduces some difficulties regarding batch program access to segmented records, this segmented record feature may be considered for installations with real storage of less than 144K bytes, which are unable to use the DL/I products because of insufficient real storage.

Recovery Considerations

Recovery of information in the event of system or program failure must be considered in data base design. This subject is discussed in more detail in Part 5, but is introduced here to identify those factors which are relevant to the recovery of CICS/VS file control data sets. The optional journaling feature of CICS/VS permits a record to be automatically logged by file control to the system log and/or to be journaled to a user journal data set. The CICS/DOS/VS Entry Level System does not support automatic logging or automatic journaling.

AUTOMATIC LOGGING

Automatic logging is required if data set backout is to be supported by CICS/VS on emergency restart following an uncontrolled shutdown of the system or on dynamic transaction backout following an individual transaction failure. Automatic logging is specified in the file control table entry of each data set for which backout is to be supported by specifying LOG=YES. (See CICS/VS System Programmer's Reference Manual.) On any update, deletion, or addition of a new record, the "before" image is automatically recorded on the CICS/VS system log if automatic logging is specified. The system log is journal number 1.

AUTOMATIC JOURNALING

Automatic journaling allows the user to specify data set activity to be recorded on any CICS/VS journal. For example, automatic logging records the "before" image of a record to be updated and goes to the system log. Automatic journaling may specify that the "after" image is to be recorded also. Additional journaling activity is identified in the FCT entry for each data set through use of the JREQ parameter, and may be directed to a user journal data set or the system log through use of the FCT JID parameter. This may be desired if a chronological record of all data set activity is to be maintained. It permits implementation of a user-written recovery program to recover a data set from a previous backup copy if an unrecoverable I/O error is detected.

The information recorded on automatic logging, and on automatic journaling, contains the identification of the task which carried out the update, deletion, or addition, the transaction code and terminal identification involved with that task, the time of day, other information required by the CICS/VS journal control program, and an exact image of the record which was updated or deleted, or the record identification of an added record. These records are written to the CICS/VS system log and/or relevant journal data set in chronological sequence.

Part 3. Data Communication Design

Chapter 3.1. Introduction

The Data Communications design has a significant effect on the success or failure of the overall project. It is in this area that the interface between the user and the computer is defined. This definition should be oriented toward satisfying the requirements of the user and the application, while still presenting information to the computer in a suitable form for processing.

Before discussing various Data Communications design approaches, it is important that the reader understand the following CICS/VS support which will aid him in his design:

- Terminal control
- Basic mapping support (BMS)
- Terminal device independence
- Terminal paging
- Message routing

These features are discussed in Chapter 3.2, then, in Chapter 3.3, various communication techniques are described. Chapter 3.4 gives an overview of the relationship between CICS/VS and the system network architecture (SNA) access method, VTAM.

Devices and Access methods

CICS/VS allows communication between the host processor system and a wide range of terminals and subsystems. These include display units typewriter-like terminals, specialist subsystems for particular industries and other processors. Many may be treated as either pre-SNA (system network architecture) terminals or SNA logical units.

The line discipline for the connection may be BSC (bisynchronous communication), start-stop or SDLC (synchronous data link control).

The access method used to manage the connection can be BTAM (basic telecommunications access method), ACF/TCAM (advanced communication function for the telecommunications access method) or ACF/VTAM (advanced communication function for the virtual telecommunications access method). (Note that, in this manual, the term VTAM is used to cover ACF/VTAM or any equivalent SNA access method, including certain releases of ACF/TCAM.) BGAM (basic graphics access method) is supported for certain graphics terminals.

For a complete list of supported combinations of terminal (or subsystem), line discipline and access method, see the CICS/VS General Information Manual.

Chapter 3.2. BMS, Terminal Control and Batch Applications

CICS/VS Terminal Control and BMS

CICS/VS application programs can communicate directly with terminals, using:

- Basic mapping support (BMS) commands
- Terminal control commands

BMS enables application programs to request terminal input or output, using the following commands:

- For input RECEIVE MAP or RECEIVE MAP FROM
- For output SEND TEXT or SEND MAP
- For terminal paging SEND TEXT ACCUM, SEND MAP ACCUM, and SEND PAGE
- For message routing ROUTE

Terminal control enables programs to request additional functions:

- Write data to a terminal (SEND or SEND WAIT)
- Read data from a terminal (RECEIVE)
- Synchronize terminal I/O with processing (WAIT TERMINAL)
- Transmit to the buffer of a banking terminal (SEND CBUFF)
- Test for the presence of a banking passbook (SEND PASSBK)
- Reset a line (ISSUE RESET)
- Disconnect a switched line (ISSUE DISCONNECT)
- Erase and write data to a visual display (SEND ERASE)
- Specify the last output message to a VTAM- or TCAM SNA-supported terminal (SEND LAST)

In addition, the system programmer may use the following DFHTC macro instructions to:

- Change the status of a terminal (CTYPE=STATUS)
- Locate a terminal entry in the TCT (CTYPE=LOCATE)
- Check the results of a previous STATUS or LOCATE request (CTYPE=CHECK)

These macro instructions are described in more detail in the CICS/VS System Programmer's Reference Manual.

The particular communication commands used in programming are not significant to the following discussion of data communication design. However, if BMS is not used, the following facilities cannot be provided by CICS/VS:

- Terminal format independence
- Terminal device independence
- Terminal paging
- Message routing

3270 BMS can be specified during system generation to provide compatibility with previous versions of CICS/VS. Terminal device independence, terminal paging, and message routing need not be specified. However, if they are specified, they require that temporary storage (and also VSAM) be used.

BMS and terminal control commands can be used in the same application program, if required. Refer to the CICS/VS Application Programmer's Reference Manual (Command Level) for further information.

PROCESSOR CONSOLE AS A CICS/VS TERMINAL

CICS/VS allows the processor console to be used as a CICS/VS terminal. Users with only remote terminals may enter master terminal operator, system administration, and CICS/VS application transactions at the processor, thereby isolating these activities from any network considerations. CICS/VS will support multiple consoles as terminals in OS/VS systems with Multiconsole Support. See the CICS/VS Operator's Guide, for additional information about this feature.

Basic Mapping Support

The basic mapping support function (BMS) enables the application program to have access to input data, and prepare output data for transmission to terminals, without regard to the physical location of the data in the terminal message. Additional information regarding basic mapping support can be found in the appropriate CICS/VS Application Programmer's Reference Manual.

BMS uses 'maps' to describe the input format of data received from terminals, and (if necessary) to describe the format of output data to be transmitted to terminals. These maps are defined by the user (generally the system programmer) and are separately assembled and cataloged into the CICS/VS program library for retrieval when needed by application programs.

The application program accesses data from input messages, and prepares data for output responses, by field rather than by location of that information in the terminal message. Consequently, the application becomes less dependent on the actual message format. This format independence is one of the most significant advantages of BMS. Changes to message formats to meet various application requirements can be readily applied, by modifying only the BMS maps describing those affected formats, reassembling them, and cataloging the changed maps to the CICS/VS program library. All programs using these maps reflect the changed formats without modification of the programs. However,

recompilation of the programs might be necessary. In this manner, the installation will be more responsive to application needs. The use of BMS by application programs is illustrated in Figure 3.2-1.

BMS MAPS

An input map can specify data in an input message which is relevant to a particular program and ignore other data in the input. Several programs can then operate on the same input message format, using a unique map for each program. In addition, constant (or descriptive) information, if desired, can be defined in an output map and be automatically incorporated by BMS in an output message.

Basic mapping support provides the following services:

- Terminal device independence
- Terminal paging
- Message routing

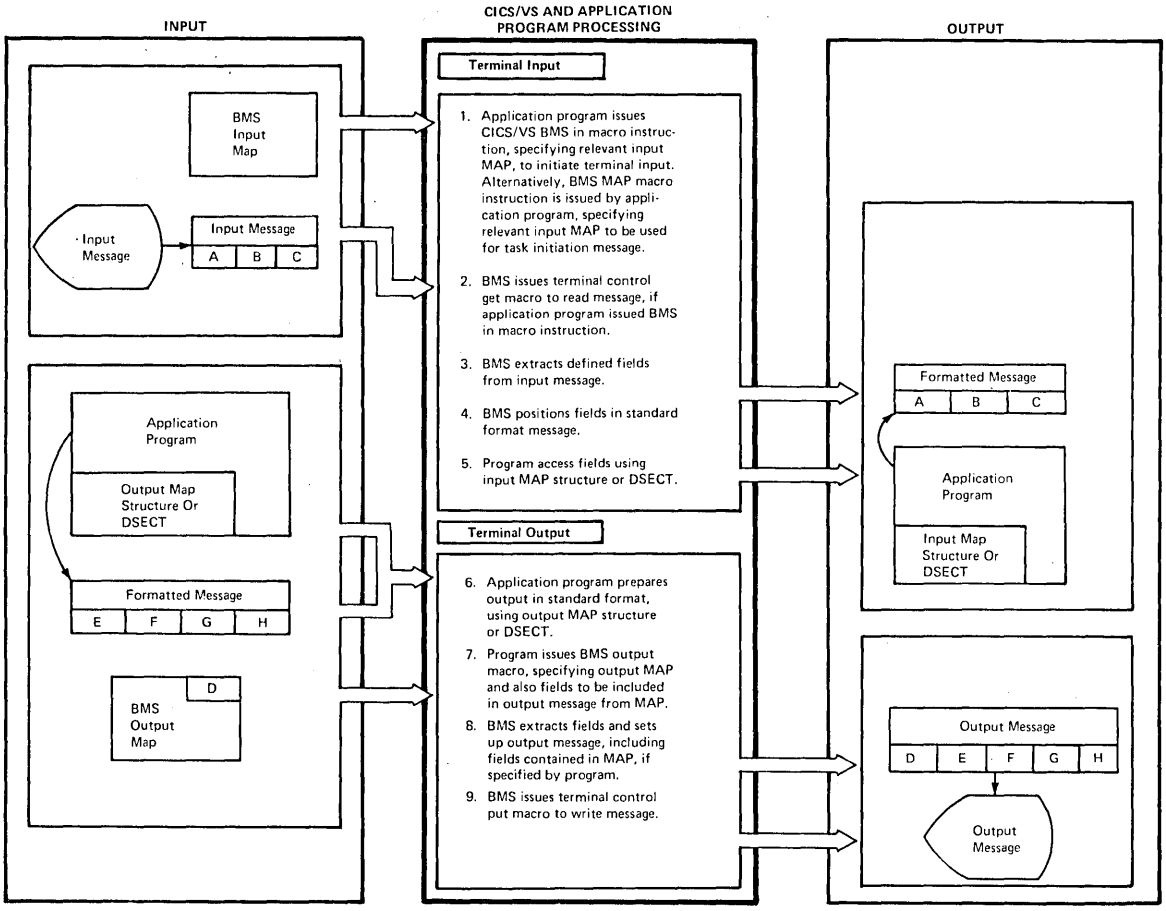
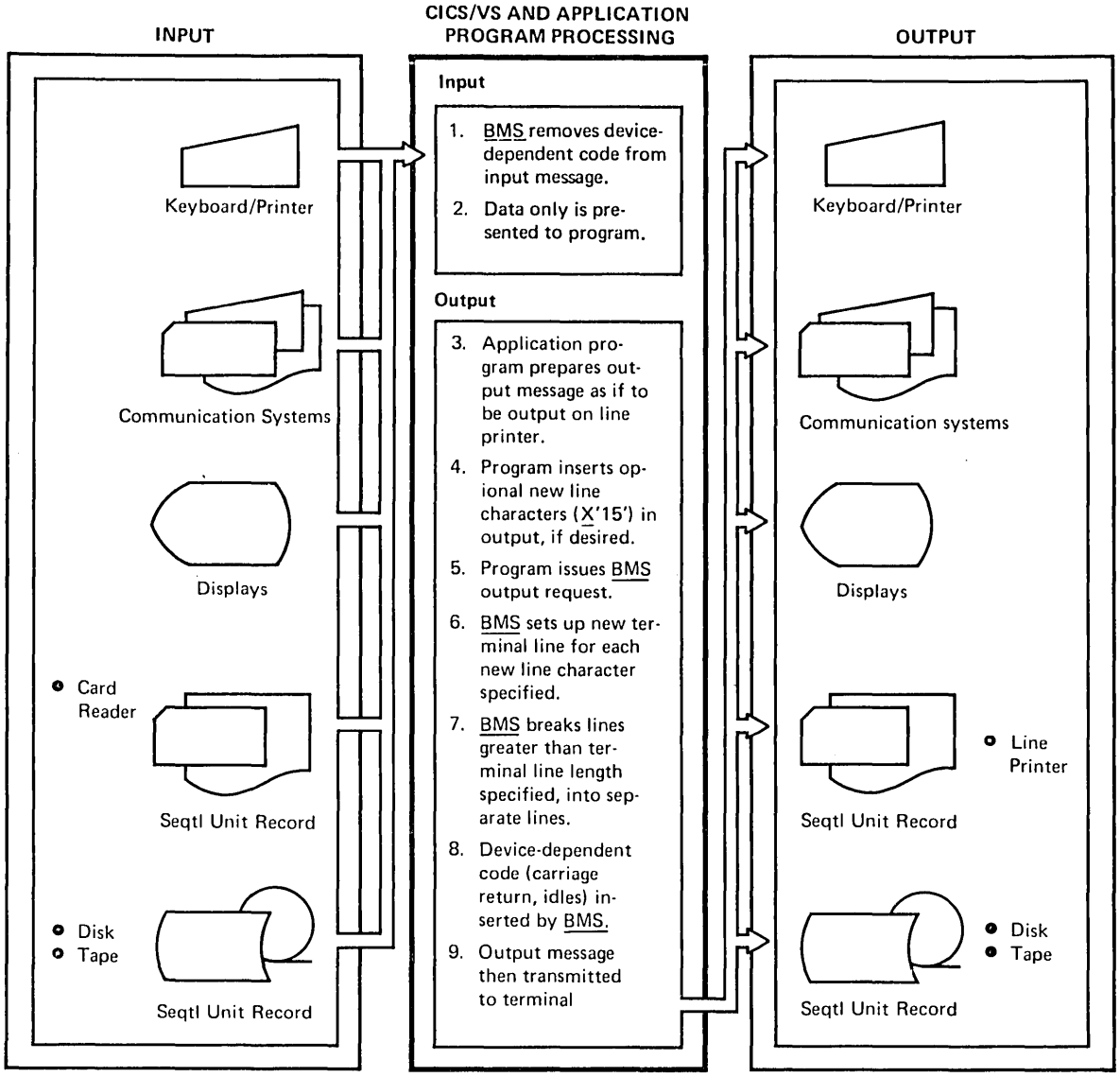


Figure 3.2-1. CICS/VS Basic Mapping Support (BMS)

Terminal Device Independence

Because Basic Mapping Support removes the need for the application programmer to code most terminal device-dependent support in his programs, programs can be written without regard to the input or output device used for transmission of messages to those terminals supported by BMS. BMS accepts input messages and transmits output messages to and from many of the devices supported by CICS/VS; for a complete list see the CICS/VS Application Programmer's Reference Manual (Command or Macro Level). The principles on which BMS operates are illustrated in Figure 3.2-2.



Extended Description
Device-dependent code may be removed by programmable controllers, such as the 3600, before transmission to CICS/VS. For the 3600 and 3650, input mapping requests are ignored.

Figure 3.2-2. CICS/VS Terminal Device Independence

Furthermore, the following sequential devices can be used to simulate online terminals, and transmit simulated terminal messages to and from the system:

- Card reader/line printer
- Tape drives
- Disk drives

INPUT MESSAGES

CICS/VS accepts input messages from any of the supported devices and, using the input map specified by the application program, converts the input message into a fixed format message, as specified by that map. Device-dependent characteristics in the input message are removed, and the appropriate fields are selected from the message and inserted in fixed locations in the mapped message.

In the case of the 3600, device-dependent characteristics in the input message are removed by the 3601 Controller and the input message is formatted for CICS/VS application program processing before transmission to CICS/VS. Consequently, BMS input mapping requests associated with 3600 input messages are ignored, and the data received from the 3600 is passed to the CICS/VS application program without change. See "Basic Mapping Support Using VTAM" for additional information.

OUTPUT MESSAGES

Output messages for transmission to terminals can be prepared without the control characters required for field positioning, or line separation. Output messages can be presented to CICS/VS as a data stream.

For 3270 display devices, CICS/VS will insert field specifications into the output data stream and position the data in the device's buffer in such a way as to give the screen layout defined in the output map. In other words CICS/VS ensures that the fields defined in the map have the specified attributes and positions.

For printer output devices, CICS/VS device-independent support divides the data stream into lines no longer than those defined for the particular terminal. If new-line characters appear occasionally in the data stream to further define line lengths, they are honored. CICS/VS inserts the appropriate leading characters, carriage returns, and idle characters, and truncates trailing blanks from each line.

Terminal device independence permits an application program to be independent of the terminal type (or types) in the installation, and can provide for support of mixed terminal types by the same program. This allows the use of backup terminals of a different type, or changeover of hard-copy terminals to display terminals when transaction volumes warrant the change, with little or no additional programming effort. It also reduces the amount of program maintenance necessary when changes are made in the terminal devices used by online programs, and permits increased growth flexibility in the installation.

Terminal device independence is the only BMS support available with the CICS/DOS/VS Entry Level System. Moreover CICS/DOS/VS Entry Level System only supports BMS requests to 3270 Information Display Systems and thus in this instance device independence means that the application programmer does not have to consider 3270 control characters etc.

Terminal Paging

Terminal paging is an additional feature that extends the capabilities of terminal device independence. The application programmer can prepare more output than can be conveniently or physically displayed at the receiving terminal. That output can be presented by CICS/VS as a series of pages. CICS/VS identifies and saves each page of information prepared by the application program.

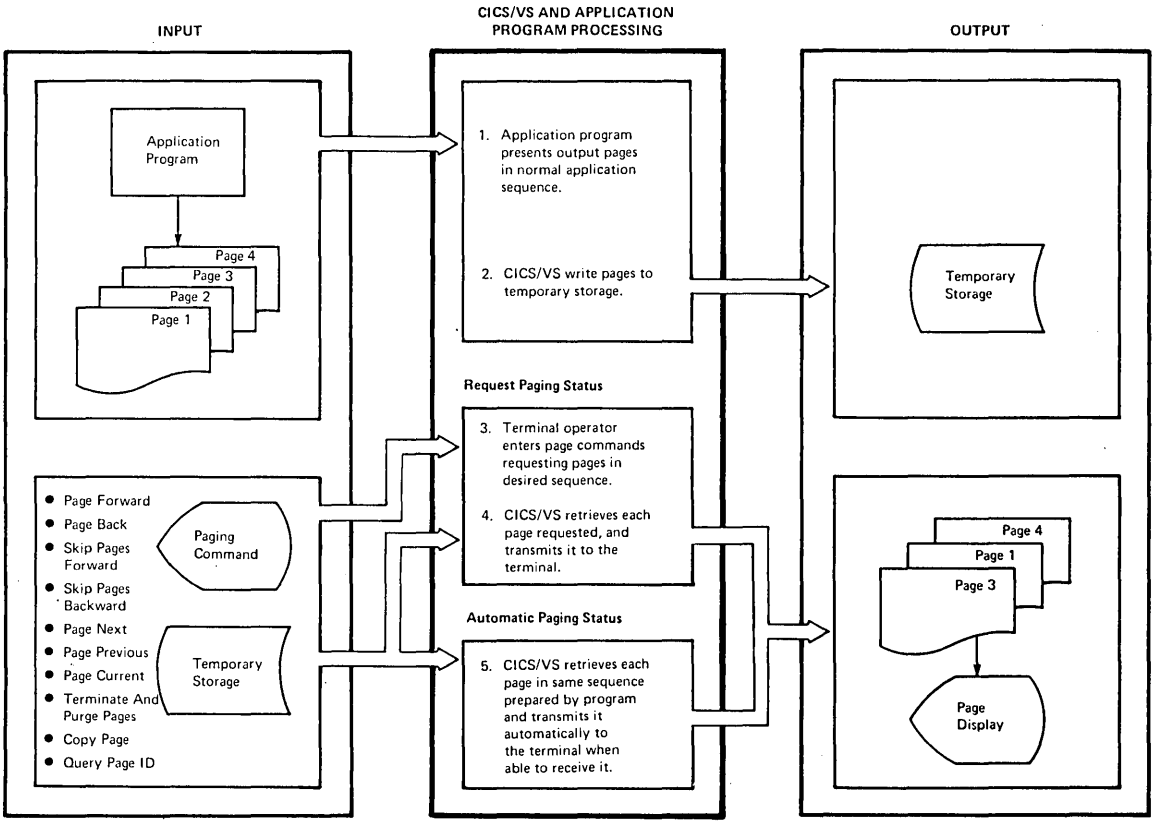


Figure 3.2-3. CICS/VS Terminal Paging

The terminal operator can then retrieve this output as a number of pages in any order; that is, in the order they were prepared, or by skipping forward or backward in the output page sequence.

CICS/VS provides a series of paging commands which can be used by the terminal operator to select pages for display in whichever sequence he desires (see Figure 3.2-3). In addition, for 3270 users, CICS/VS provides a facility known as single keystroke retrieval (SKR). Single keystroke retrieval enables the 3270 terminal operator to use program attention (PA) or program function (PF) keys to enter terminal paging commands. This significantly reduces the number of keystrokes that the operator needs to make during a page retrieval session. The way that SKR works is that the system programmer catalogs (in the system initialization table) selected page retrieval commands for representation by PA or PF keys. This does not mean that all the keys involved are dedicated to page retrieval; only when the initial page retrieval command of a page retrieval session is entered do the PA or PF

keys take on their page retrieval function. The PA and PF keys are then interpreted as paging commands according to the configuration in the SIT. Undefined keys (if pressed) are interpreted as invalid page-retrieval commands.

As an additional option, a PA or PF key can be used to represent the command that initiates a page-retieval session. The key is defined for task initiation in the TASKREQ operand of the DFHPCT TYPE=ENTRY macro for DFHTPR (the terminal paging program), as well as having a page-retrieval command allocated to it in the SIT.

When the page retrieval session ends, the PA or PF keys revert to any other functions that have been defined for them. The only exception is that if it is required to use a PA or PF key for the initial terminal paging command of a session, this key must be dedicated to the purpose.

It is possible to close the page-retrieval session and initiate a new transaction in one operation by entering the appropriate transaction code. However, a PA or PF key cannot be used for transaction initiation in this case, as it will be interpreted as a page-retrieval command.

The mechanisms by which the system programmer defines the use of PA or PF keys is described in slightly more detail under the heading "3270 Attention ID Transaction Initiation," in Chapter 3.3. For full details, refer to the CICS/VS System Programmer's Reference Manual.

Terminal paging also provides the ability to combine several small sections of data into one page which is then sent to the terminal. This is referred to as "page building" and enables the application programmer to prepare his output independent of the physical output capability of the terminal.

Terminal paging further relieves the application programmer of the need to concern himself with the presentation of information in a form suitable for display at the appropriate terminal, or with presentation of that information to the terminal in the sequence requested by the terminal operator. The application programmer can now prepare a series of pages of information, on the assumption that the terminal operator may wish to examine all of that information, and then present those pages directly to CICS/VS. No further programming is necessary to handle the selection of pages for display at the terminal. Page selection is made by the terminal operator, using the CICS/VS paging commands.

This will simplify program development of conversational applications and consequently increase programmer productivity and decrease the amount of future program maintenance necessary.

It is important that the system designer recognize that terminal pages are saved by CICS/VS in temporary storage. Temporary storage may be supported in main storage alone, or on auxiliary storage using VSAM; both will increase the demands for real storage during execution. Using VSAM on a processor with limited real storage available for virtual storage paging may increase paging and therefore influence online performance. Terminal paging is not supported by the CICS/DOS/VS Entry Level System.

Terminal paging is particularly oriented toward display terminals. However, it can also be used for hard-copy terminals. A terminal can be defined as having a "request paging" status or an "automatic paging" status.

Display terminals must use a request paging status, while hard-copy terminals can use either request paging or automatic paging status. Request paging status enables pages to be displayed at the terminal on request by the terminal operator, who can specify the sequence of pages to be displayed based upon his requirements.

Automatic paging status, such as normally used for a hard-copy terminal, causes CICS/VS to automatically output the next page of information on completion of a previous page. In this way, all information is presented to the hard-copy terminal in a continuous output stream. If required, an automatic paging terminal may be changed to request paging status by either the terminal operator or the application program, enabling only those pages to be printed which are of significance to the terminal operator. Similarly, the terminal operator or the application program can change request paging status to automatic paging for all terminals except display terminals.

Other terminal status specifications can also be used to indicate whether a terminal automatically receives messages sent from the processor or from other terminals. This is discussed under "Terminal Status" in Chapter 4.2. Additional information on terminal paging can be found in the CICS/VS Application Programmer's Reference Manual.

Message Routing

Message routing directs messages to one or more terminals in the system, either by use of the message switching transaction, CMSG, supplied by CICS/VS, or by the ROUTE command. In this context, the term "message switching" refers to the use of CMSG. The term "message routing" refers to the use of the ROUTE command. (The CMSG transaction itself uses the services of the ROUTE command.)

The CMSG transaction is entered by a terminal operator together with a message to be directed to another terminal, or to several terminals identified by the terminal operator. (This is discussed further in "Message Switching Transaction (CMSG)" later in this chapter.)

The ROUTE command permits an application program to send messages to one or more terminals not in direct control of the transaction. Message routing uses BMS, and saves messages in temporary storage to be automatically sent to the specified destination terminals if the status of those terminals allows for reception of the messages (refer to "Terminal Status" in Chapter 4.2). If a terminal is not immediately eligible to receive the message, CICS/VS preserves it in temporary storage until such time as a change in terminal status allows it to be sent, or a user-specified period of time elapses, whichever occurs first (see Figure 3.2-4). The message to be delivered is separated into a message for each terminal type that will receive it. Each separate terminal-type message is saved in temporary storage, together with a destination terminal list for that particular terminal type.

In addition, an application program can prepare pages of information to be transmitted to terminals, using BMS and the terminal paging

facilities as described above. These pages can be routed to one or more terminals or operators, through the use of the BMS ROUTE command.

MESSAGE DELIVERY

The application programmer specifies the identification of the terminal (or terminals) to receive the message, and, optionally, can also specify a time when the message is to be delivered. If the message cannot be delivered either immediately or at the specified future time, CICS/VS retains the message for a user-specified period of time. If it still cannot be delivered, CICS/VS notifies the originating terminal or an alternative terminal specified when the original message was entered.

CICS/VS allows messages to be directed, not only to specific terminals, but also to specific operators or operator classes. In this way, sensitive security information will only be delivered to those operators authorized to receive it. It is retained in temporary storage until the specified operators sign on to the specified terminals, and only then will relevant messages be delivered.

If a message is to be sent to a specified operator without identifying a terminal, that operator must already be signed on when the message is first presented to CICS/VS to establish the terminal identification to be used. If a message is sent to a specific operator and terminal, and that operator can never use that terminal (because of geographic location, for example), the message will be accepted by CICS/VS but may never be delivered. This is noted by CICS/VS upon expiration of the specified time within which the message must be delivered.

Terminals in the IBM 3600 Finance Communication System using VTAM are identified by a logical device code (LDC). Messages from CICS/VS are received by the appropriate 3601 application program which represents the specified terminal ID and controls the devices attached to the 3601. The message from CICS/VS identifies the LDC (specified by the application programmer) that is to receive the message; it is the responsibility of the 3601 application program to ensure that the message is delivered to the device indicated by the LDC. Logical device codes (which are also used by the 3770 and 3790 for device and data set selection) are described in detail in "Basic Mapping Support with VTAM" in Chapter 3.4.

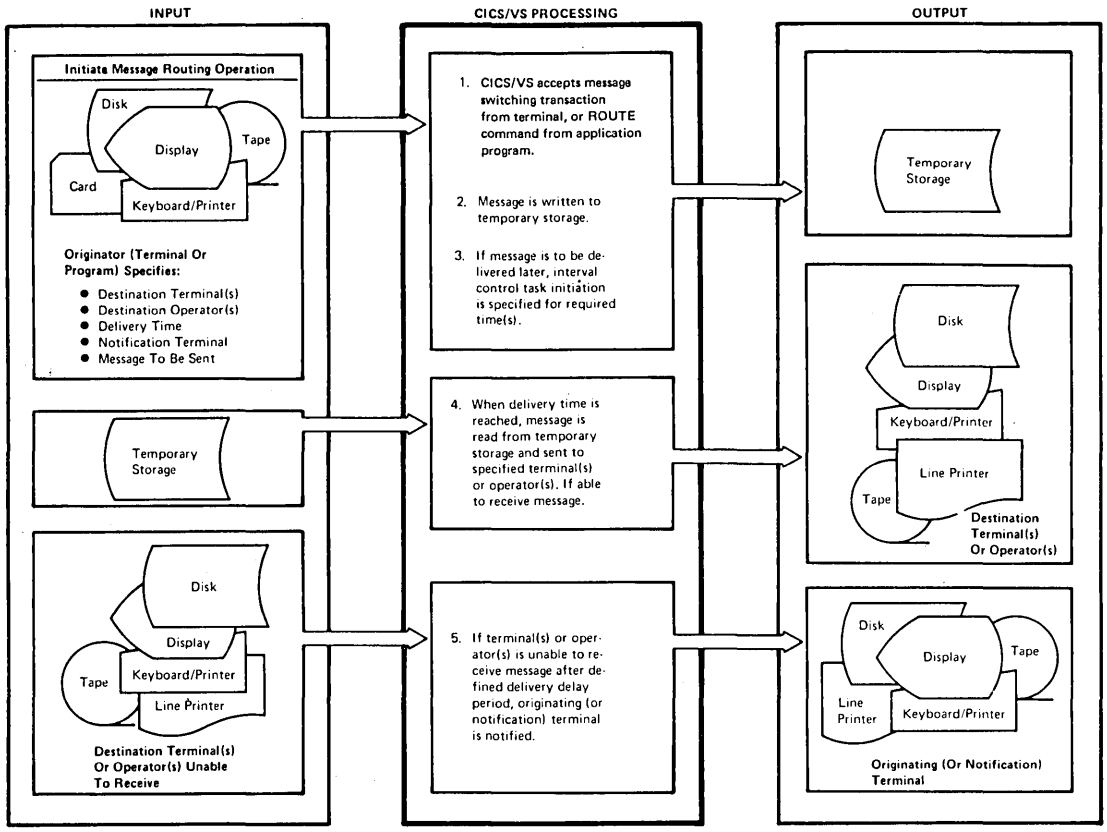


Figure 3.2-4. CICS/VS Message Routing

MESSAGE SWITCHING TRANSACTION (CMSG)

CICS/VS provides the message switching transaction CMSG for transmission of information between terminals. Figure 3.2-4 shows the use of this transaction.

The availability of the message routing feature in CICS/VS provides a valuable capability for communication of information, not only between terminals to satisfy application requirements, but also for better control of the online applications by the master terminal operator or supervisory terminal operators. These operators may broadcast messages to all terminals under their control informing them of certain significant information.

CICS/VS message routing utilizes temporary storage, which will use VSAM if auxiliary storage residence of messages is desired. Message routing and message switching are not supported by the CICS/DOS/VS Entry Level System.

Additional information about basic mapping support can be found in the CICS/VS Application Programmer's Reference Manual (Command or Macro

Level). 3600 logical device codes are described in more detail in the CICS/VS IBM 3600/3630 Guide.

Batch Applications

Batch applications are generally associated with high-speed data transmission terminals such as the 2770, 2780, 3600, 3650, 3735, 3740, 3770P, 3790, and 6670, or computers used as terminals, such as the System/3 Models 6 and 10, the System/7, or the System/370.

In this environment, the emphasis is often on the transmission of data from the terminal (or remote computer) to the central computer. Because of the nature of these devices, they are not designed for conversational interaction with a terminal operator, as is the case for conversational terminals. Generally, a batch of transactions is transmitted to the central computer, which processes that batch and then transmits any error messages back to the remote terminal or computer.

This online application environment is similar to the normal batch processing environment. In both cases, a batch of transactions is read and processed, and error messages are produced in an error list for offline correction.

This application approach is useful in an online environment where considerable amounts of information are to be transmitted across long distances. A high-speed batch terminal is able to transmit larger volumes of information than a conversational terminal, thus utilizing expensive long-distance transmission lines more economically. In this instance, the emphasis is on transmitting the data to the central computer as quickly and efficiently as possible, editing that data, and then transmitting any error messages back to the remote location quickly and economically.

ASYNCHRONOUS TRANSACTION PROCESSING

CICS/VS provides a function, called asynchronous transaction processing (ATP), which is designed for easy implementation of batch applications from pre-SNA batch terminals. ATP allows transactions, and the data associated with those transactions, to be transmitted in batches. Each batch is given a unique identification by the terminal operator. CICS/VS accepts each transaction from a batch terminal and delays its initiation until all specified input batches have been transmitted.

ATP requires that transient data intrapartition file support be generated as part of the user's CICS/VS system. This enables ATP to save batches of data for future processing and editing.

When all input batches have been transmitted, the transactions within the batch are then processed by application programs based upon their respective transaction codes. Any error messages are directed by the editing program to transient data for later transmission back to the terminal.

When the batches have completed processing, the terminal operator may then request that the output, if any, be sent to the terminal that originated the batch, or to a different terminal. Depending upon the amount of processing to be carried out on transmitted batches, the batch terminal may be disconnected from the transmission line by the user until output is available to be transmitted back to it.

The ATP facility is designed specifically for handling input from pre-SNA batch terminals such as the 2770, 2780, and 3780. ATP can also be used with some interactive terminals, such as the 2741.

Also, application programs which intend to execute under control of ATP must not use BMS terminal paging. Figure 3.2-5 illustrates the use of the CRDR and CWTR ATP commands, which are used respectively for input and output of batched data.

Batched data submitted through an SNA batch logical unit (BLU), such as the IBM 3770 Data Communication System operating in batch mode, does not need the ATP facility of CICS/VS.

The CICS/VS Entry Level System does not support ATP nor is ATP supported for VTAM terminals.

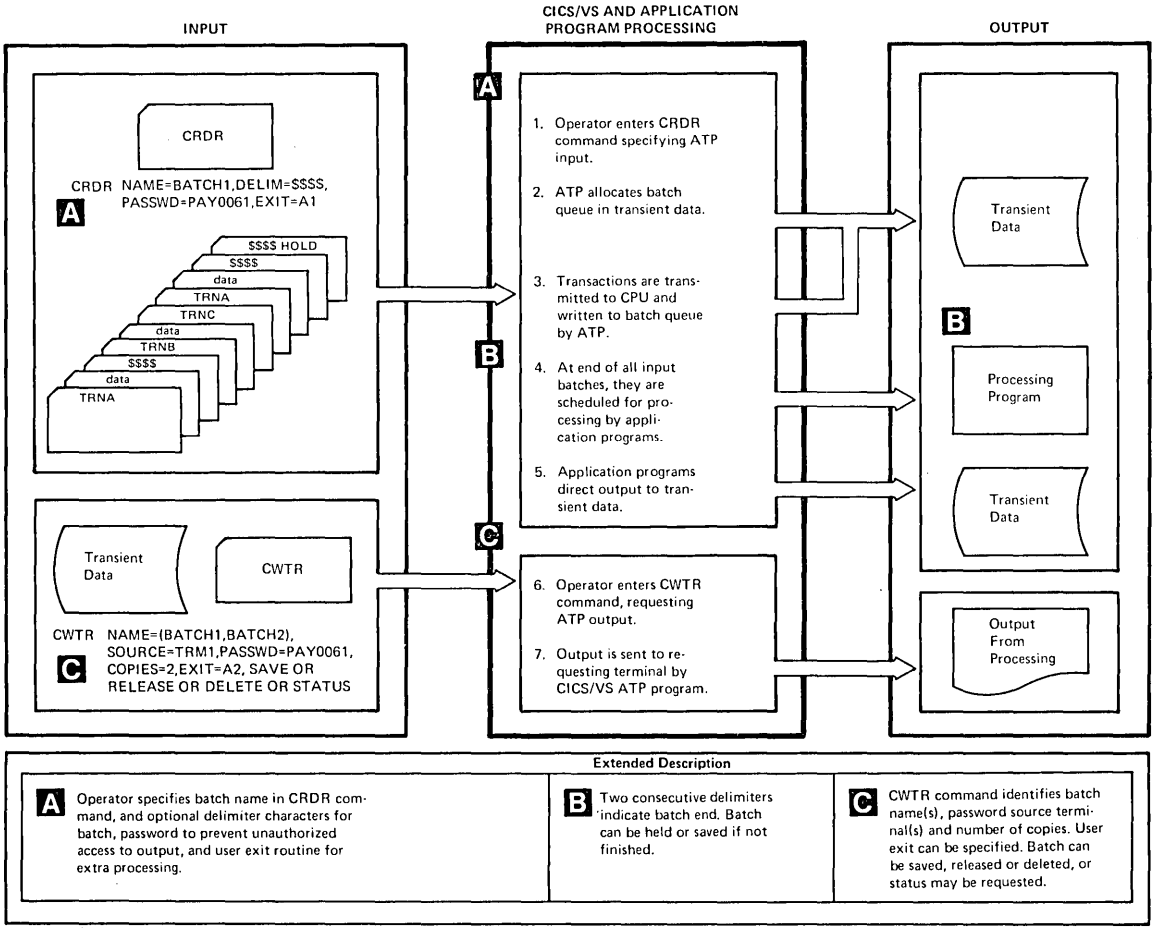


Figure 3.2-5. ATP Terminal Operator Commands

GENERAL BATCH PROCESSING

Some guidelines are presented below to assist in the design of batch applications when the design team does not wish to use ATP.

Execution of a batch application is, by its nature, of long-term duration. Accordingly, any storage required in executing batch application programs will be in use for a relatively long time, compared to conversational applications. Depending upon the amount of dynamic storage available for both batch and conversational applications, this requirement for long-term storage may affect the performance of conversational applications. To minimize the amount of storage used by batch programs, the following approach may be considered.

Ideally, an application program should be written to accept batch input transactions from the terminal, and queue these transactions on transient data. At the completion of each batch, the last transaction may automatically initiate a user program to validate and process the queued batch. In the meantime, the remote terminal is freed to allow further data input. On completion of batch processing, any error messages which were queued on transient data to send back to the remote batch terminal may be either automatically transmitted back as soon as that terminal is idle, or transmitted on request by the remote terminal.

This approach is similar to that adopted by ATP as described previously. It offers the principal advantage of very efficient utilization of data transmission lines, and the overlapping of processing one batch with the transmission of the next batch to be processed. On the other hand, ATP enables all input batches to be transmitted to the processor, and then allows the user to disconnect the batch terminal from the transmission line until all of those batches are processed. At that time, the user or the processor may re-establish connection between the terminal and the processor for output transmission.

This ATP approach is particularly economical when the processing time for all batches is longer than the input transmission time.

An alternative approach that can be used involves the batch application program reading a transaction, immediately following which the input transaction received is edited and error messages are queued on transient data for later transmission. However, this approach suffers from the disadvantage of less efficient data transmission. Data is transmitted from the remote terminal, followed by a pause for processing. Then the next transaction is transmitted and processed, with the line again being idle while the second transaction is being processed. No overlap of processing with data transmission is possible. If the pause between transmissions is long, a timeout may occur, and the terminal may lose control of the line.

Either the first method described above, or the use of ATP, is recommended for most efficient and economic line utilization.

CICS/VS BATCH DATA INTERCHANGE

The CICS/VS Batch Data Interchange functions are designed to support operation of the batch controller function of the 3790 Version 6 and 3770 Programmable and of LUTYPE4 logical units, such as the 6670 Information Distribution System.

Support is provided for the following 3790 data sets:

- Transmit Data Set
- Print Data Set
- Message Data Set

- User Data Sets
- Dump Data Set

A system designed to utilize the first three items is said to use the "Store and Forward" method of data interchange. When user data sets are included then this is termed "Store and Forward with Distributed Data". CICS/VS support for these methods uses the Data Interchange Program, which is supported only for assembler language programs. Additionally, certain functions are available through the BMS interface.

CICS/VS provides support for the following types of media on an LUTYPE4:

- word processing
- card
- console
- print

All are supported for input and output (except for print, which is output only). Application programs communicate with these media by means of the batch data interchange and BMS interfaces.

CICS/VS support is provided in accordance with SNA specifications, in particular the definitions and protocols for data management function management headers. The Data Interchange Program is conceived as a function manager in the host logical unit (primary), communicating with an equivalent function manager in the outboard controller logical unit (secondary). These function managers interchange user data as data streams.

For full details of CICS/VS support of the 3790 refer to the CICS/VS 3790 Guide, and of the 3770 Programmable and the 6670, refer to the CICS/VS 3767, 3770, and 6670 Guide.

Terminal Error Recovery

CICS/VS uses BTAM, GAM, TCAM, or VTAM for the control of terminals. These telecommunications access methods detect transmission errors between the central computer and a remote terminal, and automatically invoke error recovery procedures, if specified. These error recovery procedures generally involve the retransmission of data a defined number of times, or until that data is transmitted error-free. In the event that the error is not corrected after the specified number of retries, CICS/VS passes information connected with the error to the terminal abnormal condition program (BTAM- and TCAM-supported terminals) or to the node abnormal condition program (VTAM-supported terminals) for additional processing.

TERMINAL ABNORMAL CONDITION PROGRAM (TACP)

The TACP is used by BTAM- and TCAM-supported terminals. After determining that the error is unrecoverable, the TACP sets default actions based on keeping the network live. These may involve:

- Setting the terminal out-of-service

- Setting the line out-of-service
- Abnormally terminating the transaction
- Disconnecting a switched line

Before these default actions are taken, CICS/VS passes control to a user-supplied terminal error program (TEP) for application-dependent action if necessary (see Figure 3.2-6). On return from the terminal error program, TACP performs the indicated action as previously set by TACP or as altered by the TEP.

CICS/VS provides a sample TEP, which can be used to generate a specific TEP to meet the user's terminal error recovery requirements. A generated example of a TEP is supplied as part of the CICS/DOS/VS Entry Level System. (See the CICS/VS ELS User's Guide for additional information.) This TEP can be used without change or as an example when developing a unique user-written TEP.

Generation of a sample TEP is described in CICS/VS System Programmer's Reference Manual.

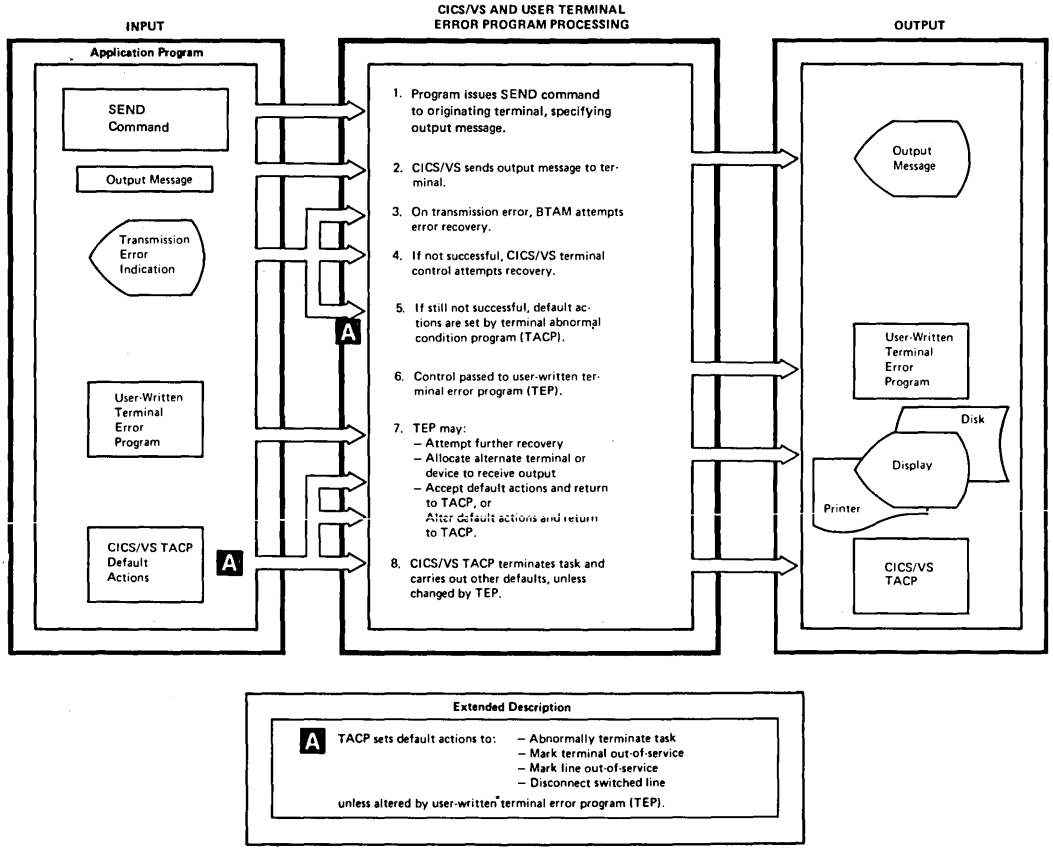


Figure 3.2-6. CICS/VS Terminal Error Recovery

TERMINAL ERROR PROGRAM

The terminal error program may be supplied by the user to attempt further error recovery, if necessary. Alternatively, a sample TEP can be generated or the generated TEP supplied with the CICS/DOS/VS Entry Level System may be utilized. (See CICS/VS System Programmer's Reference Manual.) For example, a user-written TEP can specify additional retries to be carried out by CICS/VS before the error is considered completely unrecoverable.

Alternatively, the user-written TEP can request that the output message be queued on disk using CICS/VS transient data, to be automatically transmitted to the error terminal when the problem has been rectified.

The user-written terminal error program might specify that the error terminal and line are not to be marked out-of-service, a switched line is not to be disconnected, or the task is not to be abnormally terminated. On return from the TEP, the task may be reactivated as if the error had not occurred.

This may be a satisfactory solution, if transmission of the output message is not critical to the application, but continued processing of the task is. For example, it may be necessary to allow the task to continue processing to enable various data sets to be completely processed and updated. Alternatively, the task may be allowed to abnormally terminate, and code at a label specified by the user in a HANDLE CONDITION statement may be utilized to complete urgent processing for the task.

Generally however, all processing associated with a transaction and task, and updating of relevant data sets, should be completed before the programmer makes any attempt to transmit an output message to the terminal. This can be ensured on VTAM-supported terminals by specifying that transmission be delayed until a WAIT TERMINAL command is issued, the program passes through a user synchronization point, or terminates. This is also the standard method used for TCAM- or BTAM-supported terminals. Receipt of an output message at the terminal should be regarded as an indication that all of the processing for the particular input transaction has been completed successfully.

NODE ABNORMAL CONDITION PROGRAM (DFHZNAC)

The Node Abnormal Condition Program (NACP) is used for VTAM-supported terminals to process abnormal situations associated with logical units. Information concerning the processing state of a logical unit is contained in the relevant TCT terminal entry, and in the VTAM request parameter list (RPL). There is no accompanying line entry as there is for TCAM- and BTAM-supported terminals.

The NACP is scheduled any time a VTAM request made by CICS/VS completes in error or cannot be honored. The receipt of a negative response sent by a logical unit also causes the NACP to be scheduled. This permits analysis of the sense information and issuance of any appropriate messages.

Whenever the NACP is scheduled, its analysis routines determine the actions that are mandatory to the recovery procedure. Prior to performing these actions, the NACP links to a node error program (NEP), which may be either a user-written program or the sample node error program provided by CICS/VS. The linkage between the NACP and the NEP

is through a module DFHZNEP that the user creates with the DFHZNEPI macro. Because VTAM and the the NACP take care of the transmission retries required to overcome temporary line errors, error conditions processed in the NACP should normally be regarded as permanent conditions. Thus it will not generally be necessary to override the NACP defaults in a user-written NEP. However, the NEP may be used to do application-dependent "clean-up".

NODE ERROR PROGRAM (DFHZNEP)

A sample node error program (NEP) is supplied with CICS/VS. In practice, more than one NEP is likely to be needed, in order that error recovery can be handled on a terminal/transaction-oriented basis. The purpose of the sample NEP is fourfold:

1. To provide a framework within which it is relatively easy for the user to add his own error programs. The same framework provides an environment in which user-coded error programs can run.

2. To provide fundamental error recovery for a 3270-VTAM network, consistent with that which is provided for a 3270-BTAM or -TCAM network by the sample terminal error program supplied by CICS/VS.

3. To provide error recovery for a 3767 or 3770 interactive logical unit, using a contention session when the receiver of a message is in transmission mode and the message has to be retransmitted.

4. To act as the default NEP in a multiple-NEP system.

The user is responsible for coding whatever additional node error programs are necessary for his VTAM-supported terminals.

Recovery actions for SNA devices may be dependent on the NEP 'understanding' the transaction currently attached to the terminal; it is thus necessary to associate an NEP with a particular transaction or transaction class. If a new transaction or transaction class is added to the system, existing NEPs should not be reprogrammed. Rather, the user should provide a new NEP to support the new transaction class.

To aid the user, certain optional actions are generated in the NACP. (For example, retry of a message.) If the user wishes any of these actions to be performed, he can set the relevant optional action codes in the TCT during NEP processing.

The user can issue VTAM responses or commands in the NEP (but not to a 3270). It is imperative, however, that the programmer has a full understanding of the way CICS/VS and the terminal interact, before embarking upon this type of error programming. The user can also issue SNA responses or commands from remote programmable controllers. For example, if a printer on a programmable controller runs out of paper, the user may code the controller to send a negative response to the processor, specifying a relevant user sense code. This will cause NACP (and NEP) to be scheduled in the processor. The NEP can then quiesce the logical unit using that printer, until the paper supply is replenished. For TCAM SNA support, these functions should be handled in the Message Handler.

MESSAGE LOGGING

Input and output messages may be automatically logged by CICS/VS for message recovery and resynchronization. In the event of loss of contact with VTAM supported terminals, logging and recovery protect message integrity. Transactions requiring message integrity are specified in the PCT. The programmable controllers should also log (as a minimum requirement) the VTAM sequence numbers of protected tasks.

For performance reasons transactions that do not change the system environment (such as inquiries that do not update data sets) should not specify message integrity.

In the event of system failure, CICS/VS emergency restart identifies in-flight tasks and backs out in-flight task activity. The input message for an in-flight-protected task can be used during emergency restart to establish message resynchronization with the controller. This is also true for a committed output message for which a positive indication of receipt was not received by the processor before system failure.

Chapter 3.3. Communication Techniques

Conversational Applications

The effectiveness of an online application depends to a large degree on man-machine communications. The computer is a tool used to achieve the objectives of the online application. To ensure success of online applications, the computer must provide the user with information to enable him to carry out his function effectively.

Data communication design represents the interface between the application and the machine. This is particularly true for conversational application design.

At all times during conversational message design, the system designer must keep in mind that the main objective of an online application is to assist the terminal operator. Thus, message formats should be designed to make the terminal operator's job easier. For example, input message formats generally should not be designed as fixed-format messages as for a card, but should enable the terminal operator to enter information in a variable-length format. CICS/VS can convert the variable-length input message into a fixed-length format for processing of the application program, as discussed below.

Also, if the task response time for a terminal operator is limited, the operator should be informed of the interval in which he is expected to respond.

TASK INITIATION

CICS/VS determines whether an input message received from a terminal satisfies an outstanding read request placed for that terminal by a currently executing program. If no application is currently active for the terminal which originated the input transaction, a task is initiated to process it.

Task initiation refers to the identification of a particular input transaction, the program to be used, and the creation of a task to process the transaction. Transaction identification can be achieved in several ways, as shown in Figure 3.3-1.

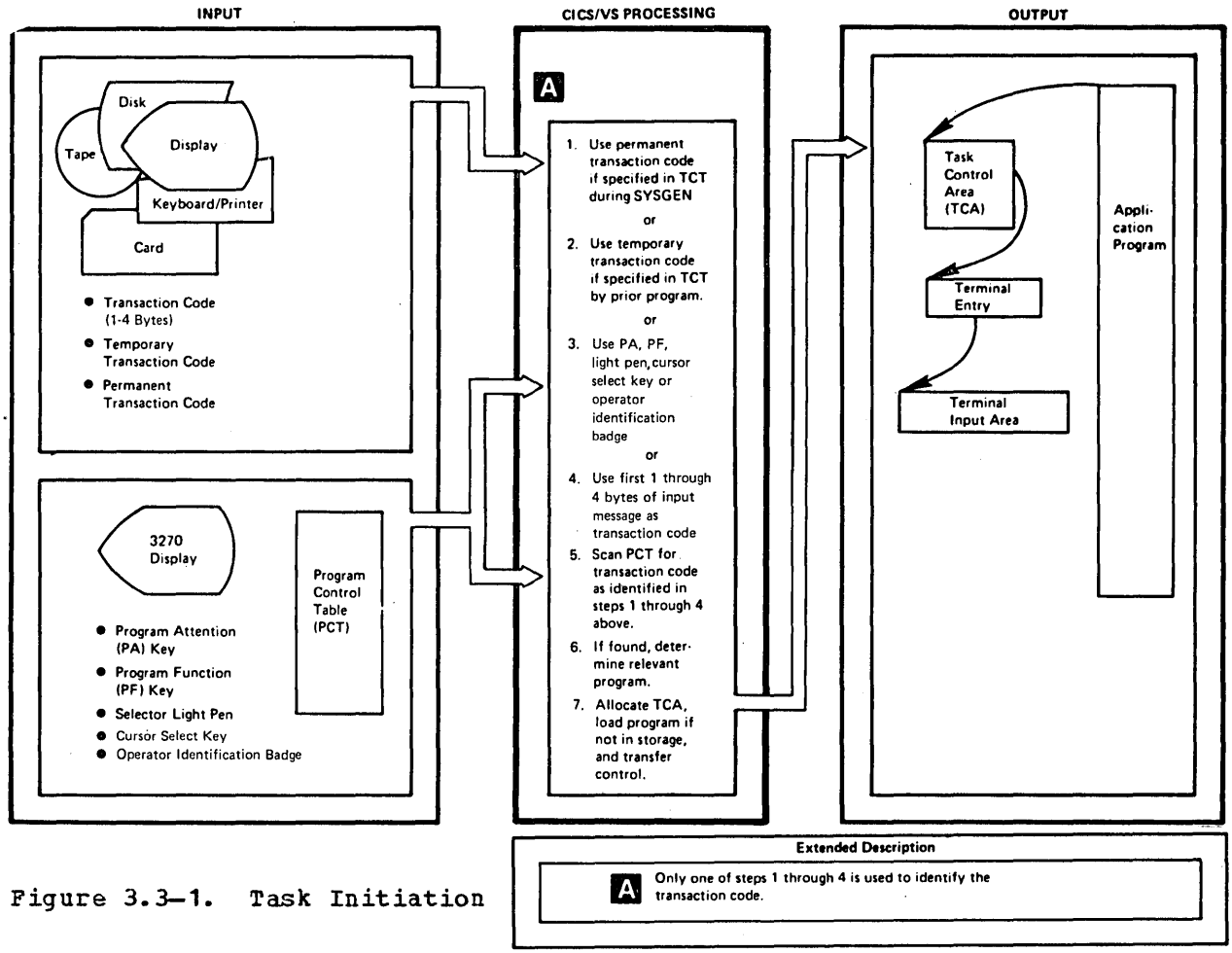


Figure 3.3–1. Task Initiation

Transaction Code

The first one to four bytes of a terminal message, delimited by a defined character, are used as a transaction code. Valid transaction code delimiter characters are the field name start character or field separator character (both of which can be defined in the system initialization table), and any code with a hex value less than or equal to X'40'. The transaction code is used to search the program control table (PCT) to identify that transaction code. On locating the appropriate entry in the PCT with the same transaction code, the name of the program to be first used to process the transaction is obtained. CICS/VS then creates a task control area (TCA) to control the processing of the transaction by the program. The PCT can also identify the size of a transaction work area (TWA) to be appended to the TCA and used as a program work area during processing.

The program name identified in the PCT entry is located by CICS/VS using an address pointer in the PCT pointing to the relevant program entry in the processing program table (PPT). The PPT entry for that program indicates the language in which it was written (Assembler, COBOL, or PL/I), the size of the program in bytes, whether it is presently in CICS/VS address space and if so, the number of other tasks concurrently using it, and the location of the program on the CICS/VS program library on disk. If the program is not already in CICS/VS address space, it is loaded from the program library and control is passed to it to process the transaction.

3270 Attention ID Transaction Initiation

In the case of the 3270 Information Display System, each of the Program Attention (PA) or Program Function (PF) keys the selector light pen, the cursor select key, or an operator identification badge can be defined in the PCT to initiate specified programs. By pressing the relevant PA or PF key, by selecting a detectable field with the selector light pen or the cursor select key, or by using an operator identification badge or magnetic stripe reader, the appropriate program is initiated. This is equivalent to entering a transaction code.

The types of transaction which can be initiated in this way are as follows:

1. Printing the contents of a 3270 Display buffer to a 3270 Printer. This is achieved by specifying PRINT=PAx in the DFHSIT macro, or at system startup.

2. Executing a program defined in the PCT with TASKREQ=PAx|PFy|LPA|OPID|MSRE.

3. As a special case of type 2, a page-retrieval session can be initiated by executing the page-retrieval program (DFHPTR) defined in the PCT. The PA or PF key must be defined in the SIT for page-retrieval by single keystroke retrieval.

The single keystroke retrieval (SKR) facility (described under "Terminal Paging," in Chapter 3.2) simplifies the 3270 terminal operator's work during page retrieval sessions by enabling him to use PA or PF keys to enter page retrieval commands. The mechanisms used by the system programmer to set up SKR differ slightly from the standard procedure for PA or PF keys. Once the initial page retrieval command of a page retrieval session has been entered by the operator, CICS/VS program logic arranges for subsequent transactions to be passed directly to the terminal paging module (DFHTPR), bypassing the usual decoding of the transaction. During the remainder of the page retrieval session, if the operator presses a PA or PF key, the system initialization table (rather than the PCT) is searched for the interpretation of the PA or PF key. It is thus in the system initialization table (SIT) that the system programmer catalogs the page retrieval commands that are to be represented by PA or PF keys. PA or PF keys so defined in the SIT are not dedicated to page retrieval. When the 3270 terminal operator ends the page retrieval session by issuing a purge command, the PA or PF keys revert to whatever function has been defined for them in the PCT. Only if it is required to use a PA or PF key to enter the initial page retrieval command in a session is it necessary to catalog that interpretation of the key in the PCT, and thus dedicate the key to that function. For full details, refer to the CICS/VS System Programmer's Reference Manual.

The use of the selector light pen and the cursor select key for transaction initiation is discussed in more detail in "Multiple Choice Format" later in this chapter.

Temporary Transaction Code

On completing the processing of an input transaction, an application program optionally may identify the transaction code to be used with the next input sent from that terminal. The next input need not be preceded by any transaction code, or PA or PF key, or be selected by the light pen, cursor select key, or an operator identification badge.

This program-identified transaction code is referred to as a temporary transaction code, and is specified in the RETURN command prior to termination of a task associated with that terminal. This temporary transaction code is used, with the next input from the terminal, to identify a program to be used to process that input. After use, the temporary transaction code is removed, and must be reestablished by a subsequent RETURN command, if it is to be used with further input from the terminal. Therefore, an application program can transmit a response to a terminal requesting further information from the operator. The next transaction code to be used is set by the program so that, when the requested information is supplied by the operator, the program to process that information will automatically be initiated.

Permanent Transaction Code

A permanent transaction code can be defined for any CICS/VS terminal, at the time the CICS/VS terminal control table (TCT) is generated. This is particularly useful for those terminals which, by the nature of their device characteristics, are unable to start an input transaction with a valid transaction identification. In this case, every input message is initially passed to the same application program, which is related to the permanently defined transaction code for that terminal. This application program examines the input to determine the processing required, and identifies subsequent application programs which operate on that transaction. The permanent transaction code is used with any input message from a terminal which does not satisfy a pending read request issued by a program. It also overrides any PA, PF, selector light pen, cursor select key, operator identification badge, or transaction code used in that message. A temporary transaction code cannot be used with a terminal utilizing a permanent transaction code. Certain VTAM sessions established for the 3650 or 3790 require that a permanent transaction code be specified in the relevant TCT entries for the sessions. (See the CICS/VS Guide for the terminal type in question.)

INPUT TRANSACTION DESIGN

The following design techniques for input messages may be used for terminals attached directly to CICS/VS or for terminals attached to programmable controllers such as the 3601, 3651, and 3791.

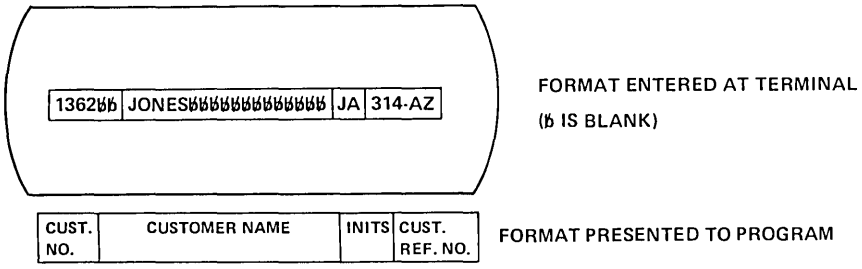
Fixed-format Messages

The fixed-format technique relates to the design of input messages such that each field of information occupies a fixed location in the input message (see Figure 3.3-2). While this is the normal technique for design of transactions entered from cards, it is not generally suitable for conversational applications. While a fixed message format makes it easy for the application program to extract information from the message for processing, this technique makes it more difficult for the terminal operator to enter that information, and is subject to operator error.

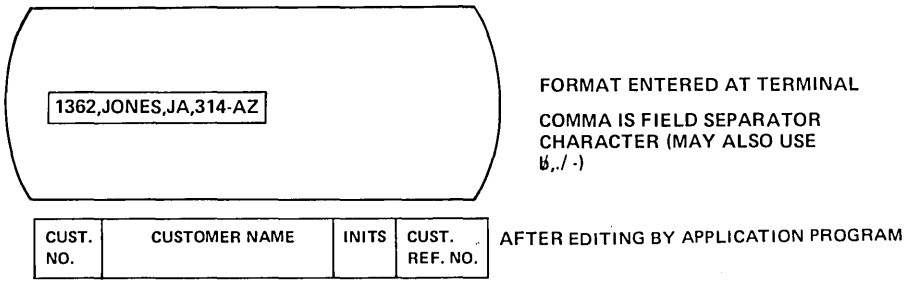
Variable-format Messages

The variable-format technique is similar to the fixed-format technique previously described, except that required fields need not always be located in the same positions in the input message (see Figure 3.3-2). Fields are identified by their relative positions within the message as for fixed-format messages, but each field is separated from others by a delimiter character or characters. Possible delimiter characters are the blank, slash (/), equal (=), comma (,), or dash (-). Using delimiters, the terminal operator can enter information in the required sequence, without concern for the actual physical location of fields within the message.

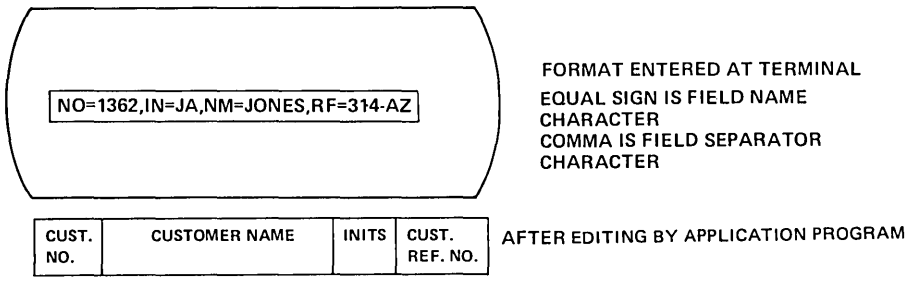
The application program must scan the input message for the delimiters and extract the data contained between them.



FIXED-FORMAT MESSAGE



VARIABLE-FORMAT MESSAGE



KEYWORD-FORMAT MESSAGE

Figure 3.3-2. Fixed-, Variable-, and Keyword-Format Input Messages

This technique can be used with CICS/VS application programs designed to process input from the 3600 Finance Communication System. BMS does not map input data from a 3601 controller, but passes it to the application program without change. (See "Basic Mapping Support with VTAM" in Chapter 3.4.) Therefore, the 3601 Controller can format the input message prior to transmission to CICS/VS for processing by the CICS/VS application program. This formatting involves insertion (by the 3601 AP) of delimiter characters in a variable format message entered from terminals attached to the 3601.

Keyword-format Messages

This format is similar to the variable-format message described above, except that each field is preceded by a field name start character and a unique keyword. The keywords and fields can be variable format. Because each field is identified by its appropriate keyword, the sequence of fields in the input message may vary.

The terminal operator enters information in variable format, in the sequence which is best suited to his requirements. The application program must locate each field based on its keyword.

Both variable-format and keyword-format information can be included in an input message. The CICS/VS input formatting macro instruction can process both input techniques as part of the same message.

Figure 3.3-2 illustrates typical fixed-, variable-, and keyword-format input messages.

Keyword-format messages offer maximum flexibility to the terminal operator, not only in the positioning of information in the message, but also in the sequencing of information in the message. A disadvantage for the terminal operator, however, is that additional information must be accurately keyed in, namely, the keyword for each input field. This additional keying takes time and is vulnerable to error, although it provides positive identification of each field. Because this keyword format permits a number of input fields to be present or absent, depending upon the characteristics of the application, it could in some instances result in less keying than for variable-format messages.

The keyword format technique can also be used for input from a 3601 Controller, as described earlier in "Variable Format Messages." The additional keying overheads of the keyword format technique are a consideration with input from 3601 controllers. The terminal operator can enter input to the 3601 in any convenient format. The 3601 AP can then insert the necessary keywords and delimiter characters before transmission to CICS/VS.

Fill-in-the-blanks Message Format

This message format accommodates the inexperienced terminal operator. It involves the display of descriptive information identifying each field to be entered, as illustrated in Figure 3.3-3, and applies mainly to display terminals.

The most useful approach is to display an image of the information normally provided on the input documents used by the application. For example, an image of a product order form may be displayed for an order entry application. The terminal operator, using the description

preceding each field, enters the required information. In the case of the 3270, only modified fields, such as that information entered from the keyborad, will be transmitted to the computer. The descriptive information is not transmitted, unless specified by the application program for identification purposes. Each input field transmitted from a 3270 is identified by its buffer address. This buffer address is

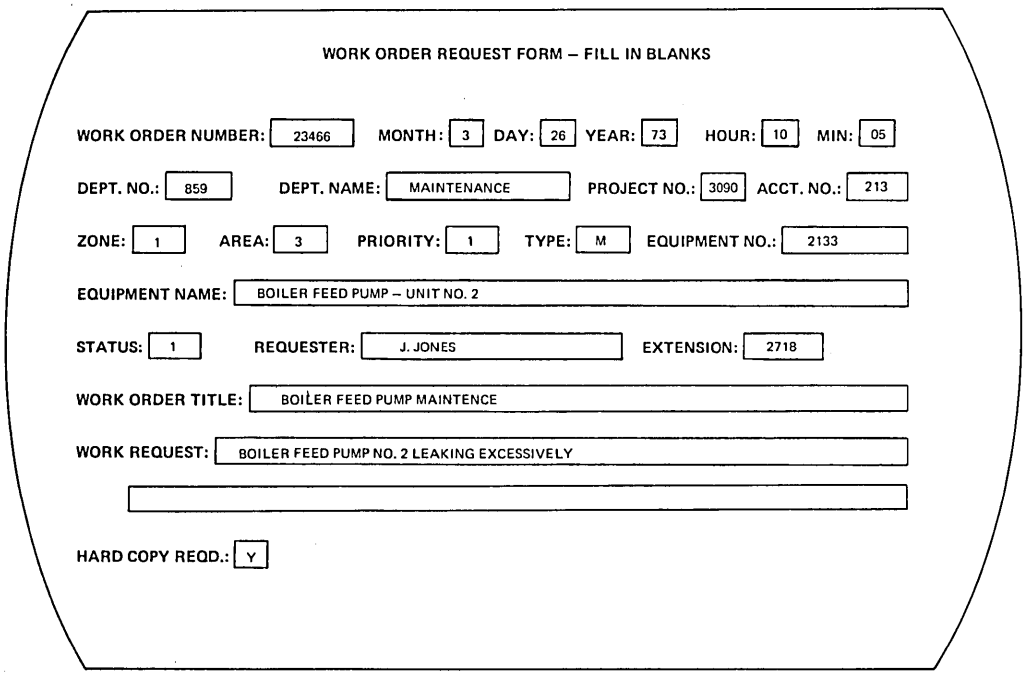


Figure 3.3–3. Fill–in–the–Blanks Input Message Format

used by basic mapping support (BMS), in conjunction with the input map defined for the transaction, to identify each input field and map the input message into a fixed–format message. Figure 3.3–3 illustrates a typical fill–in–the–blanks input message format.

An example of the use of this technique is found in the Display Management System (Program numbers 5736–XC2 for VSE, and 5736–XC2 for OS/VS). Refer to "Related Publications" in the Preface for relevant DMS publications.

Multiple–choice Message Format

This format uses the optional light pen or the optional cursor select key on the 3270 Information Display System and involves the display of a number of detectable fields. These fields present a series of multiple choices, one or several of which can be selected by the terminal operator by placing the light pen or the cursor to the detectable fields to be selected.

The output response from a previous application program may define certain fields displayed on the 3270 screen as detectable. Such fields

are identified by a question mark, an ampersand (&) symbol, or a blank character at the start of the field. A detectable field identified by a question mark is referred to as a "selection field". A detectable field identified by an ampersand or a blank is referred to as an "immediate" or "attention" field.

The appropriate choices are made by the operator, by touching the light pen to, or by placing the cursor under, the relevant field. Selection of a selection field causes the question mark to change to a greater-than character, to indicate that the field has been selected. Selection of an immediate field results in the transmission of a message to the processor. This message contains the buffer addresses of fields selected by the pen or cursor. If the immediate field is identified by an ampersand, the data in each modified field is also transmitted.

The attention ID (AID) character transmitted from the terminal on selection of an immediate field is used to locate the PCT entry for immediate detectable fields, and to transfer control to a common user-written program. This program examines the buffer addresses representing selected fields, and interprets these selections through the last BMS map used with that terminal. When designing detectable screen formats, each screen format to be supported by this common program should be identified.

Another technique for multiple choice input is for the application program to list (or display) several choices, identifying each choice by number. The terminal operator may then select an appropriate response by keying in its identifying number.

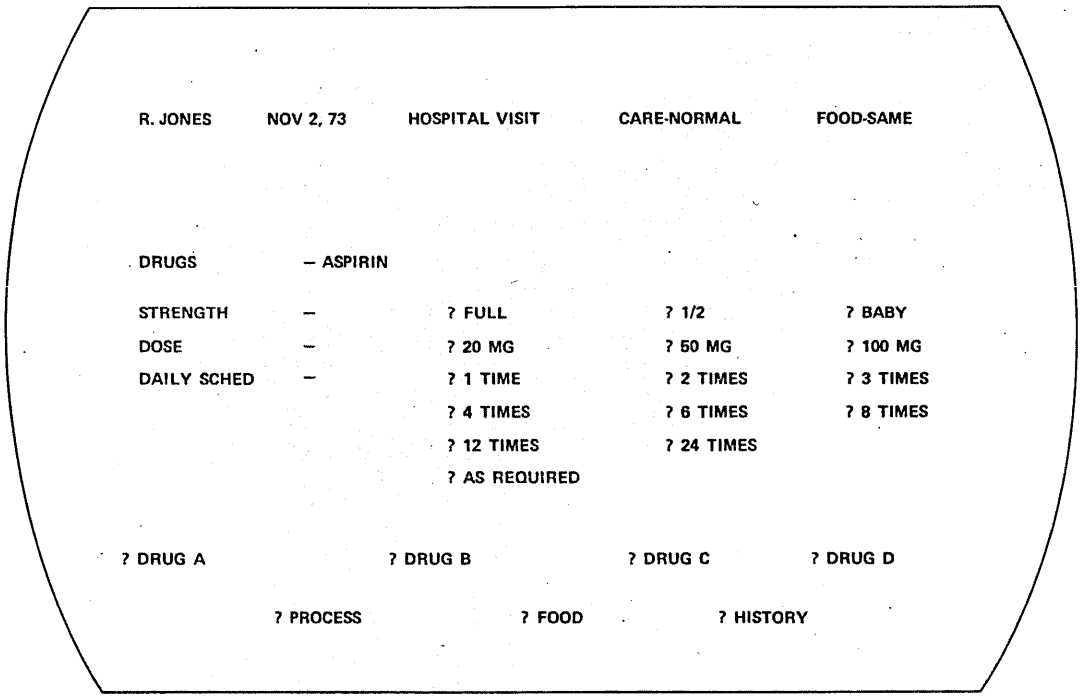


Figure 3.3-4. Multiple Choice Input Message Format

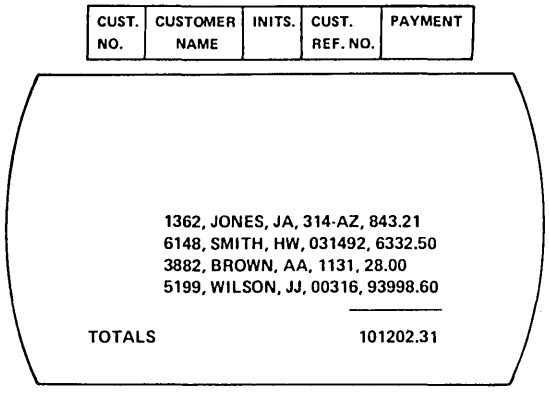
Selection of multiple choice fields can be used by unskilled terminal operators in user departments, to enter information for online applications. Figure 3.3-4 illustrates a typical multiple choice input message format using pen-detectable fields.

Any of these transaction formats may be used for terminals which communicate with CICS/VS either directly or through programmable controllers.

TRANSACTION EDITING

After defining the methods to be used for transaction initiation and designing the input message formats, the editing and validation to be performed on the message by the processor or programmable controller must be defined.

With the combination of BMS and editing, the application program is presented with an input message in a defined fixed format. The editing to be done by the application program is application-dependent; Figure 3.3-5 suggests some of the techniques available.



TRANSACTION EDITING TECHNIQUES

- FIELD VERIFY/EDIT
  - ALL ALPHABETIC (A–Z)
  - ALL NUMERIC (0–9)
  - ALL PACKED DECIMAL
- CHECK DIGIT
  - MODULUS 10
  - MODULUS 11
- HASH OR CONTROL TOTALS
- ZERO PROOF TOTALS
- LIMIT RANGE
- TABLE SEARCH
  - BINARY
  - SEQUENTIAL
- REASONABLENESS CHECK
- DATA SET CHECK
- SIGHT VERIFICATION
- KEY VERIFICATION

Figure 3.3-5. Transaction Editing Techniques

Many of these techniques may be implemented by the user in programmable controllers to provide for detection and correction of invalid data before transmission to the processor. Editing of data at the time of initial entry at the source permits: earlier detection of

errors, more efficient data transmission, and reduced processing. With the offline capability of BTAM supported programmable terminals such as the 3735, 3740, 3770P TCAM-supported programmable terminals and controllers such as the 3600, 3770P and 3790, and VTAM-supported programmable controllers such as the 3600, 3650, and 3790, data entry application availability is enhanced. Data may be edited and collected offline on disk storage for later transmission to CICS/VS. Only validated data need be transmitted to the processor. This data may be transmitted at line speed, resulting in significant time and cost savings.

Field Verify/edit

A field may be checked to verify that it contains the correct type of data. It may, for instance, be required to be:

- Entirely alphabetic (blanks, or A to Z)
- Entirely EBCDIC numeric (0 to 9)
- Entirely packed decimal (COMPUTATIONAL-3 in ANS COBOL or FIXED DECIMAL in PL/I)

If alphabetic characters have been entered into a data field that must be all numeric, for instance, an error message may be sent to the terminal operator notifying him that nonnumeric data was entered in the particular field.

Check Digit

Numeric fields may be checked for validity, by means of a check digit appended to the end of the field. Modulus 10 or Modulus 11 check digit editing is used to verify the correctness of a field or to identify errors. This technique can be used for an identification field such as a part number. When the number is first assigned, a user program computes the relevant check digit and appends it to the identification number. The check digit is then considered an integral part of the number, and can be used to check the accuracy of the entered number whenever that number is referenced.

Hash or Control Totals

Control totals of specified fields can be developed in a number of transactions. Control totals can also be developed for a batch of transactions and compared by the program against similar control or hash totals developed manually prior to entry of the batch into the system. If the program-developed and manually developed totals do not agree, the particular error or errors can be located by comparing the information entered in that field with the original source information for each transaction in the batch.

Zero Proof Totals

Generally, zero proof totals operate on a number of data fields within one transaction. These fields are added and subtracted together according to the requirements of the application. A nonzero result indicates an error in one or several of the fields.

Limit Range

This technique checks that the value in a data field lies between certain application-dependent limits. A field lying outside those limits is identified as an error.

Table Search

This editing technique uses the data field contents to locate a similar entry in an application-dependent table. If the exact field contents cannot be located in the table, an error is indicated. A substitute value can be presented to the application program, if required.

Reasonableness Check

The program applies various logical tests to the contents of a data field, to determine the reasonableness of that information as related to the particular application. If the contents of the field have not met the application criteria, it is identified as having an error. For example, the program may examine a product number entered as part of an order entry input message, in relation to the quantity of that product ordered. The application may require that certain products only be ordered in particular quantities. An ordered quantity outside that defined for the product would then be regarded as an error.

Data Set Check

This editing technique is similar in concept to the table search technique previously described, but is far more extensive and comprehensive. Information in the input transaction is used by the application program to access a data set related to that transaction. Information in that data set record is then used to validate other information in the transaction. For example, an application might require a customer's number and name to be entered. The customer number is used by the application program to access the relevant customer record, and the name in the record is compared with the name in the input transaction to determine that the correct customer number was entered with the customer name.

Depending upon the disk capacity and capability of the programmable controller, application data sets or subset information may be stored on disk in a remote controller. The 3601 controller permits up to 288,000 bytes of data to be stored on disk. The 3651 and 3791 controllers support as many as 9.3 million and 27.5 million bytes of disk storage respectively. This permits some data set validation of input to be

carried out in the remote controller before transmission to the processor.

Sight Verification

Sight verification can be used by the terminal operator in conjunction with data set checking as described above. In this instance, information from the input transaction is used to retrieve a relevant data set record. Information in that record is then transmitted back to the terminal for sight verification by the terminal operator. For example, in an order entry application, the customer number entered at the start of an order can be used to access the relevant customer record. The customer name and address are then displayed at the terminal for confirmation against the actual name and address of the person placing the order.

A less effective editing technique is the sight verification of keyed information against the source information, prior to transmitting the keyed transaction into the computer. This technique is subject to error and is completely dependent upon the accuracy and conscientiousness of the terminal operator.

Key Verification

Certain data fields cannot be edited by any of the techniques described above. An example of such data fields can be sales amounts relating to products, or dollar amounts to be entered. While control or hash totals can be developed across a series of transactions to identify an error, the application could require more complete checking than control totals alone. Key verification refers to the double keying of specified fields at different times. The first entry of the field is saved by the computer and compared against the second entry of that field at a later time. If both entries disagree, one or both of the fields are in error and correction is necessary.

Many of the editing techniques previously described can be implemented not only in CICS/VS, but also in programmable controllers such as the 3735, 3740, 3600, 3650, or 3790. The 3740 and 3790 are specifically designed for data entry and editing applications for many different industries.

These editing techniques may be integrated directly into CICS/VS application programs, or may be carried out in a prior data entry step. This step may be accomplished offline for later batch transmission to CICS/VS. Alternatively, it may be carried out online to CICS/VS with terminals such as the 3270, by using Video/370 (Program number 5736-RC3 under VSE, or 5734-RC5 under OS/VS). Refer to "Related Publications" in the Preface for relevant Video/370 publications.

ERROR CORRECTION

This section identifies some techniques which may be utilized by the system design team for error correction. Identification (through editing) of a transaction error to be corrected by the terminal operator can either be made:

1. on the first occurrence of an error in the message, or

2. after the entire message has been edited and all errors have been detected.

In conversational applications, the terminal operator should generally be notified by the application program of any errors immediately after the input transaction has been edited. The error message should be concise and meaningful, and should identify the particular field or fields in error, the nature of the error, and the action required by the terminal operator. The operator should be given the opportunity to obtain more information describing the particular type of error detected if he needs it.

Error Message Contents

The following types of error messages may be used, depending upon the requirements of the application:

- Error number
- Error number and text
- Abbreviated text, with a user-written HELP facility

An error number enables the error to be uniquely identified. Additional information describing the cause of the error may be provided in Terminal Operating Procedures documentation for the application. The user-written HELP facility enables the terminal operator to obtain more detailed information (that would otherwise be included in an operating procedures manual), by a special inquiry requesting the computer to provide the necessary detail. This technique has the advantage of keeping the most current operating procedures available to all terminal operators. It reduces the user's cost of developing, distributing, and maintaining written information on operating procedures for terminal operators. However, it has the disadvantage that it is utilizing available computer resources to provide information which can alternatively be documented in an operating procedures manual.

CICS/VS-generated system error messages to be transmitted to the 3600 terminals using VTAM consist of only the CICS/VS error messages and numbers documented in the CICS/VS Messages and Codes Manual. The 3601 AP must recognize the error numbers and insert the necessary text before transmission to the terminal operator. Additional information can be found in the CICS/VS 3600 Guide.

Error Message Documentation

The information which should be provided in documentation detailing an error includes:

- Error number and error message
- Cause of the error
- Operator correction

This documentation of error messages should be made available online or incorporated into the user's terminal operating procedures

documentation. Other required documentation should include CICS/VS terminal operating procedures and CICS/VS-supplied terminal transactions and error messages. See the CICS/VS Messages and Codes Manual and the CICS/VS Terminal Operator's Guide for additional information on error messages.

Error Field Correction

To use terminal operator time most effectively, the application should be so designed that the operator is required to enter only the field or fields in error. The operator should not be required to reenter the entire input transaction.

For example, in the case of the entry of new-business insurance policies that can approach 1000 characters in an input transaction, it would be unwise to require the entire 1000 characters be reentered if one field was in error.

However, in an order entry application, the information entered for each line item ordered is generally only product number and quantity. Detection of an invalid product number could require the reentry not only of the correct product number, but also of the quantity. Figure 3.3-6 illustrates an error field correction procedure which may be utilized by application programs.

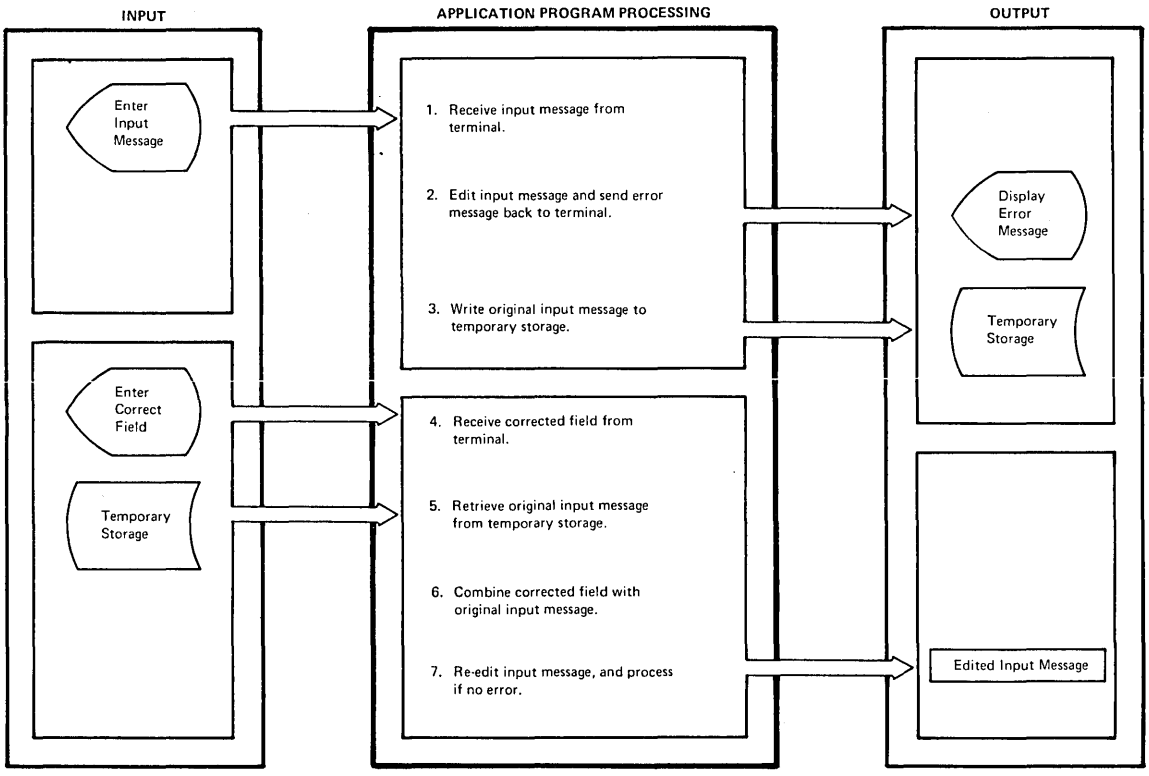


Figure 3.3-6. Error Field Correction

Use of Temporary Storage

If the terminal operator is required to input only the field in error, the application program must save the valid sections of the input transaction. Temporary storage enables application programs to save data either in dynamic storage or on disk, identifying the data uniquely for later retrieval by the same program or another program. As the terminal operator may take some time to enter the necessary correction, the valid part of the input transaction should normally be stored in temporary storage on disk, rather than in dynamic storage. Dynamic storage may then be utilized more efficiently for other purposes.

When transmitting an error message to a terminal, the application program may set a temporary transaction code (see "Task Initiation" earlier in this chapter). Using this, when the corrected field is retransmitted from the terminal, a unique correction program may be initiated, based on the temporary transaction code, requiring no further action by the terminal operator other than correction of the field.

The user's correction program retrieves from temporary storage the transaction that was originally entered by the terminal operator. The corrected field is then inserted in place of the error field, and the entire transaction is reedited to determine whether the correction is valid, and that no other errors have been introduced. In the event of other errors being detected, further error messages are sent to the terminal operator.

The error correction process may be iterative until the input transaction has been completely validated. In the event that the terminal operator is unable to correct the transaction, he should be allowed to enter a unique code (such as "CANCEL") instead of the corrected field, indicating that this error transaction is to be ignored. The transaction will then need to be completely reentered at a later time.

OUTPUT FORMATTING

The actual format of output responses is application-dependent. However, a number of guidelines may prove useful here.

The output response is the main interface between the online application, under the control of the computer, and the terminal user. Accordingly, it should be easily read and understood by a typical user of that online application.

The amount of information that must be presented in response to an input request depends upon the requirements of that request. For example, an inquiry requesting display of a customer's current account balance is a request for specific information. However, a request for display of a customer's account details generally requires all information relating to that account.

Terminal Paging

Depending upon the particular terminal device being used, the amount of information to be displayed may exceed the physical capacity of that device. For example, a 3277 Model 1 displays 480 characters in 12 lines of 40 characters per line. The display of 15 lines of information requires that information be broken into two pages.

The use of terminal paging in CICS/VS enables considerable flexibility to be achieved in output formatting, regardless of the physical capacity of the terminal which will receive the output. This feature is available only for BMS-supported terminals. Refer to "Terminal Paging using VTAM" in Chapter 3.4 for further detail.

Priority Processing

Each terminal operator is allocated a priority code as well as security codes. This operator priority is used in conjunction with terminal and transaction priorities to establish the overall task processing priority.

TASK PRIORITY

CICS/VS uses priority codes ranging from 0 to 255, where 0 is low priority and 255 is high priority.

Each operator, terminal, and transaction code can be allocated a priority code ranging from 0 to 255. The operator priority is contained in the sign-on table (SNT) and is copied across to the terminal control table (TCT) when the operator signs on. The terminal priority is also contained in the TCT, while the transaction priority is contained in the program control table (PCT) entry for that transaction code (see Figure 3.3-7).

When an operator enters a specific transaction code, his total priority is evaluated as the sum of terminal, operator, and transaction priorities. In the event that the sum of the three priorities exceeds 255, the task priority is set to 255.

This calculation of task priority provides the design team with considerable flexibility to ensure that the best performance and response time are provided in the areas where they are most needed.

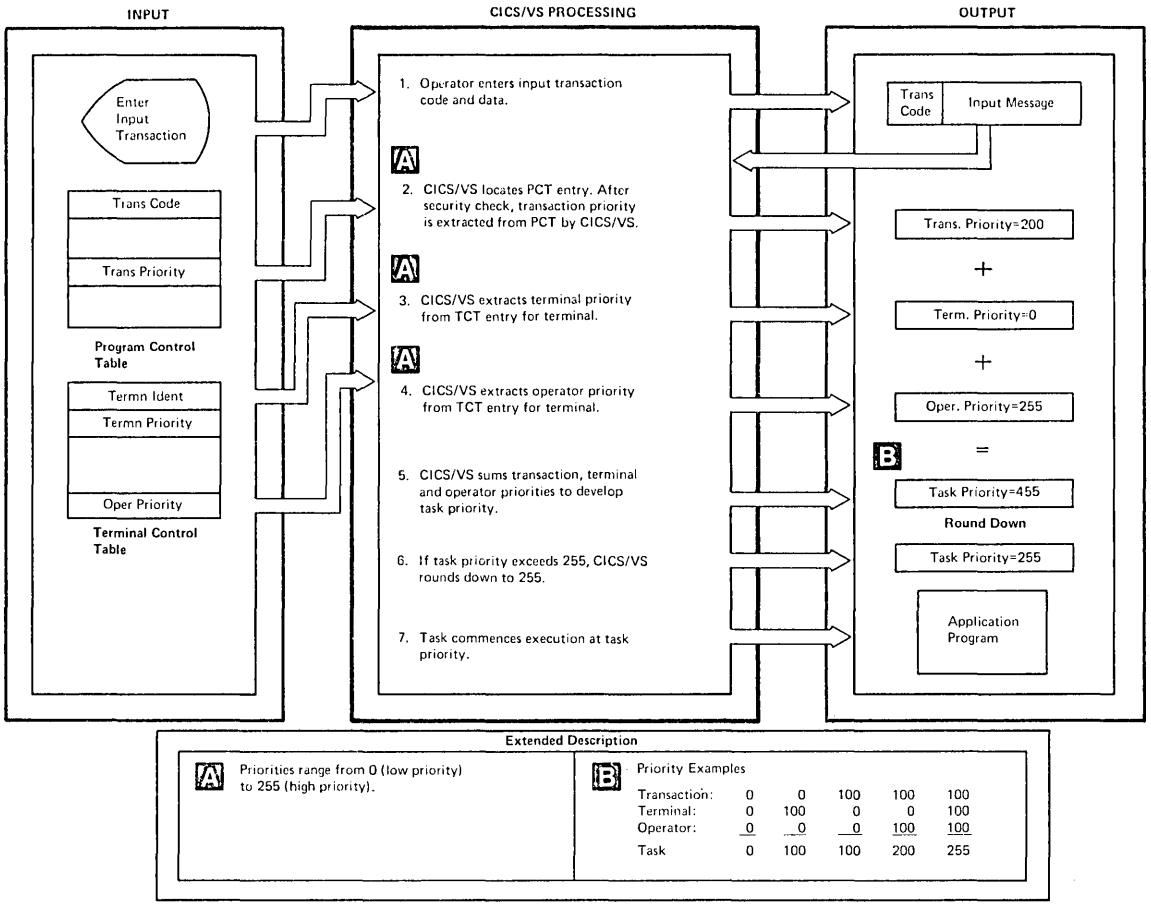


Figure 3.3-7. Task Priority

The task priority is useful in those cases where, because of the transaction volume, there may be several tasks concurrently executing. In this event, CICS/VS passes control to the highest priority task which is able to continue executing, and that task retains control of the processor until it requests various CICS/VS services. If the high priority task is not able to continue processing until a particular event (such as an I/O operation) has occurred, CICS/VS passes control to the next highest task which is able to execute. A high priority task is given preference in the use of the processor and other facilities even if entered later than a lower priority task.

In the event that two tasks with the same high priority value (for example, 255) are both ready to process, CICS/VS gives control to that task which reached the system first.

CICS/VS is an event-driven system, and as such does not seize control from a currently dispatched (executing) task. Therefore, even a low priority task will continue to execute once it has been dispatched, until it voluntarily relinquishes control by issuing a CICS/VS command. If no CICS/VS services are required by such a task, it should periodically issue a SUSPEND command.

Chapter 3.4. SNA Access Methods

This chapter describes some aspects of the relationship between CICS/VS and the SNA (system network architecture) access method ACF/VTAM. The information it contains also applies to other access methods that support SNA and provides a VTAM-like interface to user programs. These include certain releases of ACF/TCAM, and earlier versions of TCAM for which a message control program has been written to provide a VTAM-like interface.

In this chapter, as in the rest of this manual, the term "VTAM" is used to mean ACF/VTAM any access method that supports SNA and provides a VTAM-like interface.

For further information on VTAM, refer to Introduction to VTAM and VTAM Concepts and Planning.

For further information on TCAM, refer to OS/VS TCAM Concepts and Applications, GC30-2049.

Network Components

A network controlled by VTAM (or an equivalent access method) consists of the following components:

- communications controllers
- communication lines
- terminal systems (that may or may not be programmable)

These components are controlled by the following programs:

- Processor program (VTAM application program or TCAM MCP and applications). In this context, CICS/VS is an application program.
- Network Control Program (NCP)—This program resides in the communication controller.
- Function program or application program (AP)— This program is present only in programmable terminal systems and resides in the programmable control unit.

Communications Controllers

VTAM uses the IBM 3704/3705 Communications Controllers to enable part of the telecommunications processing to be moved out of the central computer and into the network. The 3704/3705 controls the flow of information between VTAM and terminals through use of a network control program (NCP). The 3704/3705 and its NCP support a variety of remote terminals. An NCP can be generated to handle lines in either network control mode (for TCAM- or VTAM-supported terminals), in emulation mode (for TCAM- or BTAM-supported terminals), or in both modes. An NCP generated with both functions is called an NCP with partitioned emulation programming (PEP) extension. This permits VTAM-, TCAM-, and BTAM-supported terminals to communicate with application programs (such as CICS/VS) through one 3704/3705.

Functions provided by the 3704/3705 include:

- line control
- dynamic buffering
- deleting and inserting line control characters
- translating character codes
- handling recoverable errors
- detecting permanent line errors
- gathering line statistics
- activating and deactivating lines and closing down the network

By performing these functions, the 3704/3705 and NCP conserve central computing resources. See Introduction to the IBM 3704 and 3705 Communications Controller for additional information.

SHARED RESOURCES

VTAM permits its network resources to be shared among the various VTAM application programs being executed in separate partitions/regions in the processor. One VTAM application program may be CICS/VS, which uses VTAM to establish communication between CICS/VS application programs and terminals, and another program could be TSO operating in a different partition/region.

VTAM controls the use of paths through the 3704/3705, communication lines, and programmable controllers so that several applications may communicate with different terminals on a single line. Also, the terminals on the same line may communicate with any of the application programs using VTAM. Thus, one terminal on the line may be communicating with CICS/VS, while another terminal on the same line is communicating with TSO. However, once a terminal begins to communicate with a VTAM application program, that terminal cannot communicate with another VTAM application program until the first program breaks its logical connection and releases the terminal. While connected to CICS/VS, the terminal may, of course, enter transactions to initiate different CICS/VS application programs.

Further efficiency in data transmission and data integrity is realized through the use of synchronous data link control (SDLC) data transmission. SDLC allows data to be transmitted in full-duplex mode (transmitted simultaneously in both directions along a communications line). VTAM supports both SDLC and full-duplex transmission.

A significant feature offered by SDLC is data integrity. Both VTAM and the control unit can check for error-free transmission of data and can request retransmission if an error is detected. Each transmission between VTAM and a programmable control unit is assigned a sequence number. Messages lost, because of component or communication failures, are easily detected and the lost data recovered.

See IBM Synchronous Data Link Control: General Information for additional information concerning SDLC.

Defining a VTAM Network

Because CICS/VS uses VTAM to communicate with advanced-communication subsystems, special considerations must be made when defining the network to VTAM. These considerations include:

- Use a VTAM APPL statement to define CICS/VS as a VTAM application program. CICS/VS must be authorized to acquire and pass VTAM connections with logical units.

  Buffer specifications in the APPL statement for CICS/VS depend upon the characteristics of the CICS/VS transactions.

  The name specified to VTAM for CICS/VS in the APPL statement must also be specified to CICS/VS. This name is specified to CICS/VS either during CICS/VS table generation (through the DFHTCT TYPE=INITIAL macro instruction, APPLID operand) or during CICS/VS initialization (through DFHSIT APPLID operand).

- Use the VTAM LU statements to define logical units. If necessary, use the APPLID operand to identify CICS/VS or another VTAM application program to which automatic logons are to be passed for each logical unit.

  Use the LOGTAB option to identify the VTAM interface tables to be used when identifying logons for CICS/VS.

- Define interpret tables, if necessary, describing valid logons for CICS/VS.

For additional information concerning VTAM definition, refer to VTAM Concepts and Planning.

Connection Services

Establishing sessions between CICS/VS and the logical units that control the terminals can be divided into two parts: connecting CICS/VS to VTAM and logging the logical units onto VTAM. The two parts must be carried out in that order and they are here described separately.

CONNECTING CICS TO VTAM

To VTAM, CICS/VS is an application program. The connection between the two is made during CICS/VS initialization by CICS/VS issuing a VTAM OPEN macro to open the CICS/VS access method control block (ACB).

The system programmer must specify, during VTAM definition, a name by which CICS/VS is to be known to VTAM, using the VTAM APPL statement. This name must then be made known to CICS/VS, for use in the OPEN macro, by means of the APPLID operand of either the DFHTCT TYPE=INITIAL macro or the DFHSIT TYPE=CSECT macro.

LOGGING THE LOGICAL UNIT ONTO VTAM

Defining the Session

To logon a logical unit to VTAM, CICS/VS must issue a VTAM OPNDST macro. The macro must specify a node initialization block (NIB) which indicates how VTAM is to handle subsequent communication between it and the logical unit.

CICS/VS associates a NIB with each logical unit's TCTTE. To cause CICS/VS to generate the NIB, the system programmer must specify ACCMETH=VTAM on the DFHTCT TYPE=INITIAL and TYPE=TERMINAL macros. CICS/VS then generates the control block by issuing a VTAM NIB macro instruction. The NIB is used by VTAM only during execution of the OPNDST macro, so the storage it occupies can be reused afterwards. For this reason, the NIB is built in storage separate from the TCTTE. The parameters it contains are established during VTAM definition; they cannot be altered by the CICS/VS user.

The connection between CICS/VS and the logical unit is known as a session. When CICS/VS issues the OPNDST macro, it supplies a set of BIND parameters. These tailor the session to the requirements of CICS/VS and are sent to the logical unit by VTAM as part of a BIND command. CICS/VS has a set of BIND parameters for each type of logical unit. Establishment of the session is completed when the logical unit sends a positive response to the BIND command.

The logical unit is known by one name throughout the network. It is defined during VTAM definition by the LU macro. It must also be defined to the NCP during the control program definition. The system programmer uses this name to identify the logical unit to CICS/VS by specifying it in the NETNAME operand of the DFHTCT TYPE=TERMINAL macro. CICS/VS then associates the TCTTE with the logical unit by using this name for the VTAM NIB macro used to build the corresponding NIB.

Logging On

Although the VTAM OPNDST macro that makes the connection between the logical unit and VTAM must be issued by CICS/VS, logon can be initiated by either the host processor or the logical unit (except that 3790 sessions cannot be initiated by the processor).

Processor sessions can be initiated by:

- CICS/VS, automatically, at CICS/VS initialization.
- CICS/VS, automatically, following automatic task initiation (ATI).
- a request by the VTAM network operator.
- a request by the CICS/VS master terminal operator.

Logical unit sessions are initiated by:

- the terminal device, when the terminal operator logs on.
- the terminal controller, on request of the the terminal operator.
- the terminal controller, automatically, at controller initialization.
- an application program running in the terminal controller.

In general, a CICS/VS logon exit is scheduled by VTAM whenever it receives a logon request, whether it comes from the processor or the logical unit. CICS/VS then issues the VTAM OPNDST macro within the exit routine.

When CICS/VS is to logon a logical unit automatically at CICS/VS initialization, as will probably be the case for an output-only logical unit, for instance, it must cause VTAM to simulate a logon request from a the logical unit. CICS/VS does this by issuing a VTAM SIMLOGON macro. The system programmer causes CICS/VS to issue this macro during initilization by specifying CONNECT=AUTO on the DFHTCT TYPE=TERMINAL for the logical unit.

If a logical unit is required by one VTAM application is already connected to another, it can be released by the one and connected to the other without the need for the full logon processing to be carried out by VTAM. To receive notification of a request by another application for a logical unit, an application must provide a RELREQ exit routine. CICS/VS provides such a routine. It will be entered when another VTAM application program issues a VTAM SIMLOGON macro for a logical unit currently in session with CICS/VS. When it has no more work for the logical unit, CICS/VS will release it to the other application, provided the system programmer has specified so in the RELREQ option of the logical unit's DFHTCT TYPE=TERMINAL macro. Conversely, when CICS/VS issues a SIMLOGON macro, it does so with the RELREQ and Q options. If the logical unit is already connected to another VTAM application program, and that program has a RELREQ exit routine, then this routine will be entered, and the application may release the logical unit.

CICS/VS Session Types

The various session types that CICS/VS supports are outlined. For a full description, refer to the appropriate CICS/VS Guide.

The Virtual Telecommunications Access Method (VTAM) is used by CICS/VS to support a number of terminal systems, some of which are programmable. Programmable terminal systems enable programming tasks (such as transaction editing, validation, and formatting) relevant to a remote location to be carried out in programmable control units at the remote location. The programmable control units can operate either online to a processor, or offline in a standalone mode. Operating offline, many of the control units can permit terminal operation to continue independent of availability of the main processor or associated communication links. Terminal transactions can be recorded on disk storage (which is part of the programmable control unit) for later transmission when processor communication is available. Disk storage also enables selected application data sets to be stored in the programmable control unit for direct reference by application programs executed in the control unit on behalf of terminals.

This concept of "distributed function" enhances performance and permits cost efficiencies to be realized in the areas of data transmission, system availability and configuration, and application flexibility.

3270 SESSIONS

Functions previously supported by CICS/VS through BTAM are supported for VTAM-3270 sessions, with the exceptions that 2260 compatibility and ASCII translation are not provided. In particular, the following services are provided:

- Operator sign-on/sign-off
- Basic mapping support
- Terminal paging
- Message routing and message switching
- Automatic task initiation
- Master terminal operator functions
- Supervisory terminal operator functions
- Printing the contents of a display screen
- Protection from unsolicited data
- Upper-case translation

VTAM-3270 support differs from TCAM- or BTAM-3270 support in the following particulars:

- Hard copy is sent to pre-assigned printers, although for the 3274 and 3276 devices hard copy may be printed locally using an installation defined print authorization matrix. This matrix may be defined via application programming and is used for printer allocation.
- ASCII is not supported by CICS/VS. However, ASCII-EBCDIC translation can be carried out for a BSC 3270 by translation tables in the network control program

- Line status is not available from the master terminal but the terminal connection status will be displayed instead
- For SDLC connected terminals, the ISSUE COPY command is not supported
- System generation for the two different access methods is not, in general, compatible

VTAM-3270 support differs from TCAM or VTAM support of some other terminal devices or systems as follows:

- 3270 is not programmable
- 3270 logical units do not employ a function management header (FMH)
- Logical device codes (LDCs) are not applicable to 3270
- The message recovery facility is available but not message resynchronization
- The transaction code to initiate a task may be a program function key, program attention key identifier, magnetic stripe reader or light pen
- Error messages, rather than negative responses, will be sent to the terminal. However, for SDLC connected 3274 and 3276 devices, negative responses as well as error messages are sent.

For further information see the CICS/VS 3270 Guide.

3600 SESSIONS

A 3600 VTAM session is established between CICS/VS and a 3600 application program block (APB) in the 3601 controller. The APB is regarded by CICS/VS as a logical unit, referred to by its terminal identification in the TCT. The APB, in turn, controls the operation of one or more terminals attached to the 3601.

The 3614 terminal may be attached directly to a 3704/3705, as well as to a 3601, and is then regarded as a separate logical unit.

Each application program (AP) may control several terminals. Individual terminals are transparent to CICS/VS and it is the responsibility of the AP to interpret specific requests from the terminals, communicate them to CICS/VS, interpret the output from CICS/VS, and direct it to the appropriate terminal.

CICS/VS communicates directly with an APB as a separate logical unit, and is unaware of the operation of other APBs in the 3601. The other APBs may be communicating with CICS/VS as other logical units, or with other TCAM application programs.

3600 sessions permit the following CICS/VS services to be supported.

- Operator sign-on
- Basic mapping support
- Terminal paging
- Message routing and message switching
- Automatic task initiation
- Master terminal operator (CSMT) functions
- Supervisory terminal operator (CSST) functions
- Message recovery and resynchronization

Pipeline Session

One pipeline session may be established in a 3601 to support multiple 3606 or 3608 terminals. The purpose of this session is to support minimum delay transactions from 3606 or 3608 terminals, such as cheque verification, credit card authorization and debit (cash) card transactions.

CICS/VS permits a number of TCT entries to be specified as belonging to the pipeline session. These are used as a pool of entries to permit multiple tasks to be initiated for different 3606s or 3608s using the pipeline session. A separate pool of TCT entries can be specified for each 3601, or all TCT entries can be combined in a single pool to service all 3601 pipeline sessions.

The TCT pools represent a number of user-specified tasks to be run. CICS/VS passes an input transaction from a pipeline session to one of the available TCT entries in the pool for that session. This permits the processing of a number of transactions to be multitasked for optimum response.

To identify the 3606 or 3608 initiating pipeline transaction, the 3601 transmits a terminal identification to CICS/VS at the beginning of the input transaction. This is not used by CICS/VS, but is passed to the CICS/VS application program. The program may use the identification to maintain audit trails or statistics.

When communicating with a pipeline logical unit, the following considerations apply:

- No automatic task initiation is supported.
- No CICS/VS logging will be performed, that is, no recovery/restart facility provided.
- Each input to the host over the pipeline session is a separate task. The CICS/VS task may request one or no writes back to the session node. Thus one message in and one message out are the maximum during a task life.

- If data is received and no terminal entry is available, the data will be discarded and a negative response returned to the 3601. System Sense will be provided indicating insufficient resources. It is the responsibility of the pipeline APB in the 3601 to handle the retry.

- The 3606/3608 terminal identification must be sent in the data stream to the host with every input from the 3601 through the pipeline session. This identification must be inserted in the message by the APB in the 3601. CICS/VS does not use this terminal identification but passes it to the CICS/VS application untouched. The user may use the identification to maintain audit trails or statistics.

  The CICS/VS host application on a pipeline session is responsible for properly formatting output messages. This includes the 3606/3608 terminal identification required by the 3601 to perform routing functions. It is the responsibility of the user to ensure that the APB and the host application program follow the same conventions with regard to the data stream.

- All SNA sessions will be initiated via the 3601 session control.

- A conversational transaction may not use this facility, that is, the output sent by the CICS/VS application program may not ask for additional input to this task. This is enforced by CICS/VS in that a RECEIVE command issued by a task will cause the task to be terminated.

- CICS/VS does not enforce the use of any particular data stream in a 3606/3608 pipeline session.

- The entire transaction must be contained in a single request unit (RU). Output chaining or multiple messages output from the CICS/VS application task are not supported. This is enforced by CICS/VS in that any SEND command after an initial output will be rejected and the task will be terminated. If the output data stream exceeds the buffer capability associated with the 3601 pipeline buffer, chaining will not occur and an error will be passed to the Node Abnormal Condition Program (NACP).

- Unique operator identification is not included and all transactions are processed as if generated by a single source in a full-duplex environment. Output from the attached task is sent to the node defined by the SNA origin address field/destination address field (OAF/DAF), and any further routing information must be placed in the message by the user application, for use by the 3601 FP.

- The 3606/3608 terminal cannot be used as a master terminal.

- Function Management Headers are not supported by the pipeline session.

The above restrictions for a pipeline session exist because the pipeline session is designed to provide a high throughput for a specific transaction type. The user need only define one session per 3601 to VTAM, only a user-defined number of tasks to CICS/VS for the pipeline, and only one pipeline session.

Refer to Introducing the IBM 3600 Finance Communication System, GA22-2764, for further information on 3600 units and features, and to the IBM 3600/3630 Guide for information on the support of the 3600 by CICS/VS.

3650 SESSIONS

The 3651 Store Controller is a programmable control unit to which terminals may be attached for use in a retail store environment.

(Refer to IBM 3650 Retail Store System Introduction, GA22.3-475, for further information on 3650 units and features.)

The 3651 controller contains function programs (FPs) which control the operation of the various terminals attached to it. CICS/VS communicates with the FP and is not aware of individual terminals. It is the responsibility of the FP to interpret specific requests from the terminals, select a relevant session type for communication to CICS/VS, interpret the output from CICS/VS, and direct it to the appropriate terminal. The different session types which may be selected are:

- 3275 host conversational session
- 3653 host conversational session
- Pipeline session
- Application program session

3275 Host Conversational Session

This session permits a 3275 to enter transactions to be transmitted to CICS/VS and to initiate CICS/VS application programs similar to transaction entry from BTAM-supported terminals. A number of 3275 host conversational sessions can be defined in the 3651 (less than or equal to the number of 3275s).

The 3275 terminal operator requests the 3651 controller to connect him to an available 3275 host conversational session. It is then the responsibilty of the 3651 to establish the logical connection (session) between the 3275 and CICS/VS. The session is allocated an available terminal entry in the TCT, and is known to CICS/VS by the relevant TCT terminal identification.

3275 host conversational sessions permit the following CICS/VS services to be supported.

- Operator sign-on
- Basic mapping support
- Terminal paging
- Message routing and message switching
- Automatic task initiation
- Master terminal operator functions
- Supervisory terminal operator functions

The following service is not supported:

- Message recovery and resynchronization

3275 host conversational sessions are the only 3650 sessions which permit BMS maps to be stored on disk in the 3651 instead of in the processor. CICS/VS transmits the unformatted output data, plus the map name, to the 3651, which inserts device-dependent characters and, using the specification in the map, formats the output for display on the 3275.

3653 Host Conversational Session

This session permits a 3653 Point of Sale Terminal to enter transactions to be transmitted to CICS/VS, and to initiate CICS/VS application programs in a manner similar to that used for 3275 host conversational sessions. A number of 3653 host conversational sessions can be defined in the 3651 (less than or equal to the number of 3653s).

The 3653 terminal operator requests the 3651 store controller to connect him to an available 3653 host conversational session. This connection, and allocation of an available terminal entry in the TCT, is performed by the 3651 in a manner similar to that used for 3275 host conversational sessions. The session is then identified to CICS/VS by the relevant TCT terminal identification.

3653 host conversational sessions permit the following CICS/VS service to be supported:

- Basic mapping support

The following services are not supported:

- Operator sign-on
- Automatic task initiation
- Terminal paging
- Message routing and message switching
- Master terminal operator (CSMT) functions
- Supervisory terminal operator (CSST) functions
- Message recovery and resynchronization

Pipeline Session

One pipeline session may be established in a 3651 to support multiple 3653 terminals. The purpose of this session is to support minimum delay transactions from 3653 terminals, such as a credit status check and update prior to initiating a particular customer transaction, or adjustment of a customer's credit status on cancelation of a credit transaction.

A 3650 pipeline session is implemented in the same way as a 3600 pipeline session. For more information, see the description of the 3600 pipeline session earlier in this chapter, substituting references to the 3606 and 3608 terminals by 3653, and references to the 3601 controller by 3651.

Application Program Session

This session (also referred to as an Interpreter session) results in communication between CICS/VS application programs and specific application programs in the 3651. The application program session is primarily intended for the noninteractive data transfer of application-oriented information such as:

- Transaction log from 3651 disk to the processor
- Inventory receipts file from 3651 to processor
- Batch store messages and reports from the processor to 3651
- Ticket data to be used in the preparation of magnetic stripe tickets on the 3657 Ticket Unit

In most cases, where CICS/VS is involved with application programs in the 3651, the logical terminal in the 3651 is disk. In some applications, disk serves as an intermediate or staging area between the CICS/VS application program and the ultimate destination. This usage is not detected by CICS/VS.

Application program sessions may be initiated either by CICS/VS or by the 3651 controller. When initiated by CICS/VS, the CICS/VS application program issues ISSUE LOAD command to identify the particular 3651 application program with which it wishes to communicate. This 3651 program may then exchange data with the CICS/VS program or may perform some function independent of CICS/VS as specified by the user.

Another possibility is to initiate the session from the 3651 controller. This type of session occurs as the result of a user-written program requesting to establish a session with the host.

Application program sessions permit the following CICS/VS service to be supported:

- Basic mapping support

The following services are not supported:

- Operator sign-on
- Terminal paging
- Message routing and switching
- Master terminal operator (CSMT) functions
- Supervisory terminal operator (CSST) functions
- Message recovery and resynchronization

Multiple input and output messages may be transmitted between CICS/VS and 3651 application programs.

3767 SESSIONS AND 3770 INTERACTIVE SESSIONS

The 3767 is a compact, desk-top terminal that provides access to a remote System/370 processor via Systems Network Architecture (SNA).

The 3770 is a family of communication terminals that offers keyboard and printer combinations, with a selection of I/O equipment and communications features that provides a variety of multi-purpose terminal configurations in an SNA environment. There are two types of interface through which the 3770 can communicate with the host system: interactive and batch. Within the 3770 system, the interactive session described here is applicable only to the keyboard-printer station. The batch session is described separately in a subsequent section.

Communication with the host is carried out via a 3704/3705 communications controller with SDLC line discipline. With both the 3767 and the 3770, the interactive logical unit consists of a keyboard-printer only.

Interactive sessions with a 3767 or 3770 work similarly to Start-Stop sessions with a 2740 terminal in that either the terminal or the CICS/VS system can bid for control of the line.

CICS/VS functions supported for 3767 and 3770 interactive sessions are:

- Operator sign-on/sign-off
- Basic mapping support
- Terminal paging
- Message routing and message switching
- Automatic task initiation
- Master terminal operator functions
- Supervisory terminal operator functions

VTAM support for 3767 and 3770 interactive sessions differs from VTAM support of some other terminals as follows:

- 3767 is not programmable
- 3767 and 3770 interactive logical units do not employ a function management header (FMH)
- Logical device codes (LDCs) are not applicable to 3767 or 3770 interactive logical units
- The message recovery/resynchronization facility is not available
- Error messages may, in certain circumstances, be sent to the print unit.

For further information about CICS/VS and these terminal subsystems, see the CICS/VS 3767, 3770 and 6670 Guide.

3770 BATCH SESSIONS

The batch logical unit session is used for the interchange of batch data between an SNA unit such as one of the 3770 family and a CICS/VS application program.

The attachment of the 3770 will usually be on leased communications lines with other SNA products. VTAM provides a transparent interface

for the physical connection process. The 3770 in batch mode is a multi-media I/O control unit. Some 3770 I/O devices are available with different speeds (for example, 40, 80, or 120 cps printers). CICS/VS is not sensitive to I/O device speed in the VTAM/SNA environment. The use of the inter-record separator (IRS) in the standard character string (SCS), and of BMS, means that the CICS/VS application program is independent of the length of card read by the card reader.

A batch logical unit is normally concerned with sending or receiving large blocks of information. A characteristic of batch is that the data was prepared prior to initiation of transmission. Conventional batch input devices are card readers and diskettes. In both cases, the information was prepared in advance. If this information was prepared in a fashion unacceptable to the application, it cannot be immediately corrected, with the result that processing generally cannot continue.

To control problems of this type, users frequently perform an extensive edit of data prior to invoking the program that processes the data. Among the edit techniques that are employed are field content (alpha vs numeric), module check, range limits and hash totals. CICS/VS support of BLUs will offer the user the ability to control his batch operations to a level appropriate to his requirements.

The batch logical unit allows contention for control in much the same way as does the interactive logical unit. The difference is that once established, communication is normally on a strictly alternating basis: the receiving unit cannot transmit until the transmitting unit relinquishes control. As an exception, the batch protocol does allow pseudo-interactive communication with a keyboard-printer, in that an error message can be directed to the printer while batch input is taking place.

CICS/VS functions supported for 3770 batch sessions are:

- Operator sign-on/sign-off
- Basic mapping support
- Terminal paging
- Message routing and message switching
- Automatic task initiation
- Master terminal operator functions, but a batch logical unit would not normally be used as the CICS/VS master terminal
- Supervisory terminal operator functions
- Function management headers (FMHs) and logical device codes (LDCs)

VTAM support for 3770 batch sessions differs from VTAM support of some other terminals as follows:

- The message recovery/resynchronization facility is not available
- Error messages, as well as negative responses, will be sent to the terminal

For further information see the CICS/VS 3767, 3770 and 6670 Guide.

3770 PROGRAMMABLE SESSIONS

The 3770 programmable session provides facilities similar to those of the 3790 full function program session. For further information see the IBM 3767, 3770, and 6670 Guide.

3790 SESSIONS

The 3790 Communication System incorporates a Programmable Controller (3791) that allows attachment of terminals and line printers. It is used in a general-purpose data collection environment by many types of industries.

3790 programs control the operator stations and communicate with host programs such as CICS/VS. This communication operates through SNA sessions between CICS/VS and logical units in the 3791.

A 3790 terminal operator can select an appropriate 3790 program to accept data entered at the terminal, edit it, and store it on disk for later transmission to the processor. General-purpose data entry editing carried out by the 3790 program ensures that:

- Alphabetic fields contain only alphabetic characters, and numeric fields contain only numeric characters
- Fixed-length fields contain the required number of characters
- Variable-length fields do not violate the minimum or maximum length specified
- Required fields are not omitted
- Self-check digits are valid
- Numeric fields fall within a specified value range
- Field values are valid based upon application criteria such as a field's relationship to other fields, or to data held in tables or stored on 3791 disk data sets
- Field values are valid based upon the existence of required 3791 disk data set records, or the status of a particular data set record. For example, the 3790 program can ensure that an inventory update that reduces "quantity on hand" does not produce a negative quantity.

A 3790 terminal operator cannot communicate with CICS/VS to invoke CICS/VS system functions without the presence of a 3790 program or 3277 compatibility support to support the operator-CICS/VS conversation. Therefore a sample 3790 program is provided. The functions available in this program allow the use of the following facilities:

- Master terminal operator functions
- Operator sign on/sign off
- Supervisory terminal operator functions
- CSFE transaction.

The 3790 can operate completely offline, accepting and editing data from terminals, and storing it on disk for later batch transmission to the processor. It can also operate online to the processor, establishing sessions between CICS/VS and 3790 programs, which edit terminal input, pass input to CICS/VS for further processing, and accept output from CICS/VS to direct to the terminal. The 3790 is suited for remote offices where the functions of data entry, data inquiry, calculation, and document preparation are required.

Refer to An Introduction to the IBM 3790 Communication System, GA22.3-167, for further information on 3790 units and features.

CICS/VS application programs communicate with 3790 programs, and do not support or directly interact with terminals controlled by the 3790 programs. CICS/VS is not aware of any other programs which may be concurrently executing in the 3790. These programs may each separately establish sessions with CICS/VS, or with other VTAM application programs.

CICS/VS also supports the 3270 Compatibility Mode controller function in Version 6 of the 3790. This support allows existing CICS/VS applications written to use a 3270, to work in the same manner with a 3270 attached via a 3790 controller, and supported in the 3791 by the 3270 Compatibility Mode program running as a VTAM-supported logical unit.

The 3270 Compatibility Mode controller function supports the 3270 displays as its primary device (3270 printers are only supported as secondary devices, used for the production of hard copy of the contents of the screen). CICS/VS supports the 3790 controller function which supports the use of 3270 printers as bulk output devices. Here the 3270 printers are the primary device. 3270 printers attached to a 3791 may be used with either a 3270 data stream or a minimum subset SCS data stream. Both modes of use are supported by CICS/VS.

3790 Program Session

This session permits a 3790 program to accept transactions from the terminals it controls and transmit those transactions to CICS/VS. Each session is allocated the specific terminal identification of its TCT entry. Any number of CICS/VS transactions can be transmitted during a session, and each transaction can involve several input and output messages.

3790 Version 6 program sessions (full-function sessions) permit the use of the following CICS/VS services:

- Operator sign-on
- Terminal paging
- Message routing and message switching
- Automatic task initiation
- Master terminal operator functions
- Supervisory terminal operator functions
- Message recovery and resynchronization
- Basic Mapping Support

The following restrictions apply to 3790 Version 1 program sessions (inquiry sessions).

- Basic mapping support is not provided
- The 3790 program must initiate the session. The 3790 cannot accept unsolicited output from CICS/VS.
- The 3790 program must start the exchange of data with a CICS/VS transaction by issuing a PUT, and must issue a GET to receive the last output from the CICS/VS transaction.
- The maximum input message is 240 bytes. However, several input messages can be transmitted by the program and be concatenated by the CICS/VS application program (through user programming) for input greater than 240 characters.
- The program is responsible for the unchaining of chained output.

Refer to the CICS/VS IBM 3790 Guide for further information on the use of the 3790 by CICS/VS.

LUTYPE4 SESSIONS

Logical units type 4 (LUTYPE4) have been defined by SNA to allow the transmission of word processing data streams. The IBM 6670 Information Distribution System is an LUTYPE4. CICS/VS supports sessions between CICS/VS as a primary logical unit and the IBM 6670 as a secondary logical unit. The CICS/VS support is described in CICS/VS IBM 3767, 3770, and 6670 Guide.

CICS/VS application programs can read from and write to data processing media (console, card, and print) controlled by an LUTYPE4, as well as word processing media. All these media are capable of both input and output, except for print. In a typical case, the application would read from magnetic cards and write to a printer.

CICS/VS support for LUTYPE4 is an extension of, and is compatible with, support for batch logical units, using basic mapping support and the batch data interchange interfaces. In a typical application the data transmitted consists of the contents of documents, this data being stored, under the control of CICS/VS, on a central data base at the host CPU. Documents could be both stored and transmitted to the LUTYPE4 in unformatted or partially formatted form, to be formatted and then printed by the logical unit.

Documents would be prepared on magnetic cards, probably using a magnetic card typewriter. The operator of the LUTYPE4 would feed the cards into the unit, which would then transmit the data to the CICS/VS application programs that are to store it.

Basic Mapping Support (BMS) functions are provided for the output of word processing data streams, but not for input. BMS supports data processing media for both input and output.

Terminal Control Communication Using VTAM

Each of the basic terminal control operations, RECEIVE, SEND, SEND WAIT, WAIT, and CONVERSE, is available for communication with TCAM- or VTAM-supported terminals.

TERMINAL I/O OVERLAP

VTAM permits terminal I/O operations to proceed concurrently with application program processing. This enables terminal I/O to be overlapped with CICS/VS application program processing. The CICS/VS application programmer can specify whether a terminal control request is to be initiated immediately to communicate with a VTAM-supported terminal while application program processing continues. The programmer can check for completion of the request at a later time by issuing a WAIT TERMINAL command.

Alternatively, the programmer may specify that a terminal control request is not to be initiated immediately, but is to be delayed until the program issues a WAIT TERMINAL command, passes through a user-defined synchronization point, or terminates. Delayed initiation of VTAM terminal control requests provides compatibility with the manner in which BTAM-supported terminals are controlled. Further, it can ensure that no output is transmitted to a logical unit until the task which generated that output is no longer vulnerable to backout in the event of a system failure.

FULL-DUPLEX TRANSMISSION

VTAM supports full-duplex transmission of data between the processor and logical units. Thus, input and output may be proceeding concurrently on the same line, to different controllers multi-dropped on that line. CICS/VS enables the application program to request terminal input when needed, and to direct terminal output to the relevant terminal or logical unit when processed (half-duplex processing).

Although CICS/VS application programs process in a half-duplex mode, for optimum line efficiency data can still be transmitted by VTAM in full-duplex mode. Logical units may transmit input to CICS/VS application programs on an anticipatory basis. VTAM queues this input until the relevant CICS/VS application program issues a RECEIVE command. The input message has already reached the processor, and is then presented to the application program to satisfy the request.

FUNCTION MANAGEMENT HEADER

The following discussion on the function management header (FMH) is applicable only to the programmable terminal systems 3600, 3650, and 3790, and to the 3770, the 3770 Programmable and the 6670.

Output messages transmitted by terminal control to particular terminals within these systems may require certain information within the message to identify the disposition of the output by the logical unit. This information is called a function management header and comprises the first part of the output message.

Similarly, input messages from terminals may include an FMH. In this case, the FMH supplies information about its origin for the application program.

See the appropiate terminal guides and the CICS/VS Application Programmer's Reference Manual for further details.

SYSTEM PROGRAMMER MACRO INSTRUCTIONS

Additional terminal control macro instructions are available for use by the CICS/VS system programmer. These enable the status of a terminal to be changed in the TCT, or a specific terminal entry to be located in the TCT, using the terminal control STATUS and LOCATE macro instructions respectively. The result of these operations can be checked using the terminal control CHECK macro instruction.

The STATUS, LOCATE, and CHECK terminal control macro instructions are intended primarily to be used by the system programmer in the network error program DFHZNEP. The DFHZNEP program enables the system programmer to attempt recovery of error exception conditions encountered with transmission to VTAM-supported terminals.

See the appropriate terminal guides and the CICS/VS System Programmer's Reference Manual, for further details on these system programmer macro instructions.

Basic Mapping Support Communication with VTAM

The benefits of format (mapping) and terminal device independence offered by BMS to BTAM-supported terminals are also available to VTAM-supported terminals.

INPUT MAPPING

BMS performs input mapping for 3270, 3650, 3767, 3770, and 3790 Version 6 systems and for data streams from 6670 data processing media (console, card, and print); it does not perform input mapping for 3600 or 3790 Version 1 systems. In 3600, and 3790 systems, and in 6670 word processing data streams, the application program associated with the logical unit is responsible for removing device-dependent characters from the terminal input message. It is also responsible for formatting input data prior to its transmission to CICS/VS. Received data is passed to the application program without change following a RECEIVE MAP command.

OUTPUT MAPPING

BMS performs output mapping for VTAM-supported terminals (except for 3790 Version 1). Device-dependent characters are inserted into output messages by BMS based upon the characteristics of the device intended to receive the output.

Where appropriate, BMS constructs and inserts the function management header (FMH) into the output message prior to issuing terminal control output requests on behalf of the CICS/VS application program. The FMH has the same format as described in "Terminal Control Communication with VTAM." The message description byte is set up by BMS to define a formatted output message. The CICS/VS application program identifies the output device to BMS for VTAM-supported terminals by means of the logical device code (LDC). The LDC is used by BMS to identify the page size and device type. BMS inserts information from the LDC in the FMH prior to requesting terminal control output. The CICS/VS application program is relieved of the responsibility of constructing the FMH. This permits programs to be written independent of particular terminal characteristics.

LOGICAL DEVICE CODE USES

The following discussion on the logical device code (LDC) is applicable only to terminal systems that use a function management header.

The LDC may be used to identify the output device to BMS and the logical unit. BMS uses the map specified by the application program to format the output data. It inserts the device-dependent characters into the output stream to ensure that the data is displayed as specified by the map. The LDC is also inserted into the output stream for transmission to the logical unit. On receipt of the data, the logical unit determines (from the LDC) which device is to receive the output.

In addition to identifying specific devices and their associated page size, the LDC can also communicate other information to the controller application program. For example, the LDC may identify a specific preprinted form number to receive the output on a specific printer, or on any printer available to that logical unit. The LDC may also be interpreted by the logical unit as a request to turn on (or off) particular terminal indicator lights, transmit data accumulated on disk during offline operation to the processor, or change processing modes (for example, change to after-hours processing operation).

The LDC is used by the CICS/VS application program to communicate logical disposition of output to the logical unit, and can represent any logical meaning useful to the installation's purpose.

CICS/VS permits the use of as many as 255 different logical device codes. Each may have a different meaning, dependent upon the particular controller application program interpretation.

MAP RESIDENCE IN CONTROLLERS

Some VTAM-supported controllers, such as the 3651, permit formats to reside outside the processor on disk in the controller for 3275 host conversational sessions. A BMS output request from a CICS/VS application program results in transmission of the output data (without mapping) and the format name in the FMH, to the remote controller. The controller retrieves the specified format from the 3651 disk, and writes it to the screen on the relevant 3275 attached to the 3651. Thus, processor processing is reduced, and additional flexibility is available to the installation to tailor a general purpose map to specific 3650 systems in individual retail stores.

BMS ALARM INDICATOR

The CICS/VS application program, using BMS, can request that an alarm indicator be turned on at the terminal upon receipt of the output message. This alarm indication is transmitted by BMS to the logical unit by means of the function management header (FMH). The logical unit responds to this request by turning on an appropriate indicator light on the terminal that is to receive the output.

BMS I/O OVERLAP

The CICS/VS application program can request that a BMS operation, and the associated terminal I/O, be initiated immediately. Alternatively, the program can request that the BMS operation be delayed until a WAIT TERMINAL command is executed, the program passes through a user-defined synchronization point, or the program terminates. This immediate, or delayed, request is specified as part of the SEND or RECEIVE command in the manner described in "Terminal Control Communication using VTAM." It has the same purpose as for terminal control: to provide compatibility with BTAM operation and to improve output message integrity.

Terminal Device Independence with VTAM and BTAM

The use of BMS permits CICS/VS application programs to be written independent of the particular terminal to be used. For VTAM-supported terminals, default options provide compatibility with BTAM-supported terminal operation. For example, the default option (unless specified otherwise) is to delay the initiation of terminal I/O until the application program issues a WAIT TERMINAL command, passes through a user-defined synchronization point, or terminates.

If an LDC is not specified in a BMS request to VTAM-supported terminals which require the LDC parameter to be used, a default value is used. This can be specified when the TCT is generated as a unique value for a specific TCT entry, for a group of entries, or for all entries in the TCT. If, however, an LDC is specified in a BMS request to BTAM-supported terminals, it is ignored.

BMS requests which specify 3270 attribute characteristics can be used with devices that do not have these characteristics. In this case, the 3270 attribute information is ignored.

For these reasons CICS/VS application programs can be written to communicate with a variety of BTAM-, VTAM-supported terminals. Unique device characteristics may be specified in a map generated specifically for a terminal to achieve a general format function required by the CICS/VS application program. The CICS/VS application program identifies a map name in its BMS request; BMS appends to that map name a character which identifies the particular terminal type which is communicating with the program. BMS then retrieves the unique device-dependent map for that terminal type and uses it to format the terminal data.

Consequently, existing BTAM-supported terminals may be used to test CICS/VS application programs intended primarily for operation with VTAM-supported terminals before the installation of the VTAM or TCAM terminals. Alternatively, sequential terminals, such as card reader/line printer, disk, or tape, may be used for testing. However, testing must model the operation of remote controller programs. For

example, input must be presented to CICS/VS in exactly the same format as would be presented by the remote controller. Output must be accepted from CICS/VS as presented to the remote controller. In the case of input mapping for testing for the 3600, the mapped input from BTAM-supported or sequential terminals must be the same as presented by the 3601. This is necessary because mapping with actual 3600 input is ignored, and the input data is presented to the application program without change.

Testing is limited to those operations which can be performed by the relevant testing terminal, or which can be preplanned in the test input. BTAM-supported and sequential terminals are unable to exercise the same data handling and processing capability possible with remote controllers.

TERMINAL PAGING USING VTAM

Some sessions established with remote controllers support terminal paging. (See "CICS/VS Session Types.") The CICS/VS application program can build pages to be associated with specific logical device codes used by a logical unit. BMS separately controls the page construction for each LDC, and then makes available all pages built for each LDC used by the logical unit.

The terminal operator at the remote controller can request that pages associated with a particular LDC for that logical unit be displayed. (See Figure 3.2-3 for terminal paging commands.) The appropriate LDC pages desired can be requested by appending the LDC to the terminal paging command; all LDC pages can then be displayed on request. However, any LDC pages which have not been viewed will be lost when the terminal operator requests purging of pages associated with the logical unit.

MESSAGE ROUTING AND MESSAGE SWITCHING USING VTAM

Some sessions established with remote controllers support both message routing and message switching. (See "CICS/VS Session Types.") Pages, built by CICS/VS application programs or by terminal operators, can be associated with particular logical device codes for transmission to one, or a group, of logical units. The only restriction is that each LDC associated with a message-routing or switching request must represent the same device type.

Pages delivered to the specified logical units can be viewed by the terminal operator using the appropriate LDC appended to the terminal paging commands.

Part 4. Application Design

Chapter 4.1. Program Design

Task Initiation

There are several methods utilized by CICS/VS to initiate tasks.

- Transaction codes
- Automatic task initiation
- Interval control
- Task control

They are outlined here.

TRANSACTION CODES

CICS/VS examines a transaction code received as part of a terminal message, to identify the particular transaction involved. This transaction code must occupy the first one to four characters of the transaction invocation message. An input message is considered a transaction invocation when it occurs and no task is active on the terminal. This transaction code is validated by CICS/VS against a program control table (PCT). If the specific transaction code exists in that table, the transaction is assumed to be a valid one, and the transaction is passed to that program identified in the relevant PCT entry for processing (see Part 3 for more detail).

The 3270 enables transactions to be initiated by the use of a Program Attention (PA) key, Program Function (PF) key, selector light pen, cursor select key, or operator identification badge.

Note: Users are advised not to use transaction codes beginning with 'C'. All CICS/VS internal transactions begin with C, and it is possible that these will be added to in the future. If a user has a transaction name which is identical to the name of a newly introduced internal transaction, he will have to change his transaction name and associated documentation.

AUTOMATIC TRANSACTION INITIATION

Automatic transaction initiation involves the queuing of transactions on disk using CICS/VS transient data management. A number of transactions may be queued based upon a specific trigger level. When the number of transactions queued reaches this trigger level, CICS/VS automatically utilizes a specified transaction code for that queue to initiate a task and allow those queued transactions to be processed by a specific program. (See Chapter 4.2.)

INTERVAL CONTROL

CICS/VS enables a task to be initiated using a specified transaction code at some future time, based upon time of day or on elapsed time. Data may be passed to that future task for use in processing when it has been initiated. (See "Interval Control" later in this chapter.)

Program Control

A program may pass control to other programs in a variety of ways. These are illustrated in Figure 4.1-1 and described briefly below. See the CICS/VS Application Programmer's Reference Manual, for additional information.

TRANSFER CONTROL TO PROGRAM (XCTL)

The XCTL command enables one application program (referred to as the calling program) to pass control to another application program (referred to as the called program). Control is not returned to the calling program on completion of execution of the called program.

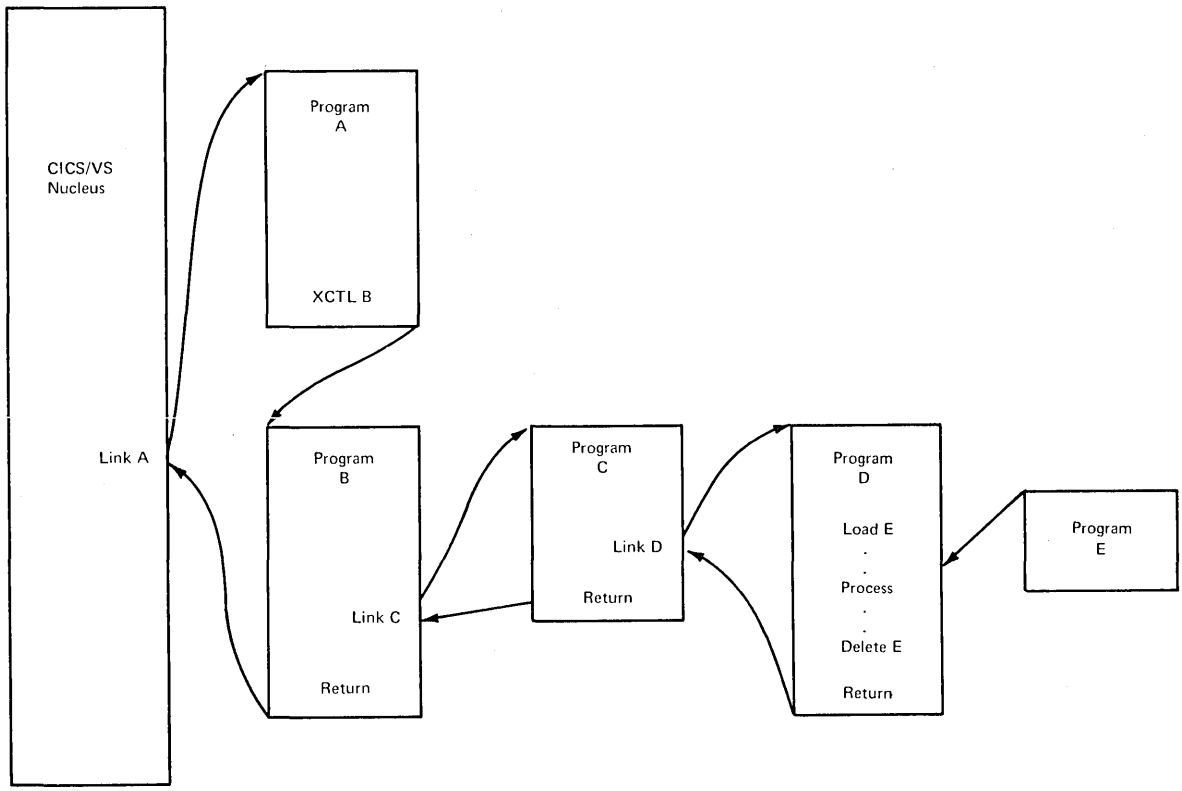


Figure 4.1-1. CICS/VS Program Control Facilities

LINK TO PROGRAM (LINK)

The LINK command specifies the name of an application program to be executed. The calling program passes control to the called program. On completion of execution, control is returned to the calling program, to the statement following the LINK.

LOAD PROGRAM (LOAD)

This command enables a program, identified by name, to be loaded into storage. However, control is not passed to that program for execution, but is returned to the statement following the LOAD command. The LOAD command can be used to load application programs, which may subsequently be linked to or transferred to. Alternatively, this command may be used to load tables of information.

DELETE PROGRAM (RELEASE)

Normally, on completion of execution of a task, all storage utilized by that task is automatically freed and made available for use in processing other transactions. Active programs being used by tasks will continue to reside in storage. If, however, the storage occupied by an inactive program is required for some other executing task, that storage will be freed. When a program is loaded, its storage can be automatically freed if it is currently inactive, allowing another task to use that storage.

Alternatively, the LOAD command can specify that the storage is not to be freed, but that the program is to remain resident in storage even though inactive, for performance reasons.

The RELEASE command is used to delete such a resident program at a point which will enable its storage to be utilized for other functions.

RETURN FROM PROGRAM (RETURN)

The Program Control RETURN command enables a program to return control to a higher level program, that program can be either another application program or CICS/VS if the RETURN is issued by the application program at the highest level. The RETURN indicates that the relevant program has now completed processing. At task completion, CICS/VS frees all of the storage associated with the task, such as terminal I/O areas, file I/O areas, and file work areas, and eventually also frees the storage occupied by the task control area. Optionally, a transaction code may be specified in the RETURN command. The transaction code is used with the next input message from the same terminal that originated this completed task.

ABNORMALLY TERMINATE PROGRAM (ABEND)

This command enables a program to immediately terminate execution of a task, with an optional dump if required. In conjunction with the optional operator sign-on facility, the ABEND command can be used to develop operator error statistics.

ABNORMAL TERMINATION EXIT (HANDLE ABEND)

This command enables a task to activate, deactivate, or reestablish a program-provided exit routine to be executed in event of abnormal termination of the task. This exit routine can be utilized either on CICS/VS abnormal task termination, or by termination through the use of the ABEND command by the task.

An abnormal termination exit routine is used to complete urgent processing by a task for recovery purposes, or it may attempt recovery of the particular error condition itself. Refer to "Program Error Recovery" in the following section for more detail.

Task Control

Task allows CICS/VS application programs to attach new tasks for execution, if required. This may be done either by the automatic task initiation feature of transient data intrapartition queues, or by time-ordered initiation (see "Interval Control," below).

SUSPEND

The SUSPEND command enables a task to wait on completion of a single event or one event in a list of events. However, the SUSPEND may first result in task switching to another CICS/VS task which is able to process. Only if no CICS/VS task is able to process is control voluntarily passed to another VSE or OS/VS partition/region.

TERMINAL READ TIMEOUT

This feature allows the user to specify a timeout limit for a conversational transaction when the transaction is waiting for a terminal input message. This keeps a single transaction from occupying system resources for long periods of time while waiting for a reply from the terminal.

ISOLATED TASK PAGING

This option allows the user to participate (with the operating systems page manager) in selecting pages to be made available for pageout. When the TCA storage of a private or long running task is acquired by task control, a number of additional pages may be acquired as specified by the ANTICPG operand in the task's PCT entry. Task control makes these pages available for pageout when the task waits for a response from a terminal, and causes them to be asynchronously paged in when the task is to be given control after the response has been received. This allows pages occupied by data areas belonging to conversational tasks to be paged out faster than the normal paging process.

ENQUEUE/DEQUEUE

Task control provides ENQ and DEQ commands for other CICS/VS modules and application programs to enqueue and dequeue on various resources.

An enqueue and dequeue facility is often necessary in a multitasking environment, to ensure that only one task is able to utilize a particular resource at a time. In the case of the potential concurrent updating of records in a file control data set, file control utilizes the task control ENQ and DEQ facilities to ensure that the first task that retrieves a particular record for update is given the exclusive use of that physical record. A second or subsequent task which also wishes to update that same physical record while the first task is still in the process of updating it, is prevented from carrying out its update. When the first task has completed its update, and is dequeued from exclusive control of that record, the next task is given exclusive control of the record through the use of ENQ. It carries out its update until it has completed, and then is dequeued from that record.

Interval Control

FUTURE TASK INITIATION

Tasks can be initiated at a future time, based on either an elapsed period of time or a specific time of day. These future tasks can be initiated either with no data transfer through the use of the START command, or with data transfer through the use of the START command with the FROM option. A unique time request identification can be allocated to the START command, to later identify that request if it is necessary to cancel the request before the task is initiated.

The use of START FROM allows a task to be initiated at a future time, based on elapsed time or time of day, and specify data to be transferred to that task. This time request is given a unique identification, and data which is to be transferred to the future task is written by interval control to a special queue. There is one such queue for each future task/terminal combination.

When the future task is initiated, its data may be retrieved from the queue by the application program issuing a RETRIEVE command. Subsequent RETRIEVE commands will retrieve each record which was initially put onto the queue for this task/terminal combination, until all expired records have been retrieved. At this point, the ENDDATA condition is raised,

unless the WAIT operand was specified on the RETRIEVE command. WAIT will cause the task to be suspended until a further START command in some other transaction supplies further data for this task and terminal. If CICS/VS shuts down or receives a request to start another transaction for the same terminal before more data is supplied, the suspension will cease and ENDDATA will be raised.

The task will be terminated if te suspension lasts longer than the value specified in the DTIMOUT operand of the DFHPCT TYPE=ENTRY macro. It is recommended that a limit is specified in case an error occurs that prevents more data being supplied.

This facility is useful when designing online applications, to ensure that particular application events which must occur at a specific time of day, or after a specific elapsed time following some other application action, can be initiated automatically. However, procedures must be developed during online system design to cancel these future application events, if necessary for the application. Generally, canceling of future events would be placed under control of the master terminal operator, through user-written programs which are given the same security code as that allocated to the master terminal operator.

The START command is provided with a number of facilities intended for use primarily in communications between transactions running in different systems. Information is given in Chapter 7.2, "Function Request Shipping and Transaction Routing". Most of the facilities are applicable when the transactions run in one system, though they would not generally be very useful.

TIME EVENT WAIT

Application programs may need to wait on completion of a specific period of time or until a specific time of day occurs. This can be achieved by the use of the SUSPEND command, specifying the request identification of the original command which specified the particular time request.

The use of SUSPEND will utilize CICS/VS resources (such as dynamic storage for residence of the particular program and associated areas), until the SUSPEND is satisfied. Accordingly, the SUSPEND command should not be used unless the time duration is only a matter of seconds. If it is necessary for a longer duration wait to be in effect, this should be achieved by initiating a future task at some short interval of time before the time event to be waited on expires. This future task may issue the SUSPEND command to wait for completion of the specified time event, and so ensure that CICS/VS dynamic storage is tied up for the shortest possible time. To minimize the effect of utilizing these resources, that future task should occupy as little storage as possible.

Application programs may also request the time of day through the use of an ASKTIME command.

TIME EVENT CANCEL

As indicated previously, it may be necessary for future time events to be canceled. This is best achieved by user-written application programs which issue the CANCEL command, specifying the time event request identification for that event to be canceled. Ideally, these canceled transactions should be placed under control of the master terminal operator, by allocating a security code to the transaction code so that it can be utilized only by the master terminal operator.

Program Error Recovery

CICS/VS features facilities for detection of program error situations, and protection of the online system from the effect of these errors. If a program-check error is detected in an application program, CICS/VS will attempt to abnormally terminate that task while still permitting other tasks to continue processing.

For fuller details of recovery and restart facilities, refer to Part 5.

PROGRAM ERROR PROCESSING

CICS/VS enables an application program to indicate, through the use of the HANDLE ABEND command, that control is to be passed to a specified routine in the program in the event of a program error, or to another program. This routine (or program) may attempt recovery from the error, record certain critical information necessary to the application before the error program is abnormally terminated, or ignore the error and continue program processing.

DYNAMIC TRANSACTION BACKOUT

If user-recovery is not successful and the abend is to continue, the Dynamic Transaction Backout (DTB) Program will be invoked, if specified for that transaction, before going to the Program Error Program.

TRANSACTION RESTART

Transaction restart is a facility which will restart an abended transaction without requiring the attention of an operator. This facility requires the dynamic transaction backout feature.

In the event that a program error requires abnormal termination of a task, the terminal operator who invoked the transaction to be abnormally terminated will be notified of this fact by CICS/VS, if appropriate. CICS/VS passes control to a program error program (PEP), after all HANDLE ABEND processing has been completed for the task and the decision has been made to permit the abnormal termination to continue. This PEP is a user-written routine, given control through a LINK from the CICS/VS abnormal condition program (ACP), which enables the user to perform installation-level abnormal termination action.

Quasi-reentrant Programming

For efficient utilization of storage, CICS/VS ensures (unless requested otherwise) that only one copy of a program will reside in the dynamic storage area. All tasks requiring the use of that program are able to execute that program concurrently.

In order to achieve a high degree of multitasking, CICS/VS supports quasi-reentrancy. This allows several tasks to utilize the same section of code over the same period of time. However, it differs from fully reentrant programming in that control is only passed from one task to another when the active task issues a CICS/VS command. Control will not pass from one task to another on an I/O interrupt, for example, as is the case in a VSE or OS/VS multitasking environment. CICS/VS provides a quasi-reentrant capability for Assembler, ANS COBOL, and PL/I but not RPG II.

Information unique to the processing of a transaction (such as the terminal input area, file I/O areas, or work areas) is separated from the body of the application program. Instead of these areas residing within the program, they are allocated from dynamic storage. The execution of each separate transaction in a multitasking environment is controlled by a task control area (TCA) that contains address pointers and other vital information for that particular transaction (task). Because the information unique to a task is separated from the main body of a program, the program can be used concurrently by several tasks.

When the command-level interface is used, dynamic storage is defined in the high-level programming language, but at execution time, allocation takes place in storage areas controlled by CICS/VS, and chained off the TCA, rather than in areas allocated by the language processor. This is achieved by special code in the language processor in the case of PL/I and COBOL; it is achieved by allocating a new copy of the program for each task in the case of RPG and it is achieved by user programming in the case of assembler.

The access methods are incorporated within the CICS/VS nucleus, and exception or error routines are included in other CICS/VS application programs. Figure 4.1-2 shows the concept of quasi-reentrant programming.

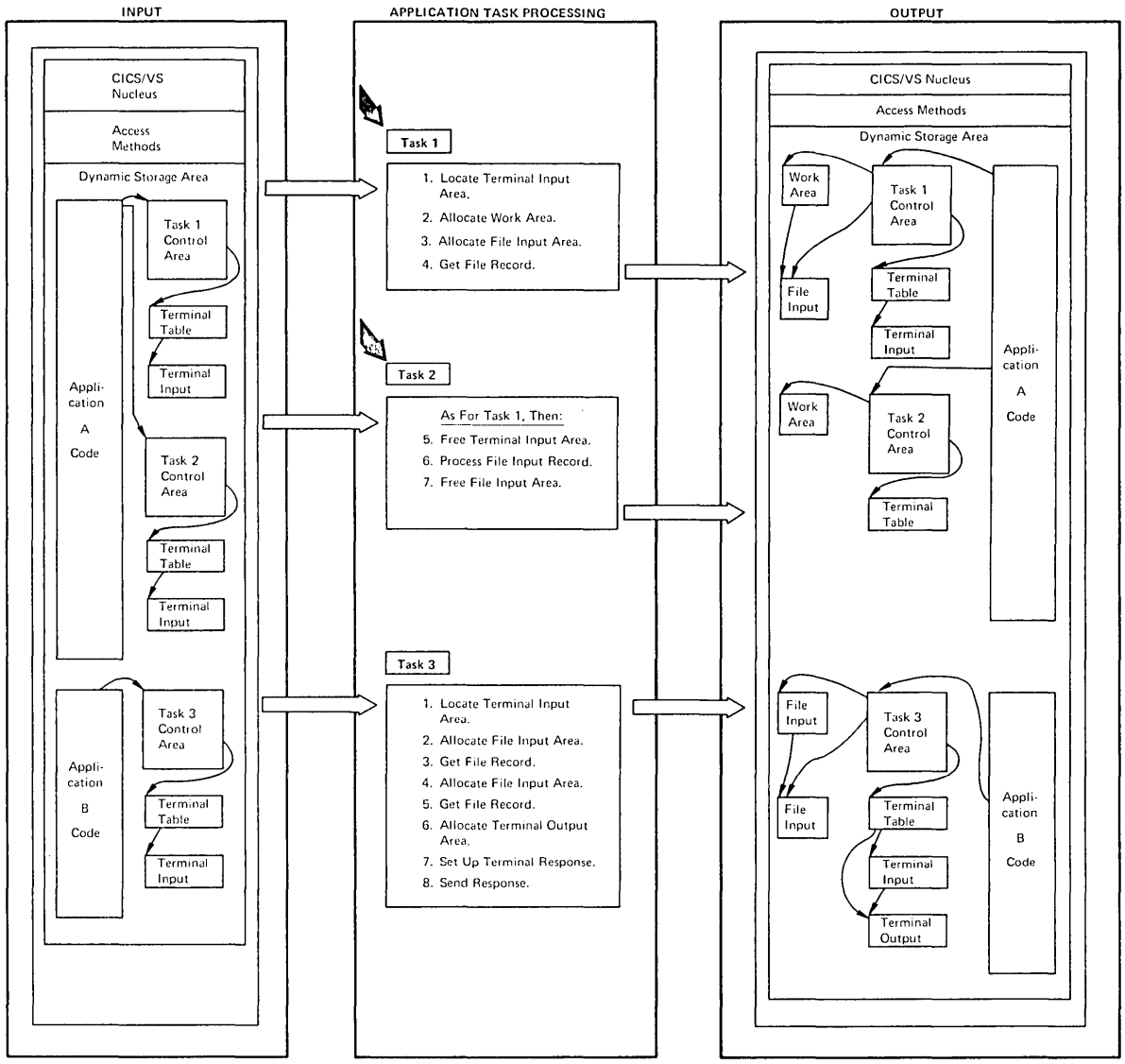


Figure 4.1-2. Quasi-Reentrant Programming and Multitasking

Chapter 4.2. Data Management Design

CICS/VS, together with DL/I, provides extensive data base capability to online applications. In addition to this data base capability (which is discussed in Part 2), CICS/VS offers additional facilities for internal data management. This chapter first identifies various application requirements which demand the services offered by CICS/VS temporary storage management and transient data management. It then describes those services which can be used as design tools by the system designer to satisfy his own application requirements.

Application Requirements

It is first necessary to define the various data management functions (as distinct from data base capability) which online applications require of a DB/DC system. These functions are briefly described below.

WORK FILE CAPABILITY

Most online applications require the ability to store information for later retrieval and use. This function is sometimes referred to as a "scratchpad" or work file capability, and is analogous to a person using sheets of paper to jot down the intermediate results of calculations for later use in processing.

The following are two main work file requirements used for most online applications:

- Scratchpad capability
- Queuing capability

Scratchpad Capability

This capability refers to the temporary storage of information for later retrieval. In a batch environment, this capability is often provided through the use of work data sets. In CICS/VS, this capability is provided by the CICS/VS temporary storage control program. The application program identifies data which is to be temporarily stored by name, and subsequently retrieved by name without any consideration of its physical location. Online application uses of temporary storage include the following.

- Intermediate Results: The storage of intermediate results developed during the processing of a transaction, for use later in the processing of that transaction.
- Error Correction: The storage of input transactions which were found to be in error, for subsequent use when the corrected error fields are received from the terminal.

- Data Transfer: A temporary storage of data so it can be used to transfer data between programs. This data transfer may occur immediately or at some future time.

- Terminal Paging: An application program may develop several pages of information to be displayed at a terminal. This information should be temporarily stored until the terminal operator requests that it be displayed for his attention.

Queuing Capability

In addition to the temporary storage of data, online applications generally require a facility which will enable data to be queued for subsequent processing. The difference between this and temporary storage is that temporary storage stores and retrieves individual sections of data, while a queuing capability enables several different sections of the same type of data to be queued, and then all sections retrieved together, sequentially, in the order that they were queued. In CICS/VS, this queuing capability is provided by the CICS/VS transient data control program. Examples in which online applications may utilize a queuing capability follow:

- Batch Transaction Processing: Transactions of a particular type may be received from many terminals. If the application requires that all of these transactions be processed together, they may be stored in a unique queue for that transaction type, in the order that they reach the processor. This queue of transactions may then be processed in the CICS/VS partition as a small sequential group of transactions.

- Batched Message Transmission: The online application may require that messages be batched and transmitted to specific terminals.

- Batch Partition Data Transfer: The online application may require that information be transferred to batch partitions for further processing, and the results of that processing be provided to the online application for input at a later time.

CICS/VS/VS Temporary Storage

The temporary storage management facility of CICS/VS provides a scratchpad capability for online application programs. It enables data to be stored either in dynamic storage or on auxiliary storage. Data to be stored can be identified symbolically, and retrieved symbolically, without application programs being concerned with the actual physical location of that data. Data can be retrieved on request by an application program in either a sequential or a direct access manner. Temporary storage allows records to be up to 32,000 bytes in length, but supports variable-length records only.

TEMPORARY STORAGE USAGE

The previous discussion of application requirements identified the general use of a scratchpad facility by application programs. CICS/VS temporary storage management is used to meet these application requirements as follows.

Data Transfer Facility

The ability to temporarily save data for later use, and retrieve it symbolically by name at a future time, enables easier implementation of complex processing. This complex processing may be broken into several logical steps, each step carried out by a separate module. Information may be passed between these modules using temporary storage.

Depending upon the amount of information to be passed, and the time period before that information will be used, this data may be stored either in dynamic storage or, alternatively, in auxiliary storage.

Scratchpad Facility

Temporary storage may be used to save information for later use. An example would be the saving of error transactions for later combination with corrected fields received from a terminal, as described in "Error Correction". Using this capability, correct information in the original error transaction does not have to be reentered by the terminal operator. Consequently, error correction is easier, and the potential for further operator errors is reduced.

Terminal Paging

Terminal paging in CICS/VS is also supported through the use of temporary storage. Pages of information developed by application programs are presented by them to CICS/VS. These pages are stored in temporary storage for transmission to the terminal operator on request. Refer to "Terminal Paging" in Chapter 3.2 for more detail.

Message Routing

The ability to transmit messages from one terminal to another terminal, using the CICS/VS message switching transaction CMSG, or the ROUTE command, is supported through the use of temporary storage. These messages are automatically transmitted to the relevant terminal when that terminal is able to receive them, or the specified operator has signed on to CICS/VS. Refer to "Message Routing" in Chapter 3.2 for more detail.

Interval Control

The CICS/VS interval control program uses temporary storage to pass data from one task to another task which is to be initiated at a future time. An application program may indicate the task to be initiated at a specified time (based on elapsed time or, alternatively, time of day) and may transfer data to that future task. The FROM option of the START command results in the data to be transferred being written to temporary storage on disk for subsequent retrieval by the RETRIEVE command. Refer to "Interval Control" in Chapter 4.1 for more detail.

Execution Diagnostic Facility

The CICS/VS Execution Diagnostic facility uses temporary storage to communicate between different invocations of the debugging task and to maintain a trace of the execution of the user task. Refer to "Execution Diagnostic Facility" in Chapter 4.3 for more detail.

The Routing Transaction

The CICS/VS Routing Transaction (CRTE) uses temporary storage to retain information about a routing session during its pseudo-conversation with the operator terminal. Refer to "CRTE Routing Transaction" in Chapter 7.2 for more detail.

DATA IDENTIFICATION

Each record may be presented to temporary storage with a unique eight-character data identification. Alternatively, several records may be presented with the same data identification. In both cases, the identification is supplied in the QUEUE operand of the WRITEQ TS command, the former case being regarded as a queue with only one element. A queue of records associated with a particular logical function (as indicated by the queue name) can be developed, and subsequently retrieved in the same sequence or a different one.

The queue name is used by CICS/VS to develop a data element which contains that name, the sequence or entry number of the record in a queue of records with the same data identification, and the location of the record either in dynamic storage or on disk. These data elements are maintained in CICS/VS dynamic storage. As records are written to temporary storage, data elements are dynamically built by CICS/VS and saved in dynamic storage. The number of temporary storage records which may be retained is limited only by the availability of dynamic storage and/or the amount of disk space allocated to the temporary storage data set.

Because many tasks may concurrently use the same program, the use of a constant in the program for identification of individual records is not advisable. The queue name may be dynamically generated by the program based upon information such as:

- A combination of transaction identification (four characters) and operator identification (three characters) will enable that operator to store one record at a time for each transaction identification.

- A combination of operator identification and time of day, or transaction identification and time of day, will enable the record to be uniquely identified. However, it requires the application program to determine the time of day and then respond to the terminal operator with the allocated data identification. He may then use it to uniquely identify the record in a later transaction.

The techniques for unique data identification described above assume an application environment where information is to be stored by the user's program, and directly retrieved at some future time under control of the terminal operator.

If temporary storage is used to pass data from one application program to another, the allocated data identification may be passed to a subsequent application program (executed under control of the same task) through the transaction work area (TWA) appended to the task control area (TCA) for that task. The program (executing under the same TCA) which is to retrieve the data from temporary storage can obtain the allocated data identification from the TWA. This data identification is then used to identify the record to be retrieved.

If a record within a temporary storage queue is to be directly retrieved, it must be uniquely referenced by the queue name and its relevant entry (or sequence) number. When a record is written to a temporary storage queue, it is placed at the end of the queue of records with that same data identification. Temporary storage management will allocate the next sequential entry number and return this entry number to the program. The record is now uniquely identified by the queue name and the entry number.

This queue name and entry number may be transmitted to a terminal operator for subsequent reentry, if the retrieval is to be initiated by the terminal operator. If the retrieval is to be initiated automatically by subsequent application programs executed by the same task, the queue name and entry number should be saved in the TWA. The program which is to retrieve that unique record may then extract this information from the TWA for use.

USE OF DYNAMIC STORAGE BY TEMPORARY STORAGE

Dynamic storage is a valuable resource, and the overall performance of the online system is directly related to the amount of available dynamic storage and its relationship to real storage available for use as a virtual storage page pool.

Generally, dynamic storage residence of records should be used only when the life of those records is to be of very short duration. Its main purpose is in passing data between program modules which are executed under control of the same task. Once the data has been passed between modules through dynamic storage, that data should be deleted and the storage occupied by it freed. Dynamic storage may be used for record queues as well as unique entries; however, write requests to dynamic and auxiliary storage with the same data identification cannot be used. CICS/VS will force all subsequent write requests with the same data identification to use the same storage facility specified by the first request.

The length of records to be stored in dynamic storage may be up to the VSAM control interval size specified during CICS/VS system initialization, less 84 bytes for CICS/VS control information.

CICS/VS permits temporary storage records to reside in dynamic storage whether or not the CICS/VS system is generated indicating no auxiliary storage residence support is required. The specification of no auxiliary storage support removes the requirement for VSAM by temporary storage. Instead, virtual storage is utilized; temporary storage information is only paged into real storage when referenced.

Any temporary storage information residing in dynamic storage is lost if a controlled or uncontrolled shutdown occurs. Auxiliary temporary storage data IDs may be specified as recoverable to provide recovery of records in the event of either a particular task failure or a total system failure. See the CICS/VS System Programmer's Reference Manual and Part 5 for additional information.

As a general rule, if a record must be stored for more than one second, it should be directed to auxiliary or secondary storage rather than to dynamic or main storage. Dynamic storage is then available as much as possible for use in initiating concurrently executed tasks. Certainly, the writing of records to disk, and the subsequent retrieval from disk, will involve file accesses and so increase the processing time of those particular tasks. However, the overall effect on the entire online system is one of potentially better performance than would result if considerable dynamic storage were utilized for temporary storage residence.

ACCESSING RECORDS IN TEMPORARY STORAGE

Temporary storage supports variable-length records only. A queue or message set of records may be developed by issuing a WRITEQ TS command for each record, using the same queue name. As each record is written, temporary storage allocates the next sequential entry number and returns it to the application program.

Using the queue name and the entry number, the records in the queue can be retrieved by application programs either sequentially, in the chronological order in which they were written, or directly accessed by referencing a specific entry number.

A queue of records can be retrieved sequentially by specifying the queue name allocated for that queue and issuing a READQ TS command. Temporary storage management retrieves the first record in the queue and presents it to the application program. Each subsequent READQ TS retrieves the next record in sequence until the last record has been retrieved, when an end-of-queue indication will be returned to the program.

Alternatively, if it is required to commence sequential retrieval, not from the beginning of the queue but from a logical point within the queue, both the queue name and the entry number are specified by the program. READQ TS commands are then issued to retrieve each record sequentially from the logical starting point in the queue.

The program may directly retrieve records by issuing a READQ TS command with the specific entry number of the record in a queue to be directly retrieved.

A record can subsequently be updated by issuing a WRITEQ TS command specifying the relevant entry number.

The facilities offered by temporary storage for direct and sequential retrieval of information make it a powerful work file capability for online applications. Information may be retrieved as often as required until it is no longer needed. At that time, the records may be deleted.

Queues of records may be purged by a DELETEQ TS command. The deletion or purging of these records results in the logical deletion of those records in the temporary storage data set, with the disk space occupied by those records being reclaimed when the space is subsequently used for another record. The data elements describing the deleted or purged records are freed, and the dynamic storage occupied by those records is reclaimed for other uses.

A program using the command-level interface will retrieve elements from a queue generated by the DFHTS TYPE=PUTQ macro instruction. However, it will not retrieve data elements written by a DFHTS TYPE=PUT macro instruction.

TEMPORARY STORAGE RECOVERY

After a controlled or uncontrolled termination of CICS/VS, temporary storage records on disk may remain available for use, if desired. Temporary storage in dynamic storage is lost.

On restart of CICS/VS, either a cold start, warm start, or emergency restart may be specified. If a cold start of temporary storage is specified, any information recorded on disk is lost.

If a warm start is specified on system restart, the information in the temporary storage data set is retained. The temporary storage keypoint recorded at system termination (see Part 5) is used to reconstruct the data elements in CICS/VS dynamic storage, to enable subsequent retrieval of information by application programs once the system has been restarted. Note that the system termination keypoint will only be valid for a warm start if it was taken as a result of a normal (non-immediate) closedown.

If an emergency restart is specified, the information in the temporary storage data set is retained. The contents of that data set, and any temporary storage update activity automatically logged to the CICS/VS system log prior to uncontrolled shutdown, are used to reconstruct temporary storage tables in dynamic storage. These tables identify the status of temporary storage at uncontrolled shutdown. The data identification of temporary storage records and queues, the number of entries in queues, the location of each entry in auxiliary storage, and the status of available space in the temporary storage data set are reconstructed during emergency restart.

The processing of in-flight tasks is also backed out during emergency restart. A task is considered in-flight if it did not pass through a user synchronization point (with no subsequent logging activity) or terminate before uncontrolled shutdown.

A further aspect of temporary storage recovery, applicable to auxiliary but not main storage data IDs, is dynamic transaction backout. If a transaction should terminate abnormally after modifying a recoverable auxiliary temporary storage ID, then the modifications will be removed as part of the abnormal termination handling.

Thus, a consideration in the use of dynamic storage or auxiliary storage as a temporary storage medium is the requirement for recoverability. Information stored in main storage will be lost;

information stored in auxiliary storage may be recovered, following either a warm restart, an emergency restart, or an individual transaction failure.

CICS/VS Transient Data

The queuing facility provided by CICS/VS for online applications is supported by the transient data management routine of CICS/VS. There are two types of transient data queues. These are:

1. Extrapartition

   Extrapartition queues are sequential data sets used for transfer of information between CICS/VS and batch partitions.

2. Intrapartition

   The intrapartition data set supports queues used within the CICS/VS partition itself, to transfer information between CICS/VS tasks.

EXTRAPARTITION DATA SETS

Extrapartition data sets in CICS/VS are used for the following main purposes:

Batch Data Transfer: Information which is to be passed from CICS/VS to batch partitions is directed to extrapartition data sets or queues. These data sets are normal sequential data sets using QSAM for OS/VS or SAM for VSE.

Similarly, information to be passed from a batch partition to CICS/VS is read by the relevant CICS/VS task from an extrapartition input data set.

Sequential Devices: Extrapartition data sets may be used by CICS/VS to communicate with various sequential devices, such as line printers. Because the standard sequential access method under OS/VS or VSE is used to support extrapartition data sets, those devices supported by the standard sequential access method can be utilized by CICS/VS. These include card reader, line printer, disk, and tape. Particularly because of OS/VS device independence, most sequential devices which are supported by QSAM may be utilized as either input or output data sets by CICS/VS, when the user specifies them as extrapartition data sets.

INTRAPARTITION DATA SETS

Intrapartition queues are used to pass direct access organized data (chained sequentially) between CICS/VS tasks. A number of application-oriented uses for intrapartition files are:

Batch Queues: Data received from many terminals for the same application may be consolidated in one queue for processing as a batch. Each concurrently executing task may direct the data to the relevant batch queue, where it is chained sequentially. Subsequently, this batch or queue of data may be processed as an input file of information by a CICS/VS task.

Automatic Tasks: Data stored as a queue as described above may be automatically processed by a CICS/VS task when a specified amount of information has been queued. Based upon a trigger level (or count) for that queue, a specified task may be automatically initiated to process that quantity of data. The trigger level may vary from 0 (which implies no automatic task initiation) through 1 (which initiates a task each time information is written to the queue) to a trigger level of greater than 1.

Terminal Output: Output may be automatically directed to a terminal from several tasks. This automatic output may not be able to be sent to the terminal for some time, because it is engaged in other activity such as entering an input transaction or receiving output from previous transactions.

In addition, the terminal may be one to which output is only sent when requested. An example of such a terminal would be a video terminal. Automatic output directed to a video terminal may not be displayed at a convenient time, or may not allow sufficient time for assimilation of the information displayed. Hard-copy terminals, however, may be able to receive automatic output at any time they are not active, unless they are used with preprinted stationery. In this case, automatic output for a terminal must be queued on disk until the terminal is able to receive it, or until the terminal operator has explicitly requested it.

Output to be directed automatically to a terminal is queued on an intrapartition queue. A trigger level may be associated with this queue such that when a specified number of output messages have been queued a task is automatically initiated to transmit those messages to the terminal, if the terminal is able to receive those messages at that time.

Audit: Intrapartition (or extrapartition) queues may be used to accumulate information for audit purposes. Intrapartition queues may be specified as being nonreusable. Data written to these queues is accumulated throughout the operational period of CICS/VS, and will only be deleted (and the disk space used will only be freed) by an explicit DELETEQ TD command issued by an application program.

Alternatively, queues may be specified as reusable, in which case information on these queues is purged automatically by CICS/VS when the data has been read by application programs.

As discussed above, extrapartition data sets provide a sequential data set capability to CICS/VS. Standard access methods such as QSAM for OS/VS or SAM for VSE are utilized. The specification of the particular sequential data set is made at system generation time. Further information describing that data set may be provided at CICS/VS system initiation time from OS/VS DD, or VSE DLBL and EXTENT, job control statements. Extrapartition data sets can be either fixed-length or variable-length, blocked or unblocked data sets.

Record Accessing

Each extrapartition data set is identified by a four-character queue name (or destination identification). This name is specified by a CICS/VS task when it requests input (GET) or output (PUT) on a particular data set (see Figure 4.2-1).

This queue name is used to locate the relevant entry in a destination control table (DCT) describing that particular extrapartition data set. CICS/VS transient data management then issues the appropriate VSE or OS/VS GET or PUT macro instructions for the particular sequential access method. (See the CICS/VS Application Programmer's Reference Manual.)

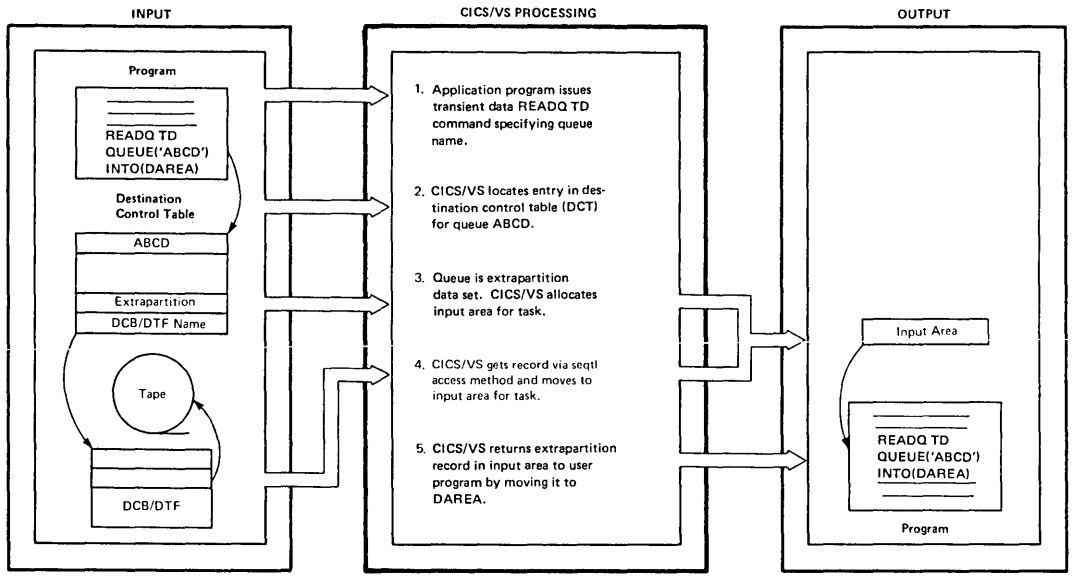


Figure 4.2-1. Extrapartition Data Set Accessing

For output to a sequential data set, the output record is constructed by the application program, after which the program issues a WRITEQ TD command indicating the relevant queue name of the output data set. The output record is then written by transient data to the specified sequential data set.

When an application program has to initiate input from a sequential data set, it issues a READQ TD command specifying the relevant queue name. Transient data determines the data set involved, and moves the next sequential input record into the specified data area. The accessing of extrapartition data sets is illustrated in Figure 4.2-1.

Extrapartition data sets may be either fixed-length or variable-length, blocked or unblocked.

Recovery of Extrapartition Data Sets

CICS/VS does not attempt to recover extrapartition data sets after a controlled shutdown or in the event of abnormal termination or system failure. See Part 5 for further information.

INTRAPARTITION TRANSIENT DATA

As discussed previously, the intrapartition data set provides a useful queuing facility for passing information between CICS/VS tasks. Its main use is to provide support for accumulation of data to be either processed as a batch or automatically transmitted to a terminal.

Record Accessing

Data is written to or read from intrapartition queues by CICS/VS application programs in exactly the same way as for extrapartition data sets. However, only variable-length records are supported, and if the data sets are on fixed block architecture (FBA) devices, only VSAM may be used to access them. An application program sets up the output record, and issues a WRITEQ TD command specifying the relevant four-character intrapartition queue name.

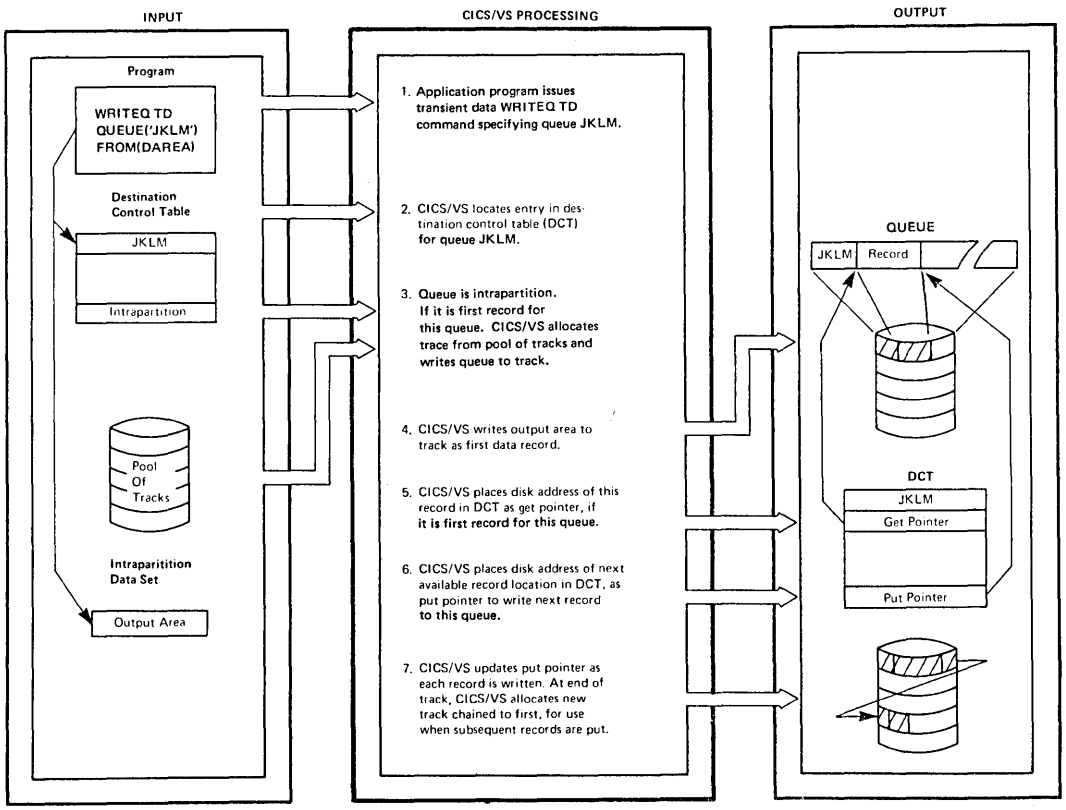
Similarly, for input, when READQ TD command is issued by an application program, the record is read and passed to the requesting task.

From a general programming point of view, there is no effective difference between reading and writing extrapartition data sets or intrapartition queues. The indication by the program as to whether an extrapartition data set or an intrapartition queue is to be used is the specification of the relevant queue name.

One main difference between extrapartition and intrapartition queues, however, is that intrapartition queues may be specified as being reusable, if required. Thus they can be used as work files if needed, queuing data to be processed, and then, after processing that data, deleting it so that the disk space it occupied can be utilized for other purposes. This is discussed in more detail in "Reusable Intrapartition Queues" later in this chapter.

Intrapartition Disk Organization

CICS/VS uses a direct access data set to support intrapartition queues. The disk space allocated for the intrapartition data set is regarded as a pool of tracks, or, for VSAM implementations, control intervals, which may be allocated to intrapartition queues (destinations) as required (see Figure 4.2-2).



Note: For VSAM implementations, the term "track" in this figure refers to a control interval.

Figure 4.2-2. Intrapartition Disk Organization

Transient data maintains a series of tracks or control intervals allocated to each active destination, based on the dynamic requirements for intrapartition disk space. However, records are logically read from a destination in the sequence in which they were written; it thus appears to the CICS/VS task as if it were operating on a normal sequential data set. The data set is actually a direct access file.

INTRAPARTITION QUEUE USAGE

As discussed above, intrapartition queues are generally used to accumulate and process data as a batch of records. The various application uses to which intrapartition queues can be applied are now discussed in more detail.

Batch Retrieval

Records may be accumulated as a batch on an intrapartition queue or destination. Retrieval and processing of these records are achieved by a task issuing READQ TD commands specifying that queue. The initiation of these tasks may be achieved in one of three ways:

Transaction Initiation: A transaction entered by a terminal can initiate a task which issues READQ TD commands for a particular intrapartition destination. Data retrieved in this way can then be processed as required.

Interval Control Initiation: A task can be initiated at a future time based upon elapsed time or time of day. This task can issue READQ TD or RETRIEVE commands to read records queued on an intrapartition destination and process them. (See "Interval Control" for information about the RETRIEVE command.)

Automatic Task Initiation: A task may be automatically initiated by transient data when a specified number of records have been written to an intrapartition queue. The trigger level specified in the DCT entry for that queue is compared with the count of records which still remain to be read. When the queue count equals the trigger level, a specified task (as identified by a transaction code in the DCT entry) is initiated. This task may issue READQ TD commands to read and process the data on that queue. This is discussed further in the section, "Terminal Output."

Intrapartition Recovery

CICS/VS supports the recovery of intrapartition transient data queues on a warm start following a controlled shutdown, and on an emergency restart following an uncontrolled shutdown. The DCT status of each intrapartition destination is reestablished to reflect the READQ pointer, WRITEQ pointer, queue count and trigger level status as it was prior to the shutdown. Intrapartition queues are recovered, and automatic task initiation can then proceed after CICS/VS restart as if shutdown had not occurred.

On emergency restart following an uncontrolled shutdown, any intrapartition queue can be recovered to reflect all activity against those queue up to the point of uncontrolled shutdown. This is called "physical recovery." Alternatively, any intrapartition queues can be recovered to reflect the activity of completed tasks prior to uncontrolled shutdown; all in-flight task activity at uncontrolled shutdown is backed out during an emergency restart. This is called "logical recovery." The user specifies in the DCT, at system generation time, whether an intrapartition queue requires physical recovery or

logical recovery. Logically recoverable TD queues can also be recovered to their start-of-task status following the abnormal termination of a single transaction that accessed those destinations. (See Part 5 for further information.)

Terminal Output

Data may be directed to a terminal from many tasks. That terminal may presently be active either entering input or receiving output from one task. When other tasks wish to transmit output messages to that same terminal, it is necessary for these messages to be queued on disk until the terminal is ready to receive them.

This message queueing is achieved by requiring the other tasks to write the terminal output messages to a transient data queue. This queue is intrapartition, and furthermore is identified as being associated with a terminal. The queue name used must be identical to the terminal identification in the terminal control table (TCT) entry for the associated terminal. Tasks may issue WRITEQ TD commands specifying as a queue name the terminal identification. These output messages will then be queued in the intrapartition data set (see Figure 4.2-3).

To initiate transmission of these output messages to the relevant terminal, a trigger level of 1 is generally specified for that terminal destination. As soon as one output message has been written to that terminal queue, a task (identified by a transaction code in that DCT entry for the relevant destination) is eligible to be automatically initiated. The program used by that transaction code will conventionally be a common program, developed by the installation, to transmit data from various queues to their relevant terminals.

However, to be able to transmit messages from the intrapartition queue to the terminal, that terminal must be idle and able to receive automatic output—that is, the output sent to the terminal is not in response to a transaction entered earlier by the terminal.

Accordingly, unless the associated terminal is idle and able to receive output, a task is not automatically initiated based upon the trigger level of a terminal queue. If these conditions exist, the task is initiated. The terminal is allocated to that task as if the terminal itself had entered the transaction code which initiated the automatic task. The automatic task may now issue commands to retrieve output messages from the particular terminal intrapartition queue. These messages may be transmitted directly to the terminal using CICS/VS terminal control or basic mapping support commands.

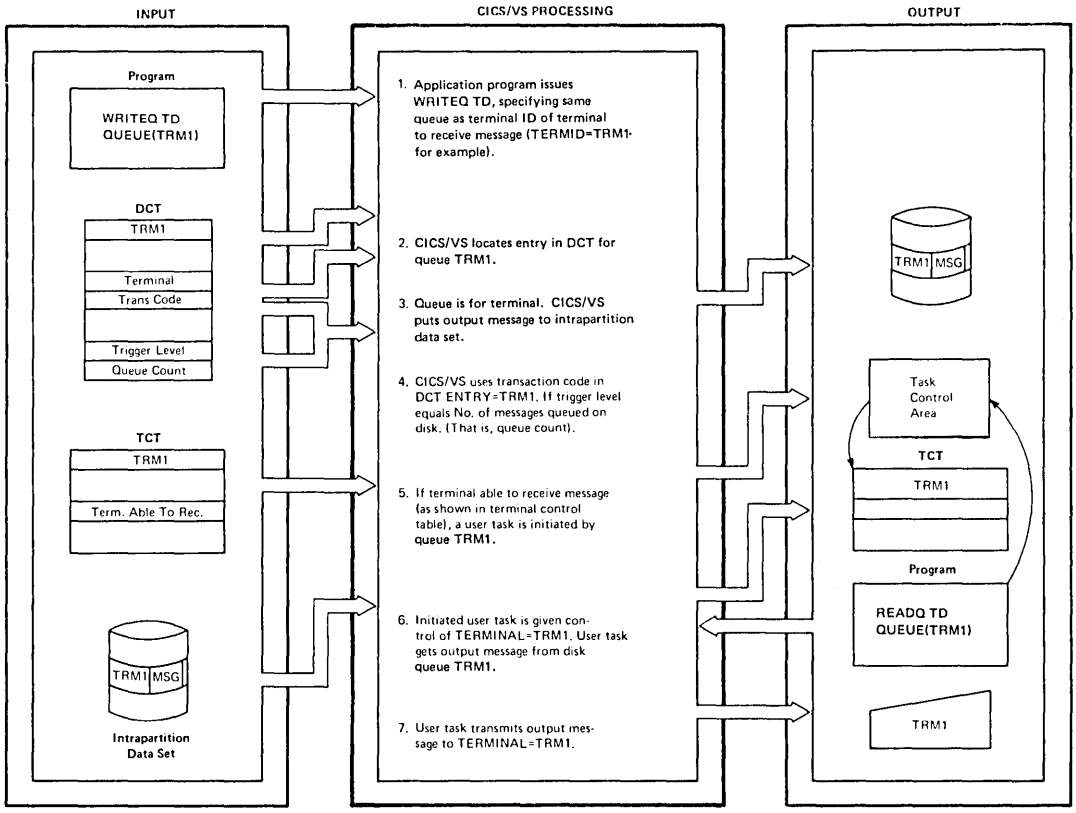


Figure 4.2-3. Terminal Output Via Intrapartition Data Set

Terminal Status

To indicate whether a terminal may receive automatic output or not, a processing status is defined for each CICS/VS terminal. The processing status codes are:

- TRANSACTION status
- TRANSCEIVE status
- RECEIVE status
- INPUT status
- PAGE status
- AUTOPAGE status

TRANSACTION processing status indicates that a terminal is unable to receive automatic output. It can receive output only as a result of an input transaction entered from that same terminal. Output queued from other tasks for a TRANSACTION status terminal can be transmitted to it

only when the terminal operator enters a transaction code which will read the data from the relevant intrapartition queue, and send it to that terminal. The terminal operator has control over when he will receive the queued output. Generally, and particularly for video terminals, one intrapartition message would be transmitted each time the relevant transaction code is entered. The terminal operator can then assimilate the information presented to him before the next output message is requested.

TRANSCEIVE status indicates that a terminal may enter input transactions, but can also receive automatic output from other tasks. This is generally used for hard-copy terminals, where several lines of output may be automatically transmitted when the terminal is idle.

RECEIVE status indicates that a terminal is unable to enter any input data, but is only able to receive automatic output from other tasks. This is generally used for printers.

INPUT status indicates that a terminal can enter data but cannot receive data.

PAGE status indicates that a terminal can only retrieve pages on request, one at a time.

AUTOPAGE status indicates that a terminal will receive all pages queued for it.

Two additional terminal status codes are used to indicate the activity status of each CICS/VS terminal. These are:

- IN-SERVICE status
- OUT-OF-SERVICE status

IN-SERVICE status indicates that the terminal is presently active and able to process as defined above.

OUT-OF-SERVICE status indicates that the terminal is presently inactive, either because it has been marked out-of-service by the master terminal operator for example, or because of an unrecoverable I/O error which occurred on that terminal. In this case, it is unable to enter any messages or receive any output, automatic or otherwise.

Terminal status is recorded in the TCT. It is set up by the system programmer and may be varied by the master terminal operator. The CEMT master terminal transaction does not use the statuses RECEIVE, TRANSACTION and TRANSCEIVE. Instead, the master terminal operator specifies whether or not ordinary terminal transaction initiation (TTI/NOTTI) and automatic transaction initiation (ATI/NOATI) are allowed. For further information about the TCT and operation of the master terminal, respectively, see the System Programmer's Reference Manual and the Operator's Guide.

Thus, for a task to be automatically initiated based upon a terminal intrapartition destination trigger level, the relevant terminal must have the following status:

- IN-SERVICE status
- TRANSCEIVE status, or RECEIVE status

- If the terminal is OUT-OF-SERVICE, messages are accumulated on the intrapartition destination queue until the terminal is placed IN-SERVICE. If the status is TRANSACTION, messages are also accumulated on the intrapartition queue until either the status is changed to RECEIVE or TRANSCEIVE, or the terminal operator enters the transaction code to initiate a task which will read the messages from transient data and send them to the terminal.

A VTAM-supported terminal (such as the 3600) which supports automatic task initiation, may be IN-SERVICE and in TRANSCEIVE or RECEIVE status as indicated in its relevant TCT entry but may not currently be connected to CICS/VS. It may be operating offline or be communicating with other VTAM application programs. If a task is to be automatically initiated for that terminal, CICS/VS will request VTAM to establish connection with the relevant logical unit. This may require VTAM to request that another VTAM application program communicating with the logical unit release it for connection to CICS/VS, or may require VTAM to establish a new logical connection (session) to the logical unit currently in offline mode. Session establishment for TCAM logical units is a function of the MCP. CICS/VS is not involved in the setup or takedown of the logical unit session.

Notification of Queued Output Messages

In the case of a TRANSACTION status terminal, some indication should be given to the terminal operator that messages are queued. This can be done either by the terminal operator periodically requesting that any messages queued be sent to him, or through the techniques shown in Figures 4.2-4 and 4.2-5.

Figure 4.2-4 shows one terminal operator notification technique. The application program that retrieves the data from Transient Data may indicate in a standard area of a display screen the number of messages to be sent. This is then presented to the program for incorporation into the output message that is sent to the terminal. Part of the response sent back to that terminal then indicates the number of messages presently queued to be transmitted to the operator upon his request.

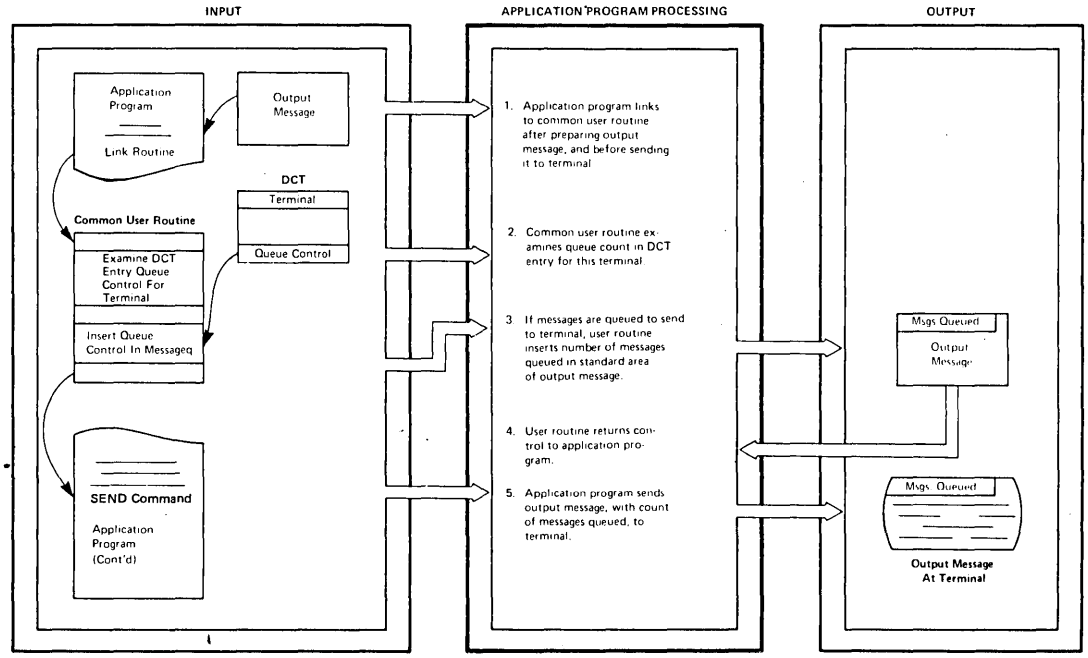


Figure 4.2-4. Notification to Terminal Operator of Automatic Output

A second technique is shown in Figure 4.2-5, and utilizes the terminal paging facility of CICS/VS to control automatic output to the terminal. The terminals must be specified as TRANSCEIVE or RECEIVE status, such that automatic output may be sent to them. Tasks preparing output to be transmitted to a specific terminal prepare that output as a series of pages to be displayed to the terminal. These pages, however, are directed to temporary storage through the use of the BMS terminal paging commands, instead of to the intrapartition destination for that terminal.

The terminal operator may then request at his convenience pages of information to be displayed in whichever sequence he requires.

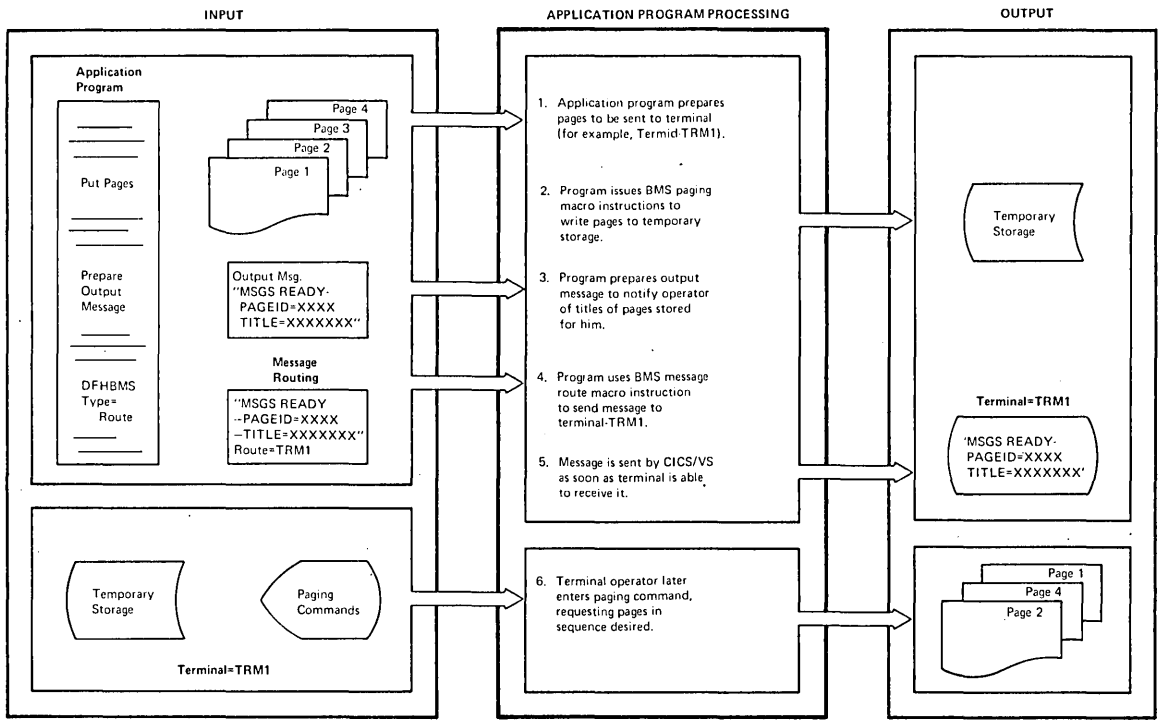


Figure 4.2-5. Notification of Paged Output

The task that generated the pages for display may also issue the ROUTE command to send a message to the terminal notifying it of the fact that pages have been stored, and identifying the pages so that they can be displayed, when convenient, by the operator entering CICS/VS terminal paging commands. Thus the amount of information the terminal operator has to read as the result of automatic output is limited to one line, and he can use the CICS/VS paging commands to request subsequent output when he desires it. Terminal output formats should be designed to reserve at least one line on display terminals for automatic system-to-operator messages of this nature.

This second technique is based on terminal paging, which utilizes temporary storage and VSAM.

If a task is to be automatically initiated to send output to a VTAM-supported terminal such as the 3600, CICS/VS establishes a logical connection, if the relevant logical unit is not currently connected to CICS/VS. The 3600 AP controlling that logical unit is then notified of the requirement by CICS/VS to automatically initiate a task on behalf of that logical unit. This is achieved by CICS/VS requesting VTAM to send a "bid" command. On receipt of the bid, the AP can notify the terminal operator (perhaps by displaying a message or by turning on an indicator light on the 3604) that automatic output is to be sent to him. If he indicates that he can receive that output, the AP can respond positively to the bid. CICS/VS then automatically initiates the task to send data to the AP, and hence the terminal operator. If, however, the terminal operator does not wish to accept automatic output at that time, the AP can respond negatively to the bid. CICS/VS will not reissue the bid at a later time. When the terminal operator is able to accept the

automatic output, he notifies the AP. The AP then transmits a "ready to receive" command to VTAM, and hence CICS/VS. CICS/VS then automatically initiates the task as discussed above. Refer to the appropriate CICS/VS sub-system guides for further information. For TCAM, these functions of "bid" and "ready to receive" are handled in the Message Handler. For further information refer to the OS/VS TCAM System Programmer's Guide.

If none of the above techniques is to be utilized, the terminal operator can periodically enter a user transaction which reads any messages queued for that terminal destination in transient data, and transmits those messages to the terminal. This does not require the use of the techniques previously described, but has the disadvantage that it is completely dependent upon the terminal operator.

Low or High Priority Processing

CICS/VS intrapartition queues may be utilized for low (or high) priority processing. A program can receive transactions from a terminal, validate them, and notify the terminal of any error messages. Valid transactions are directed to an intrapartition destination, and queued for that destination until a specified trigger level is reached.

A terminal is not associated with this destination. When the trigger level is reached, a task is automatically initiated based upon the transaction code specified for that destination. As no terminal or operator is associated with this task, the task priority used in processing these transactions is the transaction priority as specified for that transaction code in the program control table (PCT). The initiated task may read the transactions queued to that intrapartition destination, process them, and update any required data sets depending upon the application requirements. Processing of data may then proceed independently of subsequent terminal input.

This technique is utilized by the asynchronous transaction processing (ATP) facility in CICS/VS (see Part 3). A batch of transactions may be entered from a batch terminal using the ATP transaction, CRDR. This batch is given a batch name by the terminal operator, and each transaction is queued on a transient data intrapartition queue until all batch input is completed. At this time, a task (or tasks) is initiated, based upon the transactions in the batches, to process those batches. In the meantime, the terminal operator is free to enter any other transactions, including other ATP batches.

During processing of the ATP batches, terminal output is directed by application programs to intrapartition destinations. This terminal output may be retrieved and transmitted to the terminal, when requested by the terminal operator. This is achieved by entering the ATP transaction code CWTR.

REUSABLE INTRAPARTITION QUEUES

Intrapartition queues can be specified as nonreusable or reusable. Nonreusable queues accumulate data over the entire CICS/VS operational period, including any warm starts following termination of CICS/VS (see Part 5). Data on nonreusable queues is not destroyed until transient data is cold started, or until explicitly purged by user programs.

If reusable queues are employed, when an application program issuing a READQ TD command causes data to be read from a new track, the track

just read is automatically returned by transient data to the pool of tracks available for use in satisfying other WRITEQ TD requests. This also causes transient data to reformat the returned track for later use, and may in some cases result in performance degradation during this reformatting. Note that no reformatting is required for returned VSAM control intervals, and that no performance degradation is incurred.

The intrapartition data set can therefore be utilized most efficiently for those destinations for which data does not need to be retained; however, other destinations containing data which must be retained for audit or recovery purposes, are not disturbed.

INDIRECT DESTINATIONS

CICS/VS transient data uses extrapartition, intrapartition, remote, and indirect destinations.

An indirect destination has its own destination identification, but in turn identifies another destination. Output eventually to be directed to specific devices may be written to a "logical" intrapartition destination. This logical destination identification is an indirect destination, which in turn specifies the destination for the physical device to be used to receive that output (see Figure 4.2-6).

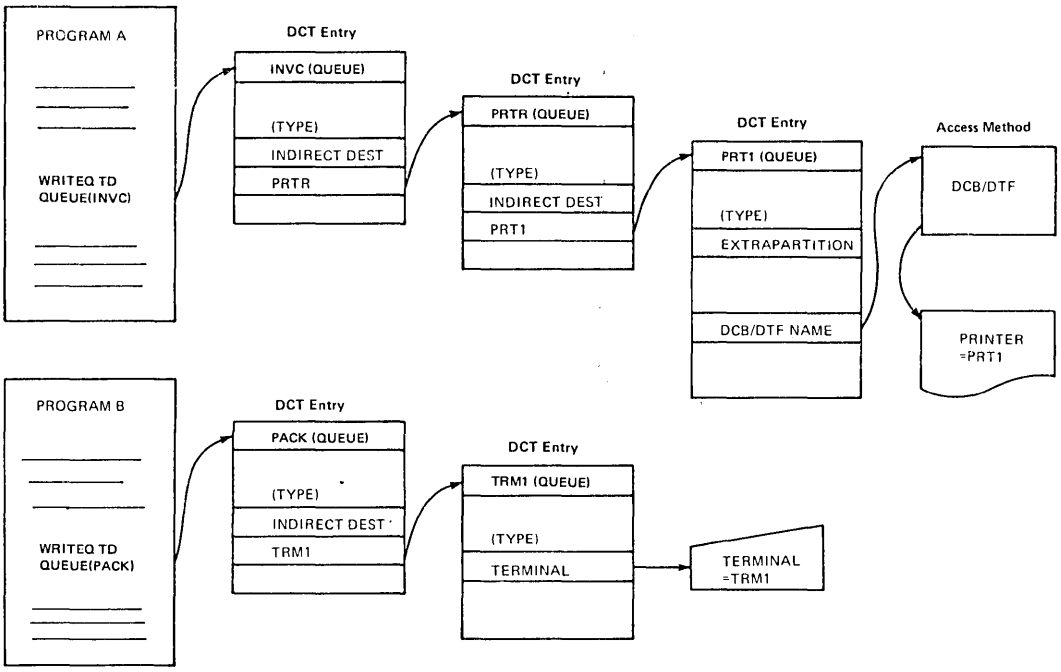


Figure 4.2-6. Indirect Destinations

If the output is to be subsequently directed to some other device, the application programs do not have to be changed. The output is directed to the relevant logical destination. However, the entry for that indirect logical destination is changed in the DCT to refer to the new device, which may be either intrapartition, such as a terminal, or extrapartition, such as a tape, disk, or printer.

Thus the amount of maintenance resulting from a change in the terminal network configuration, for example, is reduced to only a change and reassembly of the DCT.

Different types of output to be directed to the same terminal should be written to different logical indirect destinations. These different destinations may refer indirectly to the same terminal destination. If, at some later time it is decided to separate logical output across terminals, instead of having it appear on the same terminal, this can be achieved merely by changing the relevant indirect logical destinations to point to the new terminals to receive that output. No change need be made to the application programs.

As well as reducing the amount of program maintenance resulting from a change in the terminal network configuration or a change in application requirements directing output to different terminals, indirect destinations have other useful purposes. These are summarized below.

Device Independence

By directing output to logical indirect destinations instead of to specific terminal destinations, the programs now become independent of the particular device selected to receive that output. An indirect destination may point to any intrapartition or extrapartition destination. For example, the output which may normally be directed to a terminal printer may be directed to an extrapartition destination line printer. This can be achieved by writing the output to an indirect transient data destination, and then reassembling the DCT to point to the line printer extrapartition destination identification.

Terminal Backup

The use of indirect destinations and device independence raises the question of terminal backup. Through the use of indirect destinations, programs are no longer dependent upon the availability of specific terminals. In the event of a terminal going down, an alternative terminal or device (tape, disk, or printer) may be assigned to receive the output logically directed to the failing terminal.

On terminal failure, it is not practical to reassemble the destination control table to change the indirect destination to the backup device, without terminating CICS/VS. In this case, the system design team should evaluate the requirement for a backup capability to enable critical information to be received.

If it is necessary that information be directed to an alternative device, then the destination control table may be changed dynamically by user-written programs. The user may write an application program (initiated by a specific transaction code) to search the destination control table for the specified indirect destination. The destination identification pointing indirectly to the failed device can then be modified to point indirectly to the destination of an alternative device. Data already queued for the original destination cannot be sent to the failed terminal, but must then be copied by the user program to the destination queue of the allocated device. Subsequent data written to the indirect destination will then automatically be directed to that device (see Figure 4.2-7).

Note: In the event of abnormal termination because of a power failure or machine check, and subsequent reinitiation of CICS/VS, such user modifications to the DCT may be lost. The DCT will be initialized by CICS/VS as if the user modification had not occurred, since it is not aware that the DCT was changed by the user. This can be overcome by the user program journaling each DCT modification and reestablishing each modification itself after reinitiation.

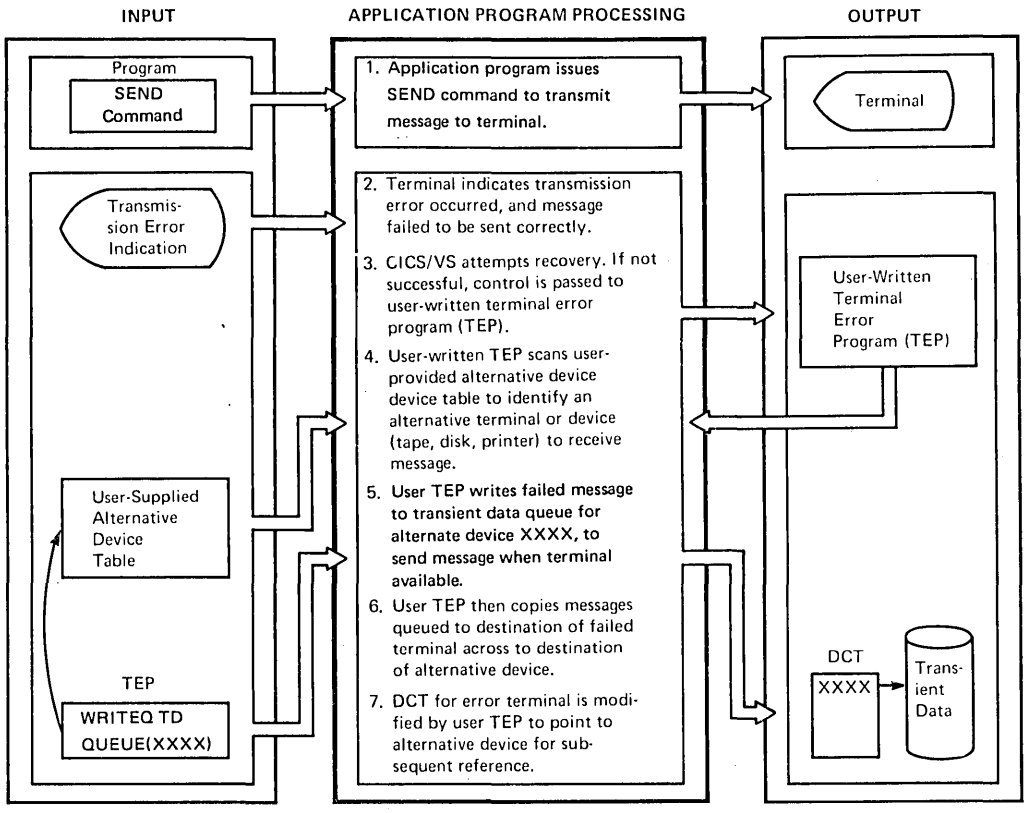


Figure 4.2-7. Terminal Backup and Reconfiguration

The transaction code allocated to the DCT modification program may be given a security code so that only certain authorized terminal operators, such as the master terminal operator, may use it.

Dynamic Terminal Reconfiguration

The user-written DCT Modification program for terminal backup described above may also be utilized for dynamic terminal reconfiguration. If at different times of the day it is required to change the destination of logical output to different physical devices, this can be achieved by using the DCT modification transaction code and program.

This raises the possibility of dynamically reconfiguring the terminal network, or other devices, to receive output. For example, at one time of the day output may be directed to a particular terminal printer, while at other times it may be directed to a display screen, and again, to a line printer.

As described above, any dynamic DCT modification made by user-written programs should be journaled by the user, and utilized after CICS/VS reinitialization to reestablish the modified DCT.

The availability of CICS/VS terminal device independence, which enables application programs to present output messages in a standard form regardless of the terminal type which will receive those messages, lends itself to such dynamic reconfiguration capability. Dynamic terminal reconfiguration is discussed further in Part 5.

Because CICS/VS allows any terminal (or simulated terminal such as card reader, disk, or tape) to enter any transaction, the user-developed support of dynamic reconfiguration also enables the master terminal operator to exercise control over where output is to be directed based upon online application requirements. Used in this way, transient data and indirect destinations become powerful online application tools.

Some TCAM- or VTAM-supported terminals (such as the 3600) and the 3600 terminals using BTAM permit dynamic terminal reconfiguration to be performed by the controller for the devices controlled by an AP. Through use of logical device addresses, the AP identifies devices to be used for I/O. The controller relates their logical device addresses to physical device addresses using a table associated with that AP. The controller also permits this table to be changed dynamically so that a specific logical device address may refer to a different physical device. The 3600 system operator may request this reassignment to be carried out by a user-developed AP. For example, a 3600 system operator may reassign an alternative printer for use by an AP with an inoperative printer.

This device reassignment is transparent to CICS/VS. CICS/VS using VTAM communicates output disposition to an AP through use of logical device codes (LDCs). The AP then relates the logical device code to a logical device address and issues the relevant output request. CICS/VS using BTAM communicates with the AP which relates the data to a logical device address and issues the appropriate I/O request. In both cases the controller then relates the logical device address to a physical device as previously described.

The AP can interpret the LDC based upon application requirements and identify a logical device address to the controller. The controller can then identify the physical device currently assigned to that logical device address for that AP.

Use of alternative devices and device reassignment support in the 3601 provides additional system flexibility and availability for 3600 users.

Other Methods of Data Transfer Between Modules

Data may be transferred between application modules by using either the COMMAREA option of the Command Level Interface, the Transaction Work Area (TWA), as well as Temporary Storage.

COMMAREA Option

The COMMAREA option of certain commands (LINK, XCTL and RETURN) may be used to pass data from one application module to another, provided that both modules have been written using the Command Level Interface. The length of the communication area is defined within the application module, whereas with the TWA the length is defined in the Program Control Table (PCT). If a change is required to the length of a communication area defined by the COMMAREA option, then the application modules need to be changed and re-linked, whereas with the TWA the systems programmer must regenerate the PCT.

TWA For Data Transfer

The TWA can be used only if the information will be subsequently used by the same application program, or by another application program which executes under control of the same TCA. That is, control must be passed to the subsequent program by either XCTL or by LINK commands. If the information is to be passed to some future task initiated by time, or by a subsequent transaction entered by a terminal operator, then the TWA cannot be used. This is because the TCA and associated TWA are destroyed when the task which generated the information terminates execution. Consequently, the TWA may be used for data transfer of a short-term nature, while temporary storage is generally used for data transfer of long-term nature.

TWA Size

A consideration in the use of a TWA or temporary storage is the amount of data to be stored. The size of the TWA associated with a transaction code is stored in the program control table (PCT). This TWA size is used to allocate a TWA appended to the TCA. Thus, if a TWA of 200 bytes is indicated in the PCT, the TCA is allocated 200 bytes more than if no TWA size is specified.

TWA For Short-term Data Transfer

A further factor is the duration of execution of the task, and the amount of time between when data may be stored in the TWA and when it will be subsequently retrieved from the TWA. As a general rule, if data may remain in the TWA for longer than one second it should be stored in temporary storage instead. This would be particularly advisable if a TWA much larger than 200 to 300 bytes was to be used. Furthermore, because of the relatively low activity of use of this data (because of the long execution time), it should be stored on disk rather than in dynamic storage address space.

Variable TWA Size Requirements

Another factor is the possible requirement of the program for different size TWAs based upon the processing required. For example, 90% of transactions which use the same transaction code and application program may require a TWA of 50 bytes. However, the remaining 10% of these transactions may require a TWA of 500 bytes, say. If a TWA were used for all transactions by this program, a 500-byte TWA would have to be specified in the relevant PCT entry. This would mean that for 90% of transactions using that program, 450 bytes of storage would be wasted.

A more efficient solution in this case would be to allocate a 50-byte TWA, and utilize this TWA for the 90% of transactions which need 50 bytes. In the case of the remaining 10% of transactions, temporary storage on disk should be utilized. Thus, storage is used most efficiently, with the additional time to store information on disk and retrieve it from disk only affecting 10% of the transactions in this example.

Chapter 4.3. Program Development and Testing

Modular Programming

BATCH ENVIRONMENT

Modular programming techniques in a batch environment may involve the consolidation of similar program functions in one program module. For example, the main execution code used may be incorporated in one module, while exception routines may be in another module and error routines in other modules. In this way, modular programming enables sections of the program to be written by programmers at different times. Apart from the advantage of distributing the program workload across several people, another advantage of modular programming is that it generally makes the application program logically easier to follow for someone who is unfamiliar with it.

CICS/VS ONLINE ENVIRONMENT

CICS/VS is oriented around the concept of modular programming. Transactions received from terminals are analogous to transaction cards read from a card reader. A transaction code defines the format and processing required for an online transaction, in the same manner as a card code defines the format and processing of a card.

This transaction code identifies the CICS/VS application program that will process the transaction. The use of such modular programming techniques is an integral part of CICS/VS and enables large programs to be broken into smaller logical modules. However, program size and CICS/VS address space availability should be balanced with the additional overheads involved in passing control between many small modules.

When a transaction code is received from a terminal, only that program code relevant to the processing of that transaction need be loaded into storage, if it is not already present. As modules tend to be smaller than complete programs, more application program modules may reside in a given address space than may full programs. This enables one copy of each of many different modules to be currently resident in the CICS/VS dynamic storage area. A high degree of multitasking may therefore be achieved within a limited storage size.

VIRTUAL STORAGE ENVIRONMENT

Using the modular programming techniques discussed above, a CICS/VS application program module should include code which is relevant to the processing of the specific transaction.

From the system design point of view, the design team should specify the various application programs which are to be written to implement

the particular application. They should also identify those application functions (and hence program coding) which will be frequently used by transactions, and those which will be infrequently used. In this way, the design team is able to broadly specify the modular program structure of the application, and define the necessary application programs.

The various application programs executing concurrently in the CICS/VS partition, and the demands made by them for CICS/VS services and resources, contribute to the total "working set" of CICS/VS. The CICS/VS working set is influenced by the sizes of the various concurrently executing application programs, the online transaction load and its use of various application programs, and the degree of multitasking permitted by the CICS/VS master terminal operator.

High Level Languages

CICS/VS accepts application programs written in Assembler, COBOL, or PL/I (compiled by the PL/I Optimizing Compiler) for VSE, OS/VS1, or OS/VS2. CICS/VS also accepts programs written in RPG II for VSE only.

When an application program is coded in either COBOL, Assembler, RPG II, or PL/I it is necessary for there to be an interface between the application program and CICS/VS, and corresponding compile-time mechanisms for invoking this interface. CICS/VS provides two methods of achieving this interface, one using commands and arguments (command-level interface) and one using assembler macros and CICS/VS control blocks (referred to as the macro-level interface). RPG II is supported only through the command level interface.

Using the macro-level interface, the application programmer may use assembler macros (DFHxx) in programs coded in either COBOL or PL/I, full details of which are given in the CICS/VS Application Programmer's Reference Manual (Macro Level). At compile-time the program is prepared by a CICS/VS preprocessor. The preprocessor is generated at CICS/VS system generation by use of the DFHSG PROGRAM=HLL macro (for full details refer to the CICS/VS System Programmer's Reference Manual).

The use of the macro-level preprocessor for a particular program is specified at program compile-time. Full details of the cataloged procedures to preprocess, compile, and link-edit COBOL, ASSEMBLER, or PL/I application programs are given in the CICS/VS System Programmer's Guide.

This macro-level interface presents several disadvantages to the application programmer; for example, the need to have a detailed knowledge of CICS/VS architecture, and the need to be concerned with the addressability of CICS/VS control blocks, their formats and contents. The command-level interface overcomes these problems by providing a set of EXEC CICS commands that give the application programmer access to the CICS/VS functions. They are coded in COBOL, ASSEMBLER, PL/I, or RPG II application program and are translated by means of a command language translator into standard CALL statements in the programming language being used.

A command level interface is also provided to DL/I, though the EXEC DLI command. It is supported by CICS/DOS/VS for COBOL and PL/I only. Under CICS/OS/VS, and for assembler, the alternative CALL interface must be used. For RPG II, the RQDLI command is used to provide the interface.

Throughout this manual, the application programmer is assumed to be using the command level interface.

For details of the format and use of the EXEC commands can be found in the CICS/VS Application Programmer's Reference Manual (Command Level).

The function of the command language translator is to accept as input a source program, written in either COBOL, ASSEMBLER, PL/I, or RPG II in which command requests (for both CICS/VS and DLI) have been coded. The command language translator produces as output an equivalent source program in which the EXEC CICS and EXEC DLI commands (and RQDLI requests, for RPG II) have been translated into subroutine call statements in the language in which the application program is written.

At execution time, the call statements invoke an EXEC interface program, passing appropriate arguments. The function of the EXEC interface program, called from a COBOL, ASSEMBLER, PL/I, or RPG II program, is to analyze the arguments and determine the requested function. Using the arguments passed by the application program, the EXEC interface program assigns values into appropriate CICS/VS control blocks and transfers control to the appropriate CICS/VS or DL/I control program. On return from CICS/VS, the interface program examines the return code to determine if an exceptional condition has arisen. If no exceptional condition is detected, control is returned to the application program normally. If an exceptional condition had been detected, then, if the application program is prepared to handle the condition, control is returned to the program at the specified program label. If the application program is not prepared to handle the exceptional condition, a standard system action for that condition is performed. Usually the system action is to terminate the transaction, notifying the terminal operator that the transaction had abended.

In some cases a single EXEC CICS command requires more than one CICS/VS request to be issued. Typically, a move mode output operation will require three CICS/VS requests to be issued. In such a case, the EXEC interface program issues a request to obtain storage, copies the data into the storage, issues a request to perform the output operation, and issues a request to free the storage.

There are four separate command language translators, one for COBOL, one for Assembler, one for RPG II and one for PL/I, and each one is produced in two versions, one for VSE and one for OS/VS, except for RPG II which is not supported under OS/VS.

The command language translator is executed in a separate job step; details of the cataloged procedures required are given in the CICS/VS System Programmer's Guide.

If the command-level interface is to be used, then at system generation the system programmer must:

1. Generate a command language translator for the language, (COBOL, ASSEMBLER, PL/I, or RPG II) required. This is done via the DFHSG PROGRAM=EXP macro.

2. Generate an EXEC interface program which supports the functions accessed via the command-level interface.

Alternatively, the pregenerated versions of these modules may be used.

Full details of the appropriate macro instructions are given in the CICS/VS System Programmer's Reference Manual.

Online Testing

The testing of individual CICS/VS commands and CICS/VS application programs, and the subsequent integration and system test of the entire online applications, must be considered by the design team.

CICS/VS provides two facilities for online testing.

- The Execution Diagnostic Facility (EDF) to test command level application programs.
- The command interpreter to test individual CICS/VS commands.

Both these facilities are run from a 3270 display terminal with a screen width of 80 columns and a screen depth of 24 lines or more.

A short description of each of these facilities follows. Further information may be found in the Application Programmer's Reference Manual (Command Level).

EXECUTION (COMMAND LEVEL) DIAGNOSTIC FACILITY

The execution diagnostic facility (EDF) enables an application programmer to test a command-level application program online without making any modifications to the source program or the program preparation procedure. EDF intercepts execution of the application program at certain points and displays relevant information about the program at these points. It is an aid to debugging not only EXEC CICS commands, but EXEC DLI commands as well.

EDF debug mode is switched on and off by a transaction or PF (program function) key named in the PCT by the system programmer; also, the PPT needs to specify the programs and maps that are used by EDF. EDF uses temporary storage and simple BMS (as provided by ELS); the system programmer must make sure that these facilities are available.

The transaction that is being tested may run from any type of terminal. The execution diagnostic facility may run from the same terminal as the transaction under test provided that the following conditions are met:

- The terminal is connected through VTAM or BTAM. (Connection via TCAM is excluded because there is no read buffer support.)
- If auxiliary temporary storage is being used, then the system programmer must ensure that the control interval specified for the temporary storage data set is large enough to accommodate the data stream obtained by a read buffer command from all user displays created by the programs to be tested (or 1.5K bytes, whichever is the greater).

If EDF is used in two-terminal mode (that is, the direct terminal for the transaction under test is different from the EDF display terminal) then the following conditions must be met:

- Automatic transaction initiation must be supported (AUTOTRN=YES must be specified on the DFHSG PROGRAM=TCP macro).
- If auxiliary temporary storage is being used, then the system programmer must ensure that the control interval specified for the temporary storage data set is at least 1.5K bytes.

- The EDF display terminal must be in TRANSCEIVE status.

Functions of EDF

During execution of a transaction in EDF mode, EDF intercepts the execution of the application program:

- At transaction initialization.
- At the start and end of the execution of every EXEC command.
- At program termination.
- At normal and abnormal task termination.
- When an ABEND occurs

At these points of interception, EDF displays the current command or status. The user may also display other information, such as:

- The command being executed.
- The values of EIB fields.
- The contents of working storage.
- The last five commands executed.
- The contents of address locations within the CICS/VS partition.

Interaction with the application is also allowed at these interception points, and the user may:

- Modify argument values in commands before execution.
- Modify responses after command execution.
- Modify fields in the EIB.
- Make command displays conditional on specific events.

COMMAND-LEVEL INTERPRETER

The command-level interpreter enables CICS/VS commands to be entered, syntax-checked, and executed interactively at a 3270 screen. The interpreter performs a dual role in the operation of a CICS/VS system.

- For the application programmer, it provides a reference to the syntax of the whole of the CICS/VS command-level application programming interface (excluding DL/I). Most of the commands can be carried through to execution, and the results of execution can be displayed.
- For the system programmer, it provides a means of interaction with the system. For example, a corrupted data-base record can be "repaired", a temporary storage queue can be created or deleted, and so on. It thus provides a useful extension to the facilities provided by the master terminal transaction CEMT.

The command-level interpreter is a CICS/VS application program and runs as a CICS/VS transaction. It is started by the transaction identification of "CECI", or "CECS", followed optionally by the command.

The general format is:

```
CECI|CECS [command]
```

where "command" can be any CICS/VS command except EXEC DLI.

The use of CECI will give the full facilities of the interpreter right through to execution of the command.

The use of CECS forces a question mark before the command. This always gives the command syntax check display and prevents command execution. In a system where security is important, CECS can be make more widely available than CECI.

A full description of the command-level interpreter is given in the Application Programmer's Reference Manual (Command Level).

SECURITY

Both EDF and the command-level interpreter may be controlled at resource or transaction level.

Resource level security, specified by an option in the PCT, means that the user's security key must match the security key of the resource to be used. For more information on resource level security see Chapter 4.4.

At transaction level, the user's security key must match the security key in the PCT for the transaction to be executed.

Resource level security is the default.

Tracing and Testing

TRACING AND DUMPING

CICS/VS permits the execution of application programs to be traced. Information relating to the use of each CICS/VS management routine by the application program is recorded by CICS/VS in a wraparound trace table in the CICS/VS nucleus. In addition, application programs may utilize trace control commands to insert their own trace entries in the table. Every time the end of the table is reached, the trace entries re-commence at the start of the table. Trace activity may be turned on or off during CICS/VS execution by the master terminal operator (see the CICS/VS Operator's Guide.)

CICS/VS uses a feature called the Auxiliary Trace Facility which permits trace entries to be time stamped and written to tape or disk; a program supplied by CICS/VS, the Trace Utility Program (TUP), lists this tape offline in a formatted form for debugging purposes. Programs may also be developed by the user to analyze the time-stamped trace entries for performance evaluation. This analysis may consolidate all of the trace entries relating to one task in a combined task report, or may

determine the elapsed processing times of all tasks initiated from a specific terminal or by a specific transaction code. These elapsed processing times may be statistically analyzed to determine the average, standard deviation, and 95 percentile values. Similarly, a statistical distribution of the peak number of tasks in the system dynamic storage usage over a defined period of time can be developed by user analysis programs.

Analysis of the trace tape is invaluable for debugging purposes and for evaluation of the performance of existing CICS/VS programs. Potential performance bottlenecks can be identified and conditions rectified.

In addition to utilizing trace entries, application programs may issue dump control commands to dump selected parts of application programs, all the storage associated with a task, or the entire CICS/VS nucleus to disk. These dumps on disk can be printed at the end of testing, or concurrently with testing if the dump data set is switched to the alternative dump data set. Refer to Part 5 for more information relating to the use of the dump data set in this environment. During testing an online test/debug program may be used, see "Related Publications" in the Preface for details.

SIMULATED SEQUENTIAL TERMINALS

CICS/VS enables terminal transactions to be entered from a "simulated" sequential terminal such as a card reader, magnetic tape, or disk. The terminal output may be directed to a line printer, tape, or disk. Through the use of simulated sequential terminals, large volumes of test data may be prepared for what is effectively batch-type testing. No online terminals need be used or even be present on the system. Instead, a card reader and line printer, and/or magnetic tape, and/or magnetic disk may be used for input of simulated terminal transactions, and receipt of the output responses from the application programs.

A simulated terminal transaction may be delimited by a user-defined end-of-data code in a card, or by a code in a tape or disk record. Several card, tape, or disk records may be read by CICS/VS until this end-of-data code is recognized or the input area is full. Large terminal input messages may be simulated by CICS/VS for testing in a batch-type environment.

The use of CICS/VS terminal device-independence enables simulated terminal messages from card reader, tape, or disk 'terminals' to be presented to application programs as if they had been read from online BMS-supported terminals.

The output responses from the application program are directed to the simulated sequential output terminal, which may be a line printer, another tape, or another disk data set. Again, these output messages are prepared by the application program as if they are to be transmitted to a particular BMS-supported terminal, and are then converted by CICS/VS device-independent routines for output to the appropriate simulated terminal. CICS/DOS/VS Entry Level System does not support BMS requests for simulated terminals.

Because testing is carried out in a batch-type environment using sequential terminals, all test data provided must anticipate the requirements of the application program for input. This is necessary, because there is no conversational capability available in testing with sequential terminals. Consequently, conversational error correction

carried out by the application program must be simulated by successive sequential terminal transactions.

A 3270 simulator program may be used to test 3270 device-dependent characteristics using sequential terminals. See "Related Publications" in the Preface for details.

SINGLE-THREAD TESTING

The first stage of testing involves the separate execution of each application program. Only one task is active at a time, and each program can be tested without considering its interaction with other concurrently executing programs. This is achieved by using only one simulated sequential terminal. As each transaction is read from that single terminal, the appropriate program is initiated and the transaction is processed as a separate task. When that transaction completes execution, the task terminates, and the next terminal transaction is read and initiated. The test plan should include abending restartable transactions in likely places to test recovery procedures.

Single-thread testing allows the logic of application programs to be debugged without complications caused by the concurrent execution of other programs.

The next stage after all application programs have completed single-thread testing is multithread testing.

MULTITHREAD TESTING

Multithread testing is equivalent to single-thread testing, except that several sequential terminals are utilized for entry of simulated terminal transactions. Each sequential terminal will result in the initiation of one task at a time to process each transaction read from that terminal. Not until a transaction has been completely processed will the next transaction be read from that terminal and another task initiated. However, at the same time, tasks may be initiated to process transactions from the other sequential terminals. These programs from each sequential terminal execute in a limited multitasking environment, the number of concurrently executing tasks being equal to the number of simulated sequential terminals.

A series of simulated sequential terminals may be set up, either using several card readers and printers, or, more commonly, using many tape and disk simulated terminals.

An approach which may be utilized in this multithread testing environment is to take the test stream which was used for single-thread testing, and copy that test stream to each separate sequential terminal on disk or tape, at the same time randomizing the sequence of transactions in the separate copied test streams. A number of sequential terminals may then be made active to allow multitasking execution to be tested. The output from each terminal should be identical to the output for each transaction as a result of single-thread testing.

A network activity simulator program may be used to assist in multithread testing with sequential terminals. Refer to "Related Publications" in the Preface for relevant publications.

MULTIREGION OPERATION

Multiregion operation facilities (MRO) allow two or more CICS/VS systems to run in one processor system, and share data and terminals between them. One CICS/VS system may be used for testing new programs, while production work continues in parallel in one or more other systems.

The test system can be started and stopped independently of any production system, and production programs can be isolated from unreliable test programs. Even if the test CICS/VS system ends abnormally, the production systems carries on. Yet the test programs can, if required, be given access to production data resources (files, data bases transient data and temporary storage) and can be run from terminals connected to the production system. Regular terminal operators can test programs from their regular terminals using real data.

MRO is described more fully in Part 7 of this manual.

Chapter 4.4. Security Design

This chapter explains how to use the facilities of CICS/VS to provide online application security. Since many online application security concerns go beyond the confines of CICS/VS - the online subsystem - security will be discussed first without reference to CICS/VS or CICS/VS facilities. The computer installation will be viewed simply as a unit containing some of the company's operational data which needs protection from accidental, if not deliberate, mis-use.

The first section contains this general discussion and covers topics such as: degree of security required; techniques for security, and counteracting threats from different sources. Subsequent sections are devoted to discussion of CICS/VS facilities related to security or useful in attaining an application's security objectives.

Application designers who read the first section and are then concerned that the security requirements of the application being designed may exceed the ability of the installation to meet them, should look elsewhere than this manual for information. The IBM Data Security Design Handbook, a series of six publications on different aspects of security, should meet most requirements.

Security Considerations

SECURITY ROLE IN ONLINE APPLICATIONS

The main aim of an on-line application is to make timely, complete and accurate information available to the people who need it. Online applications frequently replace manual procedures or batch applications and, whereas this replacement generally improves the timeliness and completeness of the information available, some factors cause a risk that the information will be both less accurate and available to people other than those who need it. Briefly, some of these factors are:

1. Unauthorised people may be able to use the application's facilities without detection because insufficient controls are imposed on use of the facilities, or because terminals or the system itself are in an insufficiently secure area.

2. The checks and balances in a manual or batch application, which mean that collusion is required between two or more employees in order to defraud the company, may be lost in the interests of timeliness. Fraud or expensive mistakes may occur simply because people do things they are supposed to be able to do, for invalid reasons.

3. Compared to a manual procedure, an online application brings a new category of employee, programmers, into contact with the data. Programmers may, by modifying programs and waiting for them to be run by someone else, be able to defraud the company.

4. Disruption of the online service by other programs in the system, or simply the ability for someone outside the company to 'pull the mains plug out' may provide scope for fraud.

5. Other programs running at the same time in the system may be able to access, destroy or modify online data because the operating system has insufficient integrity.

6. Exposed TP connections may be wiretapped.

Thus it is the objective of the security features of an application to:

- be relatively transparent to legitimate users of the system
- be as difficult to break as is thought necessary given the value of the online data
- counteract all the relevant exposures.

SOURCES OF THREAT

Threats to the data of an online system can be from five sources:

1. From people directly involved in the operation of the computer system, the operating system and subsystems running on it. They set up the security of the system and can therefore more easily break it.

2. From people who access the data managed by the online subsystem by operating a terminal connected to the subsystem. This is discussed at length in this chapter.

3. From people who write applications for the online subsystem. This is also discussed in this chapter.

4. From interconnected systems, since the purpose of the interconnection is to exchange data. This is discussed in this chapter.

5. From others. Examples are wire tapping, for which cryptography is an appropriate security technique, and exploitation of system integrity exposures from other jobs in the system. This is not discussed further here.

DEGREE OF SECURITY

One of the first questions to be answered is how difficult illegitimate activity should be made. Since degree of difficulty will normally be greater the more money is spent on security facilities, this is a question of striking a balance between expenditure and potential loss. Too great an expenditure on security is a waste of money, too little leaves an unacceptable risk of loss due to fraud or unintentional mistakes.

Data value may be calculated several ways:

1. How valuable is it to a competitor to know. This relates to how difficult it should be for unauthorised people to see the data.

2. How important to the company is it that it is accurate. This relates to controls and checks on updates.

3. How important is it that the data is available. This relates to controls preventing disruption of the service.

In many cases, detailed study of these questions is unnecessary. The online data is not highly valuable to competitors, and errors or temporary unavailability of the data causes minor disruption only. Even in these cases, however, security controls that are cheap to use and maintain should be used because impacts due to accidents and mistakes are less likely.

SECURITY TECHNIQUES

Having decided what degree of security is desired for an application, the method of achieving this must be considered. There are three techniques available for implementing security: control, audit, and surveillance. Generally a combination of the first two will be used in any online application.

Control

Control is concerned with the (automatic) prevention of invalid actions. For example, all but the Data Processing Department people should be prevented from entering the computer room by fitting a lock to the door, and non-payroll department people should be prevented from running the salary update transaction. In the case where systems are interconnected, control may also be concerned with prevention of invalid requests by one system to another.

Control mechanisms generally produce a log of "violations", that is, attempts to perform functions which are rejected due to the action of the control mechanism.

Audit

An example of an invalid action which cannot automatically be controlled is use of the salary update transaction by a payroll department employee to double his own salary. This type of fraud is detected post facto by an auditor using audit tools.

In manual procedures, the need for collusion between two or more employees (preferably with different jobs and skill levels, and so unlikely to mix socially) to perpetrate a fraud was an excellent safeguard against fraud and human error. Therefore, in designing an online application to replace manual procedures, when it is decided that procedures involving several different people are to be eliminated, the increased risk due to fraud or error should be estimated and covered if necessary by provision of auditing facilities aimed at detecting the problems.

It must be emphasised that audit features are primarily an application's responsibility to provide; the operating system or DB/DC subsystem can only provide basic building blocks from which an auditable application can be built. Auditability should be considered early in application design.

Surveillance

Surveillance is active monitoring and in relation to online computer applications, consists of the ability to display selectively the actions of system facilities, particularly the security and audit mechanisms, in real time. Surveillance is a specialized requirement and will not be described further here.

AUTHENTICATION

Both security control/prevention and audit rely for their full effect on the ability of the computer system to know who (or in the case of inter-connected systems, which system) is asking it to do things. The control mechanism bases what it allows to happen on this knowledge, and the auditor, having detected a fraud or mistake, uses the knowledge to identify the perpetrator.

This function is known as user authentication, and in an online system is usually achieved by each user "signing on", that is, saying who he is and proving it by:

1. Showing he knows something (password) and/or
2. Showing he has something (operator identification card).

Because neither of these techniques constitutes incontrovertible proof of identity (possibility of passwords being discovered by other than their rightful users, or of operator identification cards being stolen) it is useful for any authentication mechanism to provide a log of its operation which can and should on a regular basis be analysed for irregularities such as attempts at password guessing.

In some cases, a unique sign-on for each user is unnecessary. For example if all the terminals in a large office are installed on individual's desks and all the occupants of the office know who should be sitting where, each user can effectively be identified by his terminal identification. Of course, user sign-on gives added protection even in this situation, and in general, with the key proviso that security mechanisms are not too cumbersome either administratively or for end users, the more checks and cross-checks the better. Banks lock their entrance doors at night even though their money is kept in a heavy steel and concrete vault.

Conversely, good user authentication mechanisms may be important because of factors such as connections to remote or isolated terminals.

RECOGNITION

It should be noted that inter-connected systems access each other's data, and consequently each is often considered to be a "user" by the other. However, as users they are not normally required to 'prove' their identity via a sign-on process. This is for the same reason that terminals themselves do not require it: if verification is necessary, as with a dialed-in terminal or system, it is done as part of the protocol for establishing the connection. Non-dial systems are uniquely identifiable by the physical port through which they communicate.

The process of associating a user identification with an interconnected system for security purposes is called recognition.

SECURITY LOGGING

Both control and authentication mechanisms log their activities so that deliberate attempts at violation of security can be detected by analysis of the log. The term audit or security audit (as distinct from financial audit) is sometimes used to describe this activity.

Authentication Control And Security Logging with CICS/VS

CICS/VS provides a number of security related options to address different authentication and control requirements. These are summarized below, and a fuller discussion is given later on in this section.

CICS/VS has a built-in security mechanism based on use of the CICS/VS Sign-on Table (SNT), which can be used to control:

1. Terminal operator sign-on/sign-off using a four to eight character password for authentication of Sign-on requests.
2. Transaction access by terminal operators.
3. Transaction access to CICS/VS resources, such as files and queues (for command level applications only).
4. Secure transaction and resource access without prior sign-on, as for printers.
5. Access by one CICS/VS system to transactions defined in another CICS/VS system.
6. Access by one CICS/VS system, via an IRC or VTAM connection to resources defined in another CICS/VS system.

In addition to built-in security mechanisms, CICS/VS has the ability to use an External Security Manager (ESM), such as the MVS RACF program product.

CICS/VS will:

7. Allow the ESM to authenticate the sign-on using a password, and optionally an Operator Identification Card.
8. Allow the ESM to "recognize" connected systems by the TCT user identification option.
9. Use the built-in security mechanism of 1, 2, 4 and 5 above or the ESM, based on Sign-on Table and Transaction Table options.
10. Use the ESM to control the scheduling of DL/I PSBs from batch jobs using the shared database facility of CICS/VS or from IRC- or VTAM-connected CICS/VS systems.

CICS/VS has defined a Transient Data Destination (CSCS) which is used as the target for security related logging. Logging is done for:

11. Authentication - sign-on failed (with reason) or succeeded, and sign-off.

These options are now discussed in more detail.

TERMINAL OPERATOR SIGN-ON/SIGN-OFF

Each terminal operator is identified to CICS/VS in an operator Sign-on Table (SNT). The following information is contained in the table:

- Operator name
- Operator identification
- Operator password
- Operator security codes
- Resource level security codes
- Operator class
- Operator priority

Each terminal operator is required to sign on to CICS/VS at a terminal by entering the sign-on transaction code CSSN, together with his allocated password and his name, up to 20 characters in length (see Figure 4.4-1). At certain types of terminal, entering the transaction code only causes CICS/VS to prompt for additional information. The password can then be entered in a non-display or over-struck or print-inhibited field. The operator sign-on procedure is discussed in more detail in the CICS/VS Operator's Guide.

The CSSN transaction code initiates the CICS/VS Sign-on Program (SNP). This program loads the Sign-on Table (SNT) and locates the operator name and password in the table. If these two do not agree exactly, the sign-on is rejected. Successful sign-ons and unsuccessful attempts are indicated by messages to the terminal, and optionally to the transient data destinations CSCS. The latter constitutes a security log; as well as the message numbers, valid and invalid operator identifications and incorrect passwords are recorded.

Once sign-on is achieved, the sign-on program extracts the operator identification (for example, his initials), security codes, operator class and operator priority from the sign-on table. This information is transferred to the Terminal Control Table (TCT) entry for the physical terminal to which he has signed on. This information remains in the TCT entry until the operator signs off with a CSSF transaction.

The three-character operator identification code is used for subsequent operator identification, and the operator priority is used in conjunction with terminal and transaction priorities to establish the overall task priority. This is discussed in more detail in "Priority Processing" in Chapter 3.3.

The operator class specification is used primarily in conjunction with the CICS/VS message routing facility. Messages may be directed to specific terminals, specific operators (by operator identification), or

all operators in a specific operator class. An operator may have more than one operator class.

Messages directed to specific operators, or to specific operator classes, are not transmitted until the particular operator or operators sign on the CICS/VS. Refer to "Message Routing" in Chapter 3.2 for more detail.

The operator security codes and the resource security codes and discussed later in this section.

Sign-on in an MRO Environment

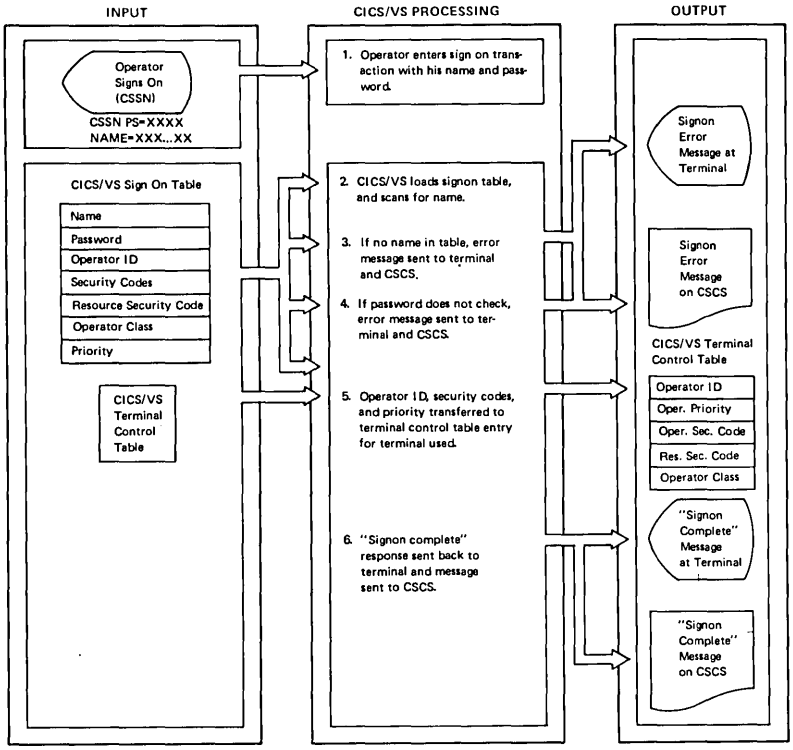


Figure 4.4-1. Operator Sign-on using Built-in CICS/VS Security Support

When the Multiregion Operation feature of CICS/VS is used, each region of the system runs its own copy of CICS/VS and can have its own sign-on table. Thus operators may be defined to some CICS/VS systems, and not to others, and may have different passwords on different systems. For BTAM connected terminals the system to which the terminal is connected is defined by the installation, but for VTAM connected terminals, it is possible for the terminal operator to log on to different CICS/VS system at different times. Having logged on, the operator must sign on also, in order to use secure transactions.

In order to temporarily use the facilities of one system when connected to another, the transaction routing sample program of CICS/VS (the CRTE transaction - see Chapter 7.2) can be used. This is like logging on to the new system, and sign on is similarly required. However when the routing is cancelled the connection and sign-on to the original system is still in effect.

Special Sign-on Formats

CICS/VS supports three special-purpose types of sign-on:

- numeric sign-on, transaction code 7777 - intended for numeric only terminals such as the 7770 Audio Response Unit. Password and name must be numeric. No keywords or delimiters are needed. The password must be exactly 4 characters long.
- 3741 sign-on - the 3741 operator id card is handled in a special way. The data on the card is treated as an operator name, and no password check is made.
- 3270 operator id card - the (non-SNA) 3270 operator id card can be used to trigger a sign-on by defining the oidcard AID (Attention Identifier) in the transaction table as a transaction which invokes the CICS/VS signon program. The card must contain a complete numeric sign-on including the numeric sign-on transaction code (7777), password and operator name.

Note that the CICS/VS sign-on transaction may not provide an appropriate security mechanism for installations using the TCAM access method, since CICS/VS does not (necessarily) have a TCT entry for each physical terminal. This may be because the TCAM queue name does not correspond to a specific terminal, or because the TCT pool feature of CICS/VS TCAM support is used.

Sign-Off

After completing a session with CICS/VS, the operator must sign off using the CICS/VS sign-off command (CSSF) in order to prevent later use (or abuse) of the sign-on the identification and authority at the terminal by someone else. The CSSF command performs the following actions:

- Sets the OPID field of the TCT entry to blanks.
- Clears all operator class codes.
- Clears all resource security codes.
- Sets to zero the operator priority field.
- Clears all operator security codes except code 1.
- If the LOGOFF or GOODNIGHT option of the command is specified, a dialed-in terminal is disconnected, and a VTAM terminal is logged off from CICS/VS (providing the TCT is not defined so as to prevent logoff for this terminal).
- If the GOODNIGHT option of the command is specified, the terminal status is set to RECEIVE (so that it cannot enter transactions, but can be sent messages).

The CSSF command has a numeric equivalent, 8888, and the CSSF command with the GOODNIGHT option has a numeric equivalent, 88888888.

In some circumstances it may be necessary to force a sign-off for a particular terminal. This can be achieved by issuing an ATI (Automatic Transaction Initiation) request for the CSSF transaction. The GOODNIGHT or LOGOFF options cannot be specified by this technique.

The CSSF transaction is forced to run when a routing (using CRTE transaction) from one CICS/VS system to another is terminated. This is so that appropriate logging of sign off is performed.

CONTROL OF TRANSACTION ACCESS BY TERMINAL OPERATORS

The operator security codes defined in the Sign-on Table are used to control transaction access. They consist of a series of numbers ranging from 1 to 24. The function of security codes 2 to 24 is defined by the user, but security code 1 is automatically given to all users defined in the SNT, and is present in the TCT entry when no user is signed on at the terminal. Security code 1 is used, among other things, to allow the sign-on transaction (CSSN) to pass the security check.

Security codes in the TCT are used in conjunction with a security code defined for each application transaction code. A transaction code with a defined security code of 10, say, can be used only by those operators who also have a security code of 10. An operator may have more than one security code. Operator security codes 5, 6, 10, and 12, for example, would enable those operators to use only those transaction codes which have been defined as security 1, 5, 6, 10, or 12. The suggested way in which this facility is used is as follows.

Transactions to be controlled must be grouped into up to 23 categories. The categories can be thougnt of as having names, for example, INVOICE, PAYROLL, STOCK, SYSTEM-CONTROL and the categories are given numbers in the 2-24 range, for example, 2 through 5 for the above categories. Terminal operators have a 'role' within the organization which requires them to have use of certain categories of transaction. Thus a certain operator may need to use the INVOICE and STOCK transactions and would thus be given security codes 2 and 4. Another operator may only need access to the INVOICE transactions and this can be supported by giving him security code 2. Transactions which are given a security code of 1 can be used by any operator and can even be used when the terminal is not signed on. Certain CICS/VS transactions have this security code (for example CSSN and CSSF) but user written transactions other than perhaps a system status display transaction should not normally be given a security code of 1.

It should be noted that this allocation and use of security codes must be coordinated with the other related uses described in this section.

If an operator attempts to enter an unauthorized transaction code, CICS/VS will reject the transaction and send an error message indicating a security violation to the terminal operator. The violation is also logged on Transient Data Destination CSMT. The operator identification, terminal identification, and transaction code used are detailed in the notification message to transient data, as shown in Figure 4.4-2. (See the CICS/VS Messages and Codes Manual for additional information). The master terminal operator may then take appropriate action.

An option in the Transaction Table (PCT) entries indicate whether or not each transaction's access to resources should be controlled. If control is requested then the resource security codes of the operator (defined in the SNT but moved to the TCT at sign-on time) as illustrated in Figure 4.4-1 are compared with the resource security code (defined in resource tables such as the FCT) of the resources the transaction attempts to access. If any check fails, the transaction is abended. The check is only made if the resource is accessed using the command level interface.

As with the operator security codes described above, the terminal operator can have multiple resource security level codes in the range 1 to 24. Resources can have just one code. The resource code must be one of those belonging to the terminal operator for the check to succeed. Note that there is no special use of code 1 for resource security, and no default security code. Hence, if either the operator or the resource being accessed has no resource security level specified, but the transaction requests checking, the access will be refused.

Files, Journals, Transient Data Destinations, Temporary Storage queue prefixes, programs and transactions can all have resource security specified in their associated tables. (Resource security on transactions controls whether they can be started using the START command).

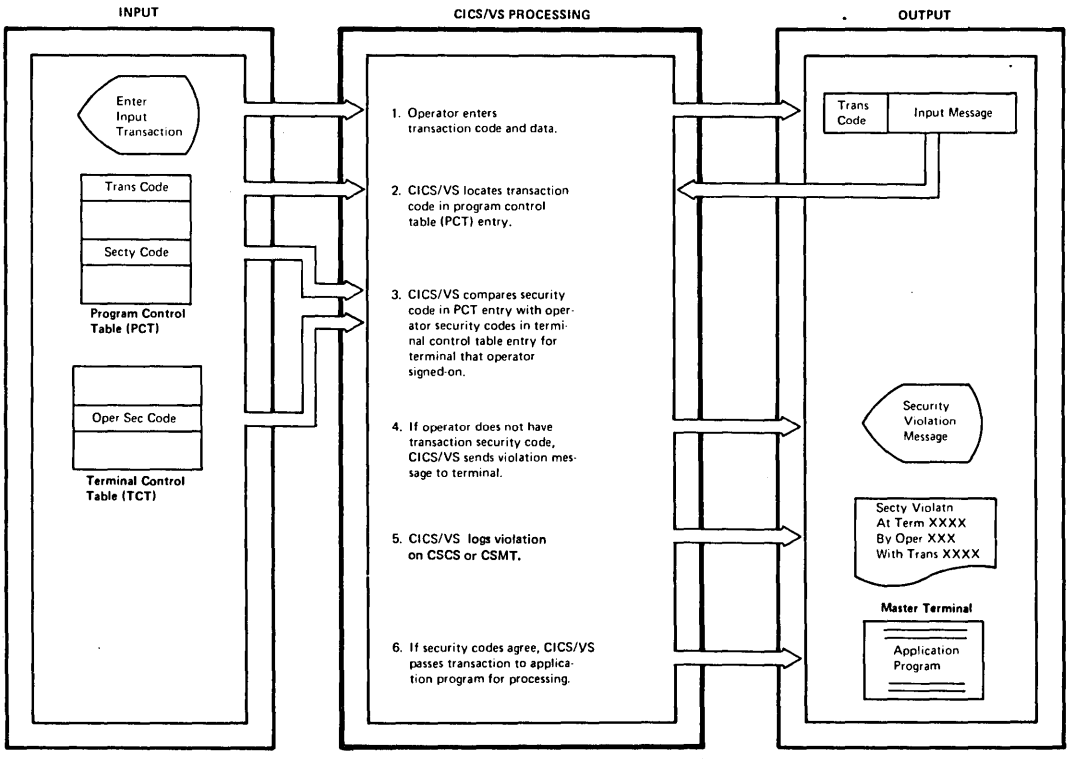


Figure 4.4-2. CICS/VS Control of Transaction Access

Note that a transaction requiring resource security checks must have a terminal associated with it otherwise there will be no information against which to perform the security check. If such a transaction is started without a terminal, it is ABENDed when it first tries to access

a resource. The ABEND is not logged to the security log since the problem is regarded as a set-up error rather than as a security violation.

Care should be taken in use of the resource security facility. Transactions in production status should not normally be defined to require it since the meaning of production status is that the transaction is believed to be correct and therefore a check against what it does is not necessary.

On the other hand, a major use of resource security is related to application development and test. Transactions that are still under test should utilize resource security in order to detect accidental access to incorrect resources. The recommended way to achieve this is to determine those resources that the transaction (or group of transactions) under test can access and allocate them one or more resource security codes. Terminal operators allowed to run the transaction for testing purposes can then be given these resource security codes. The CICS/VS Command Level Programming interface must be used in the new applications, so installation procedures and standards should be in place to ensure this is adhered to. For further discussion see "Security Control over the Application Programmer".

A case where it is appropriate to use resource security for production applications is to control general purpose programs such as file browsing programs which take the file name to be accessed as a parameter. If resource security is specified for the transaction, operators who are allowed to use the browse transaction can only do so on files they are permitted to access by the combination of resource security codes in their SNT entries and the resource security codes in FCT entries.

Resource security checks are also performed against remote resources. The definition of the remote resource carries a resource security code and a match must be found with the operator resource security codes before access of that resource (via MRO or ISC function shipping) is allowed. If a remote resource is accessed using the SYSIDNT option on the command then there is no resource definition available against which the check can be performed and in such cases the access is always rejected.

Use of Resource Security in CICS/VS Supplied Transactions

Three CICS/VS transactions normally use resource level security. These are CSMI - the mirror transaction used for function shipping; CECI - the Command Interpreter transaction used for program development; and CEDF - the Execution Diagnostic Facility of CICS.

The facilities of the Command Interpreter are described in detail in the Application Programmer's Reference Manual (Command Level). However, in essence, it allows CICS/VS commands to be issued directly at a terminal. Clearly this is a powerful facility and open to abuse if access, not just to the transaction itself, but by the transaction to CICS/VS resources, is uncontrolled. Because the command interpreter normally uses resource security, a terminal operator allowed to use it can only successfully execute commands which operate on resources he is allowed by the resource security check to access.

The Execution Diagnostic Facility is described in detail in the Application Programmer's Reference Manual (Command Level) and the Entry Level System User's Guide. It allows the terminal operator to trap and modify the commands of an application program being debugged. Since the

modifications allowed include modification of resource name, the resource level security mechanism can be used to prevent access to incorrect resources. Note that the transaction being debugged need not specify resource security, provided CEDF specifies it, since CEDF can force resource security for the transaction. Security during application development is further discussed in the section "Security Controls over the Application Programmer".

TRANSACTION AND RESOURCE ACCESS WITHOUT PRIOR SIGN-ON

Certain terminals, for example, printers and fixed format terminals, such as the 3614, cannot support the sign-on transaction. Therefore the sign-on process cannot be used to get security codes into the TCT entry for the terminal. To overcome this limitation, CICS/VS allows specification of security codes (both operator security codes and resource security codes) to be permanently associated with the TCT entry. This allows the control mechanisms described in the previous two sections to operate just as though a valid sign-on had been received. Note that operator and resource security code specifications in the TCT entry can be made for any terminal, thus a terminal in a highly secure area, such as the computer room can be used for secure transactions such as CSMT without requiring sign-on. The sign-on and sign-off transactions will not override any permanently defined security codes specified in the TCT; CSSN will not run on a terminal that has any preset security values.

Note that there are several uses of resource security codes described in this section and their allocation and use for each must be coordinated. For example eight codes may be allocated for specialized security requirements in production transactions, eight for temporary allocation to applications under development or test, and eight for control of the resources accessible by other CICS/VS systems connected via IRC or VTAM.

CONTROL OF TRANSACTION ACCESS BY A CONNECTED CICS/VS SYSTEM

When multiple CICS/VS systems are connected together, the transactions of one system can be protected from access by terminal operators connected to other systems. This control is based primarily on the identity of the other CICS/VS system, and only in specific instances also on the name of the actual terminal operator.

Control based on the other system's identity is achieved in the same way as for terminals which cannot sign on, that is, by coding the security codes on the TCT entry representing the connection to the other system. The security code of the transaction requested by another system is compared with the security codes in the TCT entry in exactly the same way as described previously for terminal operators, that is, the request is rejected if the transaction security code is not one of those occurring in the TCT entry.

Security When Transaction Routing is Used

When a transaction on a local system is requested by a remote system by means of the transaction routing mechanism, a surrogate TCT entry is built in the local system to represent the terminal. The security codes entered into this surrogate are those preset (if any) in the local TCT definition of the terminal. These codes are marked as permanently defined security codes (even if null) and so cannot be changed by sign-on or sign-off.

The security check at transaction initiation is executed using the security code in the TCT entry representing the connection to the remote system. If a match is found then the remote system has the right to execute this transaction and the transaction is initiated. Otherwise the request is rejected. Note that it is the responsibility of the remote system to ensure that only those terminal operators with the correct security classification are allowed to access the transaction that is to be executed. In other words the definition of the transaction in the remote system must have a security code, so that the security check performed when the initiation is requested verifies the terminal operator's authorization to invoke this transaction.

If the transaction has been defined on the local system as requiring resource security checks then all such checks are performed against the resource security code in the surrogate TCT entry. This would normally be null and so the resource access would be rejected. This is necessary because the system executing the transaction does not know the resource security classification of the terminal operator nor can it rely on the terminal-owning system making appropriate checks (as it does for transaction initiation) because that system does not have the required information to perform such checks.

Security When CRTE is Used

Control based on the identity of the terminal operator is used additionally when the terminal operator uses the routing transaction CRTE to route all requests to this system. Having established the routing, the minimum security code, code 1, is established in a dynamically-built surrogate TCT entry in the system being accessed, and in order to use any secure transactions the terminal operator must sign on to the system. The transaction access must pass both link and operator sign-on security checks before the access is allowed.

Similarly if the transaction is designated as requiring resource security checks then both link and terminal operator resource security checks must indicate that access is autorized before access to the resource is allowed. This ensures that the terminal operator signing-on to the surrogate TCT entry cannot increase the effective security classification of the intersystem connection.

CONTROL OF THE RESOURCE ACCESS BY IRC- OR VTAM-CONNECTED SYSTEMS

CICS/VS performs a resource security check when a request is function-shipped from an IRC- or VTAM-connected system. This operates in the same way as already described for transaction access to resources, that is, the resource security code defined for the resource being accessed must be one of the codes present in the TCT entry for the connection. However an important advantage of the check against a CICS/VS system compared to that against a transaction is that even a knowledgable application programmer over whom no administrative controls are exerted, cannot bypass the security check (apart from considerations of system integrity) provided he cannot modify the system containing the resource. Similarly, when the resource security mechanism is used to protect an ISC link, the other system cannot bypass the security restrictions, and this allows the link to be set up even when some data on the system is highly sensitive and must not be allowed to travel across it.

Since a remote system which might access resources on the local system does not sign on in the sense described earlier for terminal operators, the resource security codes which apply to, and must constrain, the connected system are not derived from the Sign-on Table (SNT) but are coded in the TCT entries representing the connections. The way in which the security check works is identical to that described previously, that is, the resource security code of the resource must be one of the resource security codes coded in the TCT entry for the system attempting to access the resource. Note that the mirror transaction CSMI is used to access the resource, hence this must indicate in its Transaction Table (PCT) entry that resource security is required in order for the check to be made.

Control of DL/I data, that is, data managed by DL1/DOS/VS or IMS/VS/DB cannot be achieved using the CICS/VS built-in security mechanism. This is because the CICS/VS resource security check relies on a field in the resource definition control block, which in the case of DL/I data is not a CICS/VS control block. Hence control of a CICS/VS transaction's access to PSBs requires use of the External Security Manager facility of CICS/VS.

THE CICS/VS CONCEPT OF AN EXTERNAL SECURITY MANAGER

For some installations which use a variety of software products it is inconvenient to have separate security mechanisms for each product. Hence CICS/VS will allow an External Security Manager (such as RACF on MVS systems), which may be managing the security of other system facilities such as batch and interactive services, to selectively take over control of CICS/VS security. Additionally some installations have specialized security requirements; the Security Program (DFHXSP) receives control for all security related events in CICS/VS and can be tailored by the user.

CICS/VS provides several versions of the security program.

1. a dummy version that provides no security checks (access is always authorized)
2. a version that provides CICS/VS security checks only

3. a version that provides CICS/VS or RACF checks which can be used on MVS systems, provided the appropriate level of RACF (Release 1.3 or later) is installed. (This version is supplied only with CICS/OS/VS).

The ESM can provide the following functions to CICS/VS:

1. A function to initialize the interface to the ESM. CICS/VS has a defined set of actions if the ESM indicates it is unavailable, which permit the installation to provide restricted service if desired.

2. A function which authenticates userids and constructs and returns to CICS/VS the address of an External Security Profile.

3. A function which recognizes "other-system" userids and constructs and returns to CICS/VS the address of an External Security Profile.

4. A function which cancels such profiles.

5. A function which returns the user identification to be associated with the CICS/VS system.

The interface to RACF supports all these functions. For further information see RACF General Information, GC27-0722.

User Supplied External Security

The CICS/VS external security manager consists of two parts, one of which performs standard CICS/VS security checks and the other is code included from the copy module DFHXSC. DFHXSC is provided as a dummy for use on VSE and OS/VS1 systems and as an interface to RACF on MVS.

If a user wishes to supply his own security manager either the whole module, DFHXSP, may be replaced or the copy module DFHXSC.

DFHXSP is entered during initialization, and at sign-on and sign-off of a user if external security has been asked for. It is entered for transaction authorization, PSB verification and resource security checking. The code represented by the copy module DFHXSC is entered only when external security checking has been requested.

By supplying a replacement for the module the user could replace the CICS/VS password facility. For example, the user could keep a table of valid passwords in a file along with other information such as the time the password has been in force, and request a new password after a certain length of time. The time of day when a transaction can be run can be controlled. At sign-on time a block related to the user's ability to access resources could be built, and its address stored in the address given in the parameter block. This address will then be available at resource check time and at sign-off, when it may be released. The interface that CICS/VS provides to a user-supplied external security manager is described in detail in the System Programmer's Reference Manual.

AUTHENTICATION USING AN EXTERNAL SECURITY MANAGER

In order to perform authentication of CICS/VS terminal operators by an External Security Manager (ESM) but to nevertheless continue to use the other features of the CICS/VS sign-on table such as operator id, operator and resource security codes, and operator priority, the External Security operand of the Sign-on Table must be specified for each user, or alternatively, the operand must be placed on the INITIAL statement of the Sign-on Table and then explicitly negated for those operators for whom it is not required.

Alternatively, if the other Sign-on Table features are not required, the operator need have no Sign-on Table entry, and a special table entry can be specified, indicating that operators whose names do not appear in the sign-on table are to be authenticated using the ESM. This entry may also supply default values for the other Sign-on Table features.

If ESM authentication is used, additional information may be requested such as new password (because the old password has expired), or the insertion of the operator id card, the CICS/VS sign-on program will automatically prompt the terminal operator if necessary.

The CICS/VS sign-on/sign-off logging to transient data destination CSCS is still performed with ESM authentication in addition to any logging to central facilities that the ESM may do.

If the External Security Manager is unavailable, CICS/VS ignores the External Security Required flag in Sign-on Table entries and attempts to authenticate sign-on requests using the built-in CICS/VS facility. This will only succeed if the operator signing on has a password defined in the Sign-on Table entry as well as the External Security Required flag. Operators without a Sign-on Table password cannot sign on when the ESM is unavailable.

ESM RECOGNITION OF CONNECTIONS

When it is desired to support a security check using an ESM but the CSSN transaction cannot be used for a particular connection, it is possible to cause the ESM to "recognize" the connection as soon as it is established. The rules for this are:

- ISC (Inter Systems Communication) connection to another system. Recognition is attempted if the TCT entry for the system specifies a user identification (as specified in the XSNAME operand of the DFHTCT TYPE=SYSTEM macro) to be associated with the connection.
- MRO (Multiregion Operation) connection to another CICS/VS system. The TCT entry for the system does not require the user identification operand. Recognition is always attempted based on a user identification value obtained by the interregion communication mechanism from the External Security Manager for the connecting system. If a User Identification is specified on the TCT entry it is compared with the derived value and the connection will only be made if the two are identical.
- Shared database connection to a batch database region. As for MRO.

Transient Data Destination CSCS receives sign-on/sign-off messages for ESM recognition of connections.

If the External Security Manager indicated it was unavailable at CICS/VS initialization all recognition requests are treated as null operations.

TRANSACTION ACCESS CHECK USING AN ESM

In order for the transaction access check to be made via the External security Manager (ESM) three conditions must be satisfied:

- The Program Control Table (PCT) entry for the transaction must specify that external security is required.
- The external security manager must be active. CICS/VS supports a mode of operation where external security is requested on CICS/VS tables etc, but the ESM is not available. Transactions which request External Security on their PCT entry can be used in this mode provided they also have an operator security code specified. This will only be used in this circumstance.
- There must be an External Security Profile associated with the terminal at which the transaction is requested. This may have been derived either via a normal user authentication process using the CSSN command, or via ESM recognition of the connection when it is established.

If either of the first two of these conditions is not met, a normal CICS/VS check using operator security codes is performed.

If the "recognized" connection's user identification is the same as the local system's user identification, then the assumption is made that the requestor of a resource is the same as the owner of the resource and further security checks are unnecessary. Hence if the ESM maintains tables relating requestors to resources (as RACF does) there is no need to keep a table of all resources up-to-date just so that the obviously valid accesses can be allowed.

One reason the above set of conditions for an ESM transaction access check is imposed is so that migration to use of an ESM is possible in a gradual way. At a given point in the migration, it is possible to have a mixture of terminal operators authenticated using the ESM or the CICS/VS SNT, and a mixture of transactions, some requesting an external security check and some a CICS/VS security check based on operator security codes. A possible migration might be to arrange for each using department's terminal operators first to be authenticated via the ESM, then to define their transactions to the ESM, and finally to change the department's transactions definitions in the PCT to request external security. Provided none of the old CICS/VS checking information is deleted until migration is complete, it is possible to fall back on to the CICS/VS check (for all departments however) simply by disabling the External Security Manager. No re-assembly of CICS/VS tables is required.

If the External Security access check fails CICS/VS will log the violation to the master terminal transient data destination in the same way as if the check had been made using the CICS/VS built in security mechanism.

DL/I DATA BASE SECURITY CHECKING

When access to DL/I data on a CICS/VS system is to be controlled for security reasons, the External Security Manager facility of CICS/VS is used. Access is controlled only when DL/I requests are function-shipped from outside the CICS/VS system, and not when requests are made by CICS/VS transactions on the same system as the DL/I data.

There are three situations:

- Shared Database access by a batch DL/I program running in a different partition/address space of the operating system.
- Multiregion Operation access by a transaction in another CICS/VS system accessing DL/I data on this system.
- Intersystems Communication access by a transaction in another processor system accessing DL/I data on this system.

In each case the access is checked against the previously "recognized" identification of the requesting program/system.

If access is refused the information is logged to CSCS.

As with the transaction check made using the External Security facility of CICS/VS, the check is bypassed when the requesting user identification equals this system's user identification.

LOGGING TO CSCS

CSCS is the Transient Data destination name that CICS/VS uses for recording security-related events. The installation may choose to do one of the following in defining CSCS:

1. Omit it from the DCT. In this case most of the messages are not sent, however for compatibility with releases of CICS/VS which did not use CSCS, some messages are sent to CSMT. This is applicable for installations for which security is not of concern.
2. Put into the DCT as INDIRECT to CSMT. This will result in all security messages going to CSMT. In particular messages which went to CSMT prior to the introduction of CSCS will still be there, and will not be duplicated. This is applicable if security is not handled by a separate function in the installation, but someone regularly browses through CSMT looking for problems such as frequent transaction abends or security violations.
3. Direct it to disk or tape. This will result in security related messages going to a special dataset. Some messages which went to CSMT prior to the introduction of CSCS will no longer be found there but will appear on CSCS. This is applicable to installations which are concerned over security and make special provision for analysis of information related to security.

Security Controls Over Application Programmer

The application programming team for an online application have the responsibility to create programs to a specification. Their job on a program is finished (apart from changes to the specification) once the program is considered by the installation to be trusted for production use.

What "trust" means here is that the program is believed by the installation to do what the specification says it should do. However, a program can violate the specification without necessarily violating even the most stringent security checks that could be made on the program's access to resources. For instance, the specification for the salary update transaction may say that the program should read the employee record, verify the current salary, add the increment to it, and re-write the employee record. If, for the programmer's own record, the verify is omitted, and the increment is added twice, no security check would detect this, but the program would not be to its specification. Hence if the programmer himself is not fully trusted, his programs must not merely have security checks performed against them, they must be reviewed before they are used in production.

To make this effective, the application programmer must be unable to install or modify programs that are in production. Security protection that is not within the scope of CICS/VS to provide must therefore be used on the object libraries of the production CICS/VS system, and (so that source corresponding to the object libraries is not corrupted) also on the source libraries.

Given the above, the role of security checking against application programmers must be restricted to prevention of accidental or deliberate damage to production data bases by programs under test.

CICS/VS provides three tools for this:

1. EDF. The Execution Diagnostic Facility of CICS/VS allows a terminal operator who is permitted to run the CEDF transaction to trace, test paths of, and to some extent modify an application program online. If this facility is used on the same CICS/VS system as normal production work, resource level security checking can be imposed on the application so that only test data can be accessed. Although EDF allows modification of storage areas, only user storage belonging to the task can be modified. Hence provided the program being debugged does not damage the system or access incorrect resources using the macro level API, it should be impossible for the programmer using EDF to do any damage.

2. MRO. If complex programs are being written and tested under CICS/VS, use of EDF alone may not provide some installations with a sufficient guarantee of the safety of the production data. In this case the multiregion operation feature of CICS/VS may be used. Provided the program being tested operates in its own CICS/VS region, it can be prevented from damaging any production data by applying a security check against the region. This method has the advantage that the program can refer to production database names but be routed to test data. EDF can be used to debug the program.

3. Command Interpreter. In the course of developing an application program, the command interpreter can be used to try out the statements that the program will use. The interpreter transaction will access only resources the operator is allowed to access, thus it will be restricted to the same resources that EDF and the application being developed can access.

PROGRAM REVIEW PROCEDURES

When the programmer has developed and tested the program to his satisfaction, installations which regard security as important will probably want a review of the source code of the program, and a test of the program under installation controlled conditions before the program is used in production mode. Even when security is not regarded as important, such procedures will help to improve the quality of the code. The following points should be considered for CICS/VS application programs:

- The program should use only command level API facilities of CICS/VS, or the programmer should clearly justify why it does not do so.
- The program should use literal resource names so that it is clear which resources it is accessing.
- The program should be tested with resource security active.

Additionally, structured programming or equivalent methodology and use of high level languages can simplify and improve the effectiveness of review procedures.

Audit Facilities

The audit process is aimed at detecting irregularities post-facto. Therefore audit facilities are aimed at recording information for the audit process, and the auditability of an application depends on judicious use of such facilities.

CICS/VS provides journaling facility specifically for the recording of information to be processed later. Journal control creates a separate file on tape or disk (which can be copied to tape offline) from journal write requests issued by user programs and/or CICS/VS management modules (for instance, File Control), and/or DL/I database management modules. Logical records from multiple executing transactions are intermixed on the journal and even within physical journal records. To facilitate subsequent processing the information is in hex format and contains time stamp information and end user origin information. Multiple journals are supported. Journals are identified by a number (01-99), and journal 01 is refered to as the system log.

JOURNALING BY CICS/VS MANAGEMENT MODULES

All journaling by CICS/VS management modules is optional.

Most of the journal record types written by CICS/VS modules are related to recovery and thus are turned on by recovery related options. They are always directed to the system log and contain information geared to emergency restart. File Control and Terminal Control, however, support automatic journaling; outputting of automatic journaling records is controlled by a separate resource-related option and the records can be directed to any journal. These options could be used for example to record all changes to a file or all transaction input requests.

JOURNALING BY DL/I

If DL/I is in use, journaling is required. DL/I records all changes to DL/I data. The records are written unconditionally to the system log.

JOURNALING BY USER APPLICATION

This can be used to record stages in an application's processing, for instance critical or security sensitive sub-functions.

CORRELATION WITH CSCS

Journal records identify the terminal which started the task making the request. Auditability is impaired if non-terminal related tasks perform critical functions. Application journaling should be used to make up the deficiency.

The CSCS log records discussed earlier can be used together with journal records to determine which user was logged on when the log record was produced by using the time stamps which occur in both.

Part 5. Recovery and Restart Design

Chapter 5.1. Principles of CICS/Recovery and Restart

Recovery and Restart Overview

Two of the most significant aspects in the design of applications to execute under control of CICS/VS are the recovery from errors and the subsequent restart. Depending upon the complexity of the online application and the CICS/VS facilities that are used, the design of recovery and restart procedures may represent little effort on the part of the design team or may be a major element of the overall system design.

For example, for an inquiry application that only retrieves information from data bases and does not update that information or add new information, recovery and restart generally involves reestablishment of the CICS/VS system to its status at the time of failure.

However, for an online application that retrieves information from data bases, and updates, deletes, or adds records to the data bases, the information, which must be recorded by the system during normal operation, may be extensive. This information is used following a system failure, to recover the status of the various data sets and data bases to a defined point, such that all necessary updates, deletions, and additions up to that point have been completed. Recovery of such an online update and addition application can involve considerable system design effort.

Regardless of the type of application, the system design team should consider the effect on the online application of each of the types of error or failures described in this chapter. The team should determine whether any action is necessary to design procedures, which ensure that information necessary for recovery is recorded during execution.

Many online applications process adequately with no recovery and restart procedures. However, the value of a well-designed online application containing recovery and restart procedures is truly realized when an error arises. Regardless of how well a system is designed, failures will occur because of program errors, transaction interlocks, data transmission errors, I/O access errors, or system component failures (such as the processor). An online system that allows for errors and has recovery and restart procedures to handle them is better able to continue operation when errors occur.

A measure of the performance of an online system is its ability not only to provide satisfactory response time, but also to provide online access to applications from terminals over extended periods.

While errors must inevitably occur, they should not be permitted to disrupt the availability of online applications for long periods. Good recovery and restart design procedures can minimize the amount of time a system is down following a failure. They can also minimize the effect of a system going down at an unscheduled point, by ensuring that the integrity of any data sets or data bases is maintained.

The recovery and restart capabilities of CICS/VS are designed to meet any type of software or hardware failure. At the same time they are designed so that the individual installation may choose to take

advantage only of those CICS/VS capabilities that are relevant to the particular RAS requirements of that installation. In addition they are designed so that they can readily be extended to provide unique RAS capabilities for a particular installation.

Because of the flexibility and extendability, it is necessary to provide a detailed description of what CICS/VS provides in various error, recovery, and restart situations, and how this may be used or extended by the individual installations.

The following recovery/restart conditions are discussed:

- CICS/VS Error Return Code processing (for example, file errors)
- Terminal I/O errors
- Program check handling
- Transaction abend handling
- Operating-system abend handling
- CICS/VS termination
- Normal CICS/VS cold start
- Warm start
- Emergency restart

Before treating each of these conditions in turn, it is necessary to provide a general overview of some of the principles that underlie CICS/VS RAS capabilities. This overview is provided in the next section of this chapter. Subsequent sections then treat each of the above conditions.

Principles Underlying CICS/VS Recoverable Resources

Before it is possible to describe the functions provided by CICS/VS to handle error, recovery, and restart conditions, it is necessary to describe certain principles underlying CICS/VS recoverable or protected resources. By "recoverable or protected" is meant those resources, particularly data sets, whose integrity is guaranteed by CICS/VS in the case of failures.

This section describes the following underlying principles:

- Defining Protected Resources
- Synchronization Points
- Enqueueing
- Logging
- Activity Keypointing
- Message Integrity

DEFINING PROTECTED RESOURCES

This section describes what is meant by "protected or recoverable resources", which CICS/VS resources can be protected, and how this protection is defined. The words "protected" and "recoverable" are used interchangeably.

If a given transaction reads and updates CICS/VS resources, such as an intrapartition transient data destination, a CICS/VS file, a DL/I data base, and so on, it will usually be the case that this group of actions should either be carried out in its entirety or not at all. However, for some reason, such as a transaction abend or a total system failure, it may not be possible to complete these actions once started. This situation may have serious implications.

Consider the following example.

An order-entry system is designed so that when an order for a particular product is entered from a terminal, an inventory file is queried and decremented; orders for more stock and to dispatch the goods are prepared where necessary. If the transaction should fail before the order to dispatch the goods is prepared, the warehouse could be ordering more goods than it is dispatching, with potentially serious results.

To guard against this sort of problem, CICS/VS allows resources to be defined as protected or recoverable. The implication of this is that if a transaction that modifies such a resource cannot complete, for whatever reason, all modifications made by that transaction to such resources will be backed out. In the above example, the inventory file may have been defined as a protected CICS/VS file, whilst the reorder and dispatch orders may have been represented by records on recoverable intrapartition transient data queues. Note that it is the resources themselves, not the transactions, which are regarded as protected.

The following types of CICS/VS resource can be defined as protected:

- CICS/VS files: each file may be defined as protected or not.
- Intrapartition Transient Data Destinations: each destination may be defined as recoverable or not.
- Auxiliary Storage Temporary Storage: each DATAID may be defined as recoverable or not.
- DL/I data bases: all DL/I data bases are recoverable.

Note that, with the exception of DL/I data bases, recoverability for all CICS/VS resources is optional. Because recovery involves some execution-time performance overhead, recovery should only be specified on those resources for which it is really required. Recovery is specified at system-definition time in the appropriate entry in the system tables - the File Control Table, Destination Control Table, and Temporary Storage Table respectively for file, Intrapartition Transient Data, and Auxiliary Storage Temporary Storage (for full details refer to the CICS/VS System Programmer's Reference Manual).

In the order-entry example described above, we saw how in the event of a transaction or system failure, all the changes made by the transaction to protected resources would be backed out as though the transaction had not run. As we shall see in the section on enqueueing, there are certain implications arising from the use of protected resources that should be borne in mind when writing a long-running transaction that modifies protected resources. To overcome these problems, CICS/VS provides the concept of Logical Units of Work (LUW's) and Synchronization Points, by which means long-running transactions may be divided into a sequence of shorter units for recovery purposes. An LUW is a unit of work, which will be regarded as a logically related sequence of actions as far as the CICS/VS recovery mechanisms are concerned. That is, changes made in a completed LUW will not be backed out should the transaction or system subsequently fail, whereas all changes in an uncompleted LUW will be backed out. A Synchronization Point is that point in the flow of the transaction that marks the end of an LUW and the beginning of the next one.

For a short-running transaction, the whole transaction will be a single LUW and the only synchronization points will be the implicit ones at the start and end of the transaction. For a long-running transaction, the user may explicitly specify intermediate synchronization points, and therefore define the intermediate LUW's, at whatever points in the logic of the transaction permitted by the recovery requirement of that particular installation.

A synchronization point may be defined by any of the following forms:

- A SYNCPOINT command.
- A DL/I call with the TERM or T function code. Such a DL/I call is logically equivalent to a SYNCPOINT command, although it is, of course, only permitted if the task is currently scheduled to use a particular DL/I PSB. A DL/I call is not equivalent to a SYNCPOINT command if the PSB that is being terminated is a read-only PSB that resides on the local system in a shared data base environment.
- The end of every transaction is implicitly a synchronization point.

In deciding whether or not to specify intermediate synchronization points, and if so, where to position them, the system designer and application programmer must both be aware of the following implications:

- Any changes made to protected resources during an LUW will be backed out should the transaction or system fail during the LUW, so that the resources are left in the state that they were in at the start of the LUW.
- It is good practice to ensure that all other aspects of the transaction, such as user storage and non-protected resources, are also in a self-consistent state at synchronization points. This will facilitate extendability of the application program design in such ways as making non-protected resources protected, splitting the transaction into separate smaller transactions, or restarting the transaction from the last synchronization point.
- User-specified synchronization points generate an implicit DL/I TERM call, and vice-versa. This is necessary to ensure that the CICS/VS and DL/I resource changes are synchronized together. Thus it may be necessary, if required, to reschedule DL/I after a user-specified synchronization point.

A discussion of the implications of synchronization points with respect to enqueued resources and deferred work is given in the following section of this chapter.

ENQUEUEING

Consider the following example of two tasks which update the same record in a data set. Task A gains exclusive control of the record, updates it, and releases exclusive control. Then task B gains exclusive control, updates the record, and completes normally. However, task A (a longer-running task than task B) could not complete normally because of system termination. Consequently, the effect of the partially completed task A must be backed out on restart. When task A's update is backed out, the update of task B will also be backed out, erroneously.

To avoid this, CICS/VS provides a protection facility to enqueue a task on specific elements of protected resources which are being updated, deleted, or added. The enqueue applies until the end of a logical unit of work, and protects records from subsequent modification by other tasks, until it is determined that the logical unit of work is completed.

Thus in the above example, when task A issues a file control GET for update, CICS/VS enqueues explicitly on that record. It then reads the record from the data set. When the task issues the subsequent PUT, to update the record, CICS/VS does not dequeue its use of that record until task A indicates completion of a logical unit of work. CICS/VS then dequeues the task from the data set record.

In the meantime, if task B wishes to update the same record, when CICS/VS enqueues on that record, the task is placed on the suspended task chain by CICS/VS until task A completes and dequeues from the record. Task B then gains control of the record and carries out its update. If the system terminates before the completion of task A, the task A update can be backed out without affecting other processing of the record because task A had exclusive control of the record of termination.

Enqueueing on the record for the duration of the LUW, therefore, protects the record from possible loss of an update made by a completed task, because of backout of the update of a partially completed task.

The same procedures also apply to additions and deletions.

Similar protection is carried out by CICS/VS for transient data intrapartition destinations. Only one task at a time is permitted to access a protected intrapartition destination for input and one is permitted to access it for output. The destination is regarded by CICS/VS as two separate resources - one input resource, and one output resource.

Similarly enqueueing and dequeueing control is performed for modifications to Auxiliary Temporary Storage.

By serializing updates as described above, the integrity of the data set following backout on emergency restart or transaction abend is protected. However, this may have some effect on performance if several tasks are operating on the same resource at some time during their combined total period of execution. This effect on performance must be evaluated in terms of improved integrity.

The user should also recognize the possibility of a lockout occurring if several tasks attempt to update two or more records concurrently in a protected data set, and the records are not accessed in the same sequence by each task. This is illustrated in the following example.

Consider an order-entry application, which, in the same CICS/VS transaction, accepts orders for several products. This transaction updates the stock availability for each product in a product data set that is defined as protected. If one terminal enters a transaction for orders against product numbers 638, 815, and 1068, the application program will update those product records. CICS/VS file control will enqueue against each record until the task passes through a user-synchronization point, or terminates. If another terminal also enters a transaction at the same time for orders against product numbers 501, 1068, and 815, an enqueue interlock will occur on product numbers 815 and 1068, and neither task will terminate. It will appear to the operators of these terminals that the system has gone down, while in fact it is still processing other terminal transactions successfully.

To solve this enqueue deadlock problem, the CICS/VS time-out and dynamic transaction back facilities may be used, or the application may be programmed or used so as to avoid it arising.

The CICS/VS facilities are used by specifying DTIMOUT=time-value and DTB=YES in the DFHPCT TYPE=ENTRY macro for the transactions. In a deadlock, CICS/VS will abend one transaction after the specified time, and back out any changes it has made. The abend frees the lock, which allows the other transaction to complete. The abended transaction may then be restarted.

The restart will be carried out automatically by CICS/VS if RESTART=YES is specified on the transaction's DFHPCT TYPE=ENTRY macro and an appropriate DFHRTY program is written. The terminal operator will be unaware of the restart. Alternatively, the terminal operator can be allowed to restart the abended transaction.

The programming and use techniques for avoiding deadlock are:

1. The terminal operator should always enter product numbers in the same sequence (such as ascending sequence).

2. The application program first sorts the input transaction contents so that product numbers are ascending, before processing begins.

3. The application program issues a user SYNCPOINT command after processing each product order in the transaction.

The first solution requires special terminal operator action which may not be practical within the constraints of the application. (For example, orders may be taken by telephone in random product number sequence.)

The second solution requires additional application programming, but imposes no external constraints on the terminal operator or application.

The third solution requires less additional programming than the second solution. However, by issuing a user sync point, it implies that previously processed product orders in the transaction are not to be backed out if a system or transaction failure occurs before the processing of the entire transaction is completed. This may not be valid for the application, and raises the question as to which products in the transaction were processed (orders accepted) and which were backed out by CICS/VS. If the entire transaction must be backed out, either a user sync point should not be issued, or only one product order should be entered in each CICS/VS transaction.

Of the three solutions, the second solution (sorting product numbers into ascending sequence by programming) is most widely accepted.

The possibility of an enqueue deadlock occurring exists for any application which processes several application-oriented logical units of work (product orders), within one CICS/VS logical unit of work.

DL/I Scheduling

When DL/I data bases are being used rather than CICS/VS files, there are two scheduling methods available. One of them, Intent Scheduling, cannot give rise to transaction deadlock. When a schedule call is issued to DL/I, the processing intent of the PSB is checked against those PSB's currently scheduled; if any possible conflict exists, the second transaction is not scheduled. This mechanism ensures that, not only are deadlocks impossible, but that backout of failing transactions can always be performed. A transaction that cannot be scheduled because of PSB intent conflict must wait until the conflicting transaction terminates its connection to DL/I (either by terminating, issuing a DL/I TERM call, or specifying a CICS/VS synchronization point).

The other scheduling method, Program Isolation, can give rise to transaction deadlock. The mechanism of this is explained in "Simultaneous Update Protection" in Chapter 2.3. When a transaction deadlock occurs, one of the tasks is abnormally terminated so that the other may continue. Any changes made to protected resources by the task that abends may be backed out by the Dynamic Transaction Backout facility. The terminated task may optionally be restarted, without terminal operator intervention, by the Transaction Restart facility. Further details on Transaction Restart can be found later in this chapter.

In the event of a program isolation deadlock, one transaction must be abnormally terminated. Two types of transaction will be involved, mirror transactions acting on behalf of the shared data base region, and other types of transactions. The method of choosing the transaction is as follows:

- If the transactions are of the same type it is arbitrary which transaction is abnormally terminated.
- If a mirror transaction and another type of transaction are deadlocked, the non-mirror transaction must be abnormally terminated.

LOGGING AND DEFERRED WORK

So far, we have seen how certain of the resources defined in the CICS/VS system can be defined to be protected, which entails the backout of uncompleted changes after a system or transaction failure, and how the enqueueing mechanism is necessary to ensure that the backout of one transaction's changes can be accomplished independently of another. This section describes two concepts, logging and deferred work, that are necessary to ensure that the backout mechanism has sufficient information about the status of the resources, at the last synchronization point of this transaction, to allow the backout to be performed.

Logging

If protected resources are to be backed out to their status at the last synchronization point, information must be kept to enable this to be performed. This keeping of information is known as logging. CICS/VS has two forms of log, the System Log and the Dynamic Log.

The System Log is used during emergency restart to perform backout of all tasks that had not completed processing (that is, were "in-flight") at the time of a total system failure.

The Dynamic Log is used during dynamic transaction backout to perform backout of an individual abending transaction, whilst the rest of CICS/VS continues normal operation.

The System Log is a standard CICS/VS journal, with a journal identification of 1. As such, it may be on tape or on disk. In writing information to the System Log, CICS/VS uses standard Journal Control facilities. In order to ensure that, whatever form of system failure might occur, it will still be possible to perform emergency restart backout, CICS/VS ensures that log information has actually been written to the external medium before it actually performs any destructive change to a protected resource.

Information written to the System Log includes "before images" of changed or deleted file records, the status of intrapartition transient data and temporary storage destinations at synchronization points, and replaced temporary storage records. By means of the deferred work mechanism described below, CICS/VS minimizes the amount of information that must be written to the System Log. In addition, start- and end-of-task records are written to the log to enable the emergency restart mechanism to determine which tasks were "in-flight" when the system failure occurred.

The Dynamic Log is a main storage log organized on the basis of a storage area for each task that modifies protected resources. The size of this storage area can be specified, with a default of 500 bytes being assumed. However, if the area is filled during the execution of a task or if the area is not large enough for the records produced during a single LUW, or even for a single large log record, an overflow mechanism exists, whereby records are written to a temporary storage destination, which may be in main or auxiliary storage. Note that much less dynamic logging is required than system logging, because the Dynamic Transaction Backout mechanism can rely on the contents of main storage tables, whereas the emergency restart mechanism cannot, and no information need be logged about the start or end of transaction. The contents of the Dynamic Log for a particular transaction, except some records needed for transaction restart, are cleared at the end of each LUW.

Note: When running DL/I (IMS/VS DB or DL/I DOS/VS) under CICS/VS, all DL/I logging is directed to the CICS/VS System and Dynamic logs. The normal DL/I logs (as used in batch DL/I processing) are not used.

Deferred Work

Certain types of CICS/VS activity are liable to become involved with a large amount of logging. Examples include the purging of transient data queues or temporary storage destinations. To avoid the logging of a large number of records as they are removed from, say, a transient data queue, deferred work is a principle employed whereby the records are not actually removed from the queue and logged but are only flagged as having been removed. Only when the Logical Unit of Work has completed successfully will the records actually be removed from the queue.

SYSTEM ACTIVITY KEYPOINTS

The purpose of system activity keypoints is to indicate to CICS/VS, during an emergency restart, the user tasks active at the time of the keypoint, the status of transient data intrapartition destinations, temporary storage data identifications, and terminal control table status for VTAM-supported terminals. This information is available on the System Log anyway. The activity keypoint is simply a means of reducing the amount of the System Log that must be scanned during emergency restart time. Without such a mechanism, the System Log would have to be scanned completely to ensure that all "in-flight" tasks had been found.

A system activity keypoint is written periodically by CICS/VS during normal operation, to record on the system log data set the processing activity status of CICS/VS and user tasks at the time of the keypoint. The frequency of recording system activity keypoints is a function of the number of output operations to the system log. This frequency may be specified by the user either at CICS/VS system generation or at system initialization, and may also be changed dynamically during online operation through the use of the master terminal operator transaction (CSMT or CEMT). The frequency of the activity keypoint, and the amount of journaling performed by active tasks, determines the amount of data on the system log to be processed when CICS/VS is restarted. The amount of this data will influence the duration of the CICS/VS emergency restart.

The information recorded by the system activity keypoint comprises the following CICS/VS tables and control blocks:

TCAs
: status of tasks that have at least logged one record or have completed a LUW by issuing a sync point

DCT
: status of transient data intrapartition destinations

TSUT
: status of temporary storage unit table entries

TCT
: identification of terminals waiting for response to committed output messages (see later)

The activity keypoint function is initiated by the transaction CSKP, which is attached periodically by the journal control program (JCP). This transaction transmits system data to the system log and then issues a conditional link to a user activity keypoint program, DFHUAKP. The purpose of this facility is to allow the user to insert application-dependent information in the system activity keypoint. In order to

operate properly, the program must be resident and should use only storage control and journal control functions. Journal operations should be asynchronous without start I/O. Synchronization is provided by an end-of-keypoint synchronous record, written by the activity keypoint program at the end of keypoint processing. The user should ensure that the keypoint transaction has higher priority over other tasks that are eligible to log data.

During an emergency restart, the recovery utility program (RUP) copies to the restart data set only that data which has been output by in-flight tasks (tasks that were still active at system termination time). In order to force RUP to copy user activity keypoint data to the restart data set, the user must provide a journal record identifier with the high-order bit ON in the record ID field of the user keypoint data record. Refer to the CICS/VS System Programmer's Reference Manual for more detail. The user keypoint should be used to keypoint only limited amounts of data, for example, selected user data or tables.

Note: An activity keypoint must not be confused with a warm keypoint.

An activity keypoint is taken periodically throughout CICS/VS execution after a user-specified number of log records have been written on the System Log. It records the status of various pieces of system information necessary for protected resource backout so that the scan of the System Log, should an emergency restart be necessary, can be shortened. An activity keypoint is written on the System Log.

The warm keypoint is taken at system termination to record the status of the quiesced CICS/VS system so that it can be reinstated at a subsequent warm restart. It is written on the Restart Data Set. For more details, refer to the description of warm restart later in this chapter.)

PROTECTED MESSAGES (VTAM ONLY)

Some VTAM-supported terminals, such as the 3600, enable message recovery and resynchronization to be attempted during emergency restart of CICS/VS. The user can specify in the Program Control Table that various transaction codes are to be "protected." The term "protected" applies to the task with respect to terminal messages. All other protection is on a resource rather than a transaction basis.

Message Recovery and Resynchronization

If message-protected transaction codes are used with VTAM-supported terminals, the first input message during a logical unit of work associated with a task is logged to the system log. If uncontrolled shutdown occurs, the input message for each in-flight LUW is identified during emergency restart and transferred to temporary storage. It can be retrieved from temporary storage by user programs based on the identification of the terminal that entered the message. The in-flight activity associated with each task is backed out during emergency restart. The user may then initiate reprocessing of that input message.

The last output message transmitted by message-protected tasks without a WAIT are regarded as "committed output" messages. The output message is logged and the message is transmitted by VTAM, together with a request for a positive response from the VTAM terminal on receipt of the message. When the positive response is received by VTAM, it

notifies CICS/VS, which logs the receipt of the positive response to the system log.

If an uncontrolled shutdown occurs before the positive response is logged by CICS/VS, it can be detected by the CICS/VS recovery utility program during emergency restart. The output message is transferred to temporary storage, from which it can be retrieved by user programs based on the identification of the terminal designated to receive the output. The task that generated the output message may have terminated normally prior to uncontrolled shutdown, but the terminal may not have received that committed output message. On emergency restart, CICS/VS retransmits this output message with a request for positive response, to ensure its receipt by the terminal.

CICS/VS uses the VTAM sequence numbers, which are allocated to each input and output message associated with a logical unit, to establish message resynchronization with programmable controllers during emergency restart. These are obtained from the system log by the CICS/VS recovery utility program.

Inquiry transactions which do not modify data sets should not be specified in the Program Control Table as protected. Logging of input or output messages will not occur. Such transactions can be reprocessed on emergency restart, if necessary.

Deferred Output Integrity

An uncontrolled shutdown or transaction abend may occur during the processing of an LUW belonging to a protected task. Such an in-flight LUW will be backed out on emergency restart or during abend processing by CICS/VS. If a committed output message is transmitted to a terminal when first requested by the application program, it may reach the terminal even if an uncontrolled shutdown or transaction abend occurs before the protected task terminates normally. The contents of the output message may indicate to the terminal that processing of the task completed normally, because it was sent before the task really completed. However, if the task was in-flight, it will be backed out on emergency restart. The task must then be reprocessed following emergency restart or abend processing. The terminal operator is not aware that the task was in-flight and was backed out, and would not normally reenter it on emergency restart. The result is the loss of that task's processing.

To avoid this possibility, CICS/VS defers transmission of the last output message until completion of the task's current logical unit of work. Thus, the output will be only transmitted if the LUW completes normally. In the event of uncontrolled shutdown or transaction abend before completion of the LUW, the in-flight LUW is backed out on emergency restart or during abend processing, and the remote programmable controller (or the terminal operator) can retrieve the original input message from temporary storage and resubmit that input message for reprocessing. If the LUW completed and output was initiated, but uncontrolled shutdown or transaction abend occurred before a positive response was received from the terminal, the completed LUW is not backed out on emergency restart or during abend processing. The committed output message is retransmitted by CICS/VS on emergency restart as previously described.

The result of this deferred output is improved message integrity. The trade-off is a delay before transmission of the output with a possible increase in response time at the terminal. If response time is

not to be increased, the user should either request a user sync point, or terminate the task as soon as possible after requesting output.

Terminal control and BMS both permit the user to request immediate initiation of terminal I/O for VTAM-supported terminals. This avoids the deferred output delay, but the possibility is increased that the terminal operator may receive output indicating completion of tasks which are subsequently backed out.

An alternative approach is to break the output into two or more sections. The program can request output of the first section, specifying either immediate output or the default of delayed output. When output of the next section is requested, the first section is transmitted if it indicated delayed output. This continues until either a sync point is reached, or the task terminates when the last section of output is considered a committed output message and is handled as previously described. The partial sections of output may satisfy the requirement for rapid response time, with each section indicating that further output is following. Only receipt of the last section of output is an indication to the terminal operator of completed task execution. The disadvantage of this approach, however, may be increased CICS/VS and VTAM overheads.

The use of BMS terminal paging introduces another consideration. If the terminal that initiated a protected task is in transaction and request page status, terminal pages are written to temporary storage for subsequent display, but are not directed to the terminal. The application program should send a committed output message to the terminal, indicating completion of the task and availability of the terminal pages, through the terminal paging commands (see the CICS/VS Operator's Guide.) If the terminal is in TRANSCEIVE status, the first page will be transmitted to it automatically when the protected task completes.

Chapter 5.2. Error Handling

CICS/VS detects any of many possible errors during its processing of particular CICS/VS requests by application programs. Full descriptions of all possible errors arising from specific requests are given in both macro- and command-level versions of the CICS/VS Application Programmer's Reference Manual. It is good practice to ensure that all requests for CICS/VS services from application programs include tests and appropriate action for any possible errors.

Recovery from Terminal I/O Errors

Through the teleprocessing access methods, CICS/VS detects any of many possible errors arising from faults in lines and terminals. The CICS/VS Terminal Abnormal Condition Program and Node Abnormal Condition Program are invoked to perform standard error-recovery action for BTAM and VTAM terminals respectively. These programs in turn invoke user-written Terminal and Node Error Programs (DFHTEP and DFHZNEP) respectively to allow user-defined error recovery actions to be performed.

The writing of Terminal Error and Node Error Programs, the functions that may be performed in them, and their use are described in Part 3. Techniques for dynamic terminal reconfiguration (by user programming) following an unrecoverable terminal I/O error are also described. Techniques are discussed for use with BTAM-supported terminals and VTAM-supported programmable controllers.

Session failures and terminal I/O errors can cause transactions to abend. The dynamic transaction backout and transaction restart facilities, which are discussed later in this chapter, may be used to recover from such abnormal terminations.

Program Check Handling

CICS/VS supplies a system recovery program (SRP), which provides logic for recovery from program-check interrupts. In addition, the system recovery program handles partition/region ABEND recovery implemented within the system recovery table or within user routines. This latter function of the system recovery program is described under the heading "Operating System Region/Partition Abend Handling" later in this chapter.

SYSTEM RECOVERY PROGRAM

The system recovery progam is invoked during initialization to issue the OS/VS SPIE or VSE STXIT PC macro to establish the address of a routine in SRP as the routine to be invoked by the operating system in the event of a program check occurring in any part of CICS/VS or the application programs running under its control.

If a program check occurs at any time during the execution of CICS/VS, SRP receives control from the operating system at the point specified in the OS/VS SPIE or DOS/VS STXIT PC macro. The handling of

the program check depends on which particular part of CICS/VS or user application programs caused the interrupt to be raised. The following situations are identified by the system recovery program:

- Program check during the execution of the Storage Control Program: SRP activates storage control recovery.
- Program check in application program or whilst performing a CICS/VS service for a particular application program: SRP abnormally terminates the task.
- Program check in CICS/VS system module whilst performing a CICS/VS function not related to a particular application program: SRP abnormally terminates the partition/region.

Program Check In Storage Control

If the program check occurred whilst the storage control program was executing, SRP passes control to the storage control recovery program (SCR). (Generally, such a program check occurs if a storage accounting area (SAA) or storage chain pointers have been destroyed by prior incorrect execution of an application program.) The storage control recovery program attempts to recover the destroyed information using either a duplicate SAA appended to each allocated storage area, or by using forward and backward chain pointers connecting free area queue elements (FAQE). If recovery is unsuccessful, the region or partition is abended. If recovery is successful, the storage violation is noted in the CICS/VS statistics and task execution continues.

In addition, if the user has so specified, the Formatted Dump Program will be invoked, before any repairs are attempted by the storage control recovery program, to dump all or part of the partition, depending on the options specified for the formatted dump program. The user's choice of dumps to be taken is specified by the SVD operand of the DFHSIT macro (refer to the CICS/VS System Programmer's Reference Manual). If it is subsequently found that no satisfactory correction is possible or the violation cannot be identified, the region or partition will be abended but there will be no second dump. To use this dump facility, the system programmer must generate the DFHSCP macro with the RECOVER=YES operand, and set the FDP option to SNAP or FORMAT, as well as specifying the SVD operand.

Program Check Under a User Application Program Task

If the program check occurred in a CICS/VS application program or in a CICS/VS module whilst performing a service for a specific task (that is, running as part of the application program's task), the task is abnormally terminated with a program control ABEND macro instruction. This activates program-level ABEND exit routines associated with the task, as described under "Transaction Abend Handling", below.

Program Check Under a CICS/VS System Task

If the program check occurred while a CICS/VS system task is executing, the system recovery program issues a VSE DUMP or OS/VS user ABEND macro instruction to abnormally terminate the entire CICS/VS partition/region. This causes the partition/region abend handling mechanism to be invoked (see "Operating System Region/Partition Abend Handling" later in this chapter). CICS/VS system tasks are:

- Task Control (whilst performing task dispatching)
- Terminal Control
- Journal output tasks (one per user journal)

All other CICS/VS services are performed whilst running under the application program's task.

Transaction Abend Handling

This section describes the levels of recovery function provided by CICS/VS to handle transaction abends. Corresponding actions when the whole CICS/VS system abends are described under the heading "Operating System Region/Partition Abend Handling" later in this chapter.

The functions described here are presented in the order in which they are invoked at transaction-abend time, namely:

- User Exit Routines
- Dynamic Transaction Backout
- Abnormal Condition Program
- Program Error Program
- Transaction Restart
- PCT/PPT Disable and Enable
- Transaction Dumps

Program Control Abend Requests

Program control ABEND requests, issued either by CICS/VS application programs or by CICS/VS system modules (as, for example, in the case of a program check as described above), are intercepted by the program control program. Control may be passed to program-level ABEND exit routines (see Figure 5.2-1) specified by each separate program level reached as the result of a program control LINK request. These ABEND exit routines may:

- Attempt recovery and retry of the condition that caused the ABEND to be requested
- Record application-dependent information for later recovery and allow the ABEND to continue
- Choose to ignore the ABEND and specify that normal execution is to continue

Control is then passed to the next higher program level whose relevant ABEND exit routine is given control if the ABEND is allowed to continue. If the ABEND is ignored at the lower ABEND exit, control is returned to the statement following the LINK request that originally activated the lower level program.

Program Level Abend Exit Routine

Program-level ABEND exits are supported by CICS/VS, so that user-written routines can remove the effects of incorrectly executing tasks. The exit is activated and deactivated by a CICS/VS HANDLE ABEND command coded in an application program (see "HANDLE ABEND Program Processing" below). The exit routine may exist either as a separate program, or as a routine within the program issuing the command.

Once a program control ABEND occurs in a task and an abend exit routine has been entered, any of the following three ways can be used to terminate the exit routine processing:

1. Issue a RETURN command to continue processing this task as if the ABEND had not occurred. In this case, control is passed to the program on the next higher logical level (at the statement following the LINK request) or, if the program in control at the time of the ABEND was at the highest level, the task is normally terminated by CICS/VS. (See A in Figure 5.2-1.)
2. Issue an ABEND command to continue with ABEND processing. This may indicate execution of a specified exit routine for a program on a higher logical level, or at the highest level may cause a LINK to the CICS/VS abnormal condition program to complete the abnormal termination. (See B in Figure 5.2-1.)
3. Branch to a point in the program that was in control at the time of the ABEND, and attempt to retry the operation. (See C in Figure 5.2-1.)

The CICS/VS Messages and Codes manual contains a list of transaction ABEND codes for the abnormal terminations initiated by CICS/VS.

HANDLE ABEND Programming Processing: In order to activate the exit for a particular task, the application program may issue the HANDLE ABEND command at each program level reached by a LINK request. This identifies either the name of a separate program or of a routine within the abnormally terminated program, to which control is to be passed if an ABEND occurs while that program level is in control. If the program level, after further processing, wishes to cancel the exit, it issues a HANDLE ABEND command without specifying a program or routine name.

The HANDLE ABEND command can be issued by any assembler language, American National Standard (ANS) COBOL, or PL/I program. The ability to pass control to a specified label or program on a program ABEND allows it to be implemented in a manner similar to that provided by PL/I ON-conditions.

In program-level ABEND exit routines defined for a task, the user may wish to record application-dependent information relating to that task prior to its abnormal termination. He may also attempt any local recovery that may be desired.

DYNAMIC TRANSACTION BACKOUT

If no user abend routine was specified, or if the routine determined that the ABEND is to continue, CICS/VS will invoke Dynamic Transaction Backout (DTB) as part of the abnormal task termination process.

DTB is the backing-out of the effects of a transaction that has abnormally terminated. The resources that have been specified as protected are restored to the state they would have been in had the transaction terminated at its last user-defined synchronization point, or had not run at all. The resouces are thus restored to a well-defined self-consistent state. If the task was initiated by interval control, dynamic transaction backout will call interval control programs to re-initiate the task.

DTB is similar in effect to the backout of "in-flight" tasks carried out during emergency restart following a CICS/VS failure. The most important difference is that DTB operates on a single abnormally terminating transaction and is carried out online, while the rest of the CICS/VS system continues to run normally. DTB thus provides immediate recovery of data base integrity following a transaction failure.

DTB will backout the changes made to the following CICS/VS resources by the transaction that abnormally terminates:

- CICS/VS files
- DL/I data bases
- Transient data (intrapartition only)
- Auxiliary temporary storage
- Terminal messages (VTAM only)
- TCT user area
- Command level communications area (COMMAREA)

- TIOA

  Note: For restartable transactions (those with RESTART=YES specified in DFHPCT TYPE=ENTRY), the last three areas can be restored to the status they had at the beginning of the task.

The CICS/VS features used by DTB, many of which are also used for emergency restart, include automatic logging, deferring work to the end of the logical unit of work, the saving of pertinent information, and enqueueing on recoverable resources. These features are described under the heading "Principles Underlying CICS/VS Recoverable Resources" earlier in this chapter.

To restore the resources to the state they were in at the beginning of the logical unit of work, a description of their state at that time must be preserved. For the tables that are maintained by CICS/VS (the destination control table and the temporary storage unit table), the information is held in the tables themselves. For transient data and auxiliary storage temporary storage, records that have been deleted or the "before" images of records that have been changed are saved on the transient data or temporary storage data sets themselves. For DL/I or CICS/VS files, the "before" images of deleted or changed records are recorded on a dynamic log. The first input messages from protected VTAM terminals are also held on this log.

The dynamic log, which resides in a main storage buffer, is allocated (one per transaction) when, and only when, required. The size of the storage to be allocated to the dynamic log is specified in the DBUFSZ operand of the DFHSIT macro, and has a default value of 500 bytes. If the main storage buffer is too small to hold all the records (or even a single large record), temporary storage will automatically be used instead.

Resource Recovery

As stated, the changes made to certain CICS/VS resources by the terminating transaction will be backed out by DTB to either the last user-defined sync point or the start of the transaction. The following descriptions outline what is involved in the recovery of these resources.

CICS/VS Files and DL/I Data Bases: If these resources are defined as recoverable in the File Control Table (as for emergency restart), any changes to them will be recorded on the dynamic log. During DTB, the changes since the last sync point are recovered from the log and are reversed. A user exit is provided for error situations (see following heading "User Exits in the Dynamic Transaction Backout Program"). Dynamic Transaction Backout must not be invoked for VSAM files that are in load mode. A dummy record must be added to empty and close files before adding any new records. Note that all DL/I data bases are recoverable.

Transient Data: Intrapartition queues that have been specified as logically recoverable will be restored by DTB to their status at the last sync point. This means that any records that have been fetched will be restored to the front of the queue, and the space occupied by records added since the sync point will be freed. The information in the Destination Control Table will be correspondingly backdated.

The initiation of any transactions because the queue has reached its trigger level since the last sync point will be suppressed.

Physical recovery, as can be specified for emergency restart, is not meaningful for DTB and is ignored.

These recovery procedures apply only to intrapartition queues; recovery of extrapartition queues is not supported.

Temporary Storage: Auxiliary storage destinations, which are defined in the Temporary Storage Table as recoverable (as for emergency restart), are restored to their status at the last sync point. This means that released or purged records are recovered, replaced records are back-dated to their former contents, and added records are deleted. The Temporary Storage Unit Table and the byte map reflecting the control interval status are also restored.

Terminal Messages (VTAM Only): If the transaction is specified as message-protected in the Program Control Table and is used with a VTAM-supported terminal, on abnormal termination of the transaction a Logical Unit Status (LUS) command and sense bytes, indicating whether DTB completed successfully or not, are sent to those terminals that support LUS commands. An ABEND message with the DTB status is sent to those terminals that do not support LUS.

The transmission of any deferred output, which would normally occur at transaction termination, is suppressed by DTB. This is because committed output messages must not be sent if backout has occurred.

The first input message since the last sync point is recovered from the dynamic log and presented to a user exit.

Note that the retransmission of "in-doubt" committed output messages as carried out during emergency restart is not required in DTB. This is because the system is continuously active, so the positive response cannot have been lost.

User Exits in the Dynamic Transaction Backout Program (DFHDBP)

The Dynamic Transaction Backout Program has four user exits, which the system programmer may code. Full details for coding these user exits can be found in the CICS/VS System Programmer's Reference Manual; a brief description of each one follows.

1. XDBINIT - this exit is given control on entry to DFHDBP.
2. XDBIN - this exit is given control when each dynamic log record other than DL/I is obtained.
3. XDBFERR - this exit is given control when some error condition has been returned from the File Control Program during the backout processing or if an error has been detected by DFHDBP itself.
4. XDBDERR - when the DL/I backout routine detects an error, its error message is routed to the operator's console and this exit is then given control.

Dynamic Transaction Backout Generation

In order to generate DTB in his CICS/VS system, the system programmer must specify. . .

1. . . that DTB is to be part of the system generated by DFHSG. This is done via the DFHSG PROGRAM=DBP macro instruction, and the DTB operand of the DFHSG PROGRAM=JCP macro.

2. . . whether DTB is to be included in this execution of the CICS/VS system, and if so, which module is to be used. This is done via the DBP and DBUFSZ operands of the DFHSIT macro.

3. . . that an entry for the dynamic transaction backout (DFHDBP) will be required in the Program Processing Table if DTB is to be used. This is done via the DFHPPT TYPE=ENTRY macro.

4. . . whether or not a transaction may be a candidate for backout. This is done via the DTB operand of the DFHPCT TYPE=ENTRY macro.

Full details of all the macro instructions mentioned can be found in the CICS/VS System Programmer's Reference Manual.

A successful execution of DFHDBP depends on how the system is generated and how it is initialized. The File Control Program (DFHFCP) should be generated to support all needed functions (for example, VSAM DELETE if VSAM additions are to be backed out). The File Control Table should also be generated to allow "reverse" operations on data sets.

The Temporary Storage queues used by DTB should not be specified as recoverable in the Temporary Storage Table.

DTB Performance Considerations

Because DTB requires resources to perform successfully, it is recommended that, where possible, transactions are not specified as both stall purgeable and as candidates for DTB. If a transaction that is a DTB candidate is stall purgeable, backout will be attempted, but may not complete successfully because of a lack of resources.

Statistics are available through the normal mechanisms indicating:

- The number of times a dynamic log buffer is used.
- The number of times the buffer or a large dynamic log record was spilled to temporary storage.

These statistics can be used by the system programmer to assist in selecting the optimum buffer size.

ABNORMAL CONDITION PROGRAM

The purpose of the Abnormal Condition Program (ACP) is to perform "tidying-up" actions. The principal action is to determine whether it is possible to send a message to the terminal connected to the backed-out transaction and, if so, to send the abend messages that may be necessary. In addition, ACP is responsible for linking to the user's Program Error Program (PEP).

PROGRAM ERROR PROGRAM

A program error program (PEP) capability is provided by CICS/VS, to offer an opportunity for a user-written program error program to carry out installation-level action following a transaction abend. Such action may involve the recording of application-dependent information, for utilization by user-programs when CICS/VS is reinitialized.

The PEP is given control during the processing of abnormal task termination through a LINK from the CICS/VS abnormal condition program (ACP) after all program-level ABEND exit routines have been executed by the task that abnormally terminates and after Dynamic Transaction Backout has been performed, if required. Included in the data passed to the PEP are the PCT and PPT entry addresses for the transaction code that initiated the program, and the ABEND code. The PEP can decide that CICS/VS is to mark the PCT and/or PPT entry as disabled (inaccessible) when control is returned to the ACP. The user may perform any additional functions he desires. The ACP will write a message indicating abnormal task termination to the master terminal destination, and indicate if the PCT and/or PPT entries have been put out of service (disabled). Any further information to be passed to the master terminal operator may be written by the user-developed PEP.

The PEP is given control on any program control ABEND requested by a user or system module, with the exception of a forced ABEND, in an effort to alleviate a stall situation. In this case, if the LINK to PEP would be suspended because of a shortage of storage, the LINK does not take place and no action is taken against the PCT entry. A message is sent to the master terminal destination stating that the PEP was not executed, so that the master terminal operator can disable the PCT and/or PPT entries, if desired.

TRANSACTION RESTART

Transaction Restart is a facility that can restart a transaction that has abnormally terminated and (when the logic of the application program permits) will do so without intervention by or effect on the terminal operator. Transaction Restart requires the Dynamic Transaction Backout to be available, and can be selected or suppressed dynamically by means of a user exit program. Transaction Restart is particularly useful in handling interlocks caused by Program Isolation, details of which can be found in Chapter 2.3 under "Simultaneous Update Protection".

To use Transaction Restart, the system programmer must:

- Generate Dynamic Transaction Backout support, details of which can be found under "Dynamic Transaction Backout Generation" earlier in this chapter.
- Specify RESTART=YES and DTB=YES in the DFHPCT TYPE=ENTRY macro for the transactions that are to be restarted.
- Check the logic of those transactions for any additional resources that need to be made recoverable. Note that if a START FROM command is used to create a restartable task, the initial data should be protected by, for example, a DFHTST TYPE=RECOVERY,DATAID=DF macro.

- See if any restartable application programs could give false results after a restart. Transactions that execute as a single logical unit of work or which execute a loop (on each pass reading one record from a recoverable destination, updating other recoverable resources, and closing with a sync point) are safe. On the other hand, a transaction in which the first and subsequent logical units of work change different resources, or in which the contents of the input data area were used in several logical units of work, would need modification to avoid erroneously repeating work done in logical units of work that precede the abend. The transaction could be modified to refer to a "restart" flag or to save a suitable indicator of its progress in non-recoverable temporary storage.

- Write a Retry Program (DFHRTY) and include an entry for it in the PPT, unless the requirements of the installation are met by restarting only those tasks that satisfy a default rule. The installation's Retry Program may be based on the CICS/VS-supplied version and can examine (in addition to the resources that are backed out) the TCA (both the user part and the system part), the TWA, and the task's application storage in order to identify the transaction, the cause of the failure, and the state of progress. The program can then indicate whether the transaction is to be restarted, and can perform any other operations (such as notifying the terminal operator and creating a progress-indicator for the restarted application program) that are necessary in the particular circumstances. The criteria for restarting a task should be made tight enough to prevent a loop through a recurring abend.

The CICS/VS System Programmer's Reference Manual contains further information on the Transaction Restart facility.

A transaction can only be restarted after abend processing (including the execution of HANDLE ABEND code) has been done and after dynamic transaction backout has completed without error. Whether restart can take place depends on certain criteria:

- If the abend was due to program isolation and occurred in the first logical unit of work, and if no terminal activity had occurred beyond reading the initial terminal input, the "default rule" is to allow a restart to take place.

- The retry program (DFHRTY) will be invoked if it exists, and may override the "default rule" to cause restart in other circumstances.

The initial program is invoked as at its initiation, with the following information available to it:

- The task's TCA.

- The associated terminal, if any.

- All recoverable resources (files, data bases, temporary storage (including interval control data) and transient data) in their state at the previous sync point or at initiation. Non-recoverable items will be unchanged by the process.

- The initial task input/output area, if any, all other task input/output areas having been freed.

- An indicator (the "restart flag"), which is able to be read by the ASSIGN RESTART command, that this activation results from a restart.

- The TCTUA and COMMAREA with their contents as at the start of the task.
- The transaction work area after it has been cleared, all other transaction storage being freed.

Further points on Transaction Restart are:

- Statistics on the total number of restarts against each transaction are kept.
- When an abend is followed by a restart, system messages are not issued and the program error program (DFHPEP) is not invoked. When a restartable transaction abends and is for any reason not restarted, a message will be issued giving the reason and DFHPEP will be invoked in the usual way.
- Emergency restart, which occurs after the CICS/VS partition has failed does not restart any tasks.
- Making a transaction restartable involves slightly more overheads than dynamic transaction backout because more items are logged.
- In some cases, the benefits of Transaction Restart can be obtained instead by using the SYNCPOINT ROLLBACK command. Using the latter method, all the executable code is kept in the application programs, not in the system programmer's area.

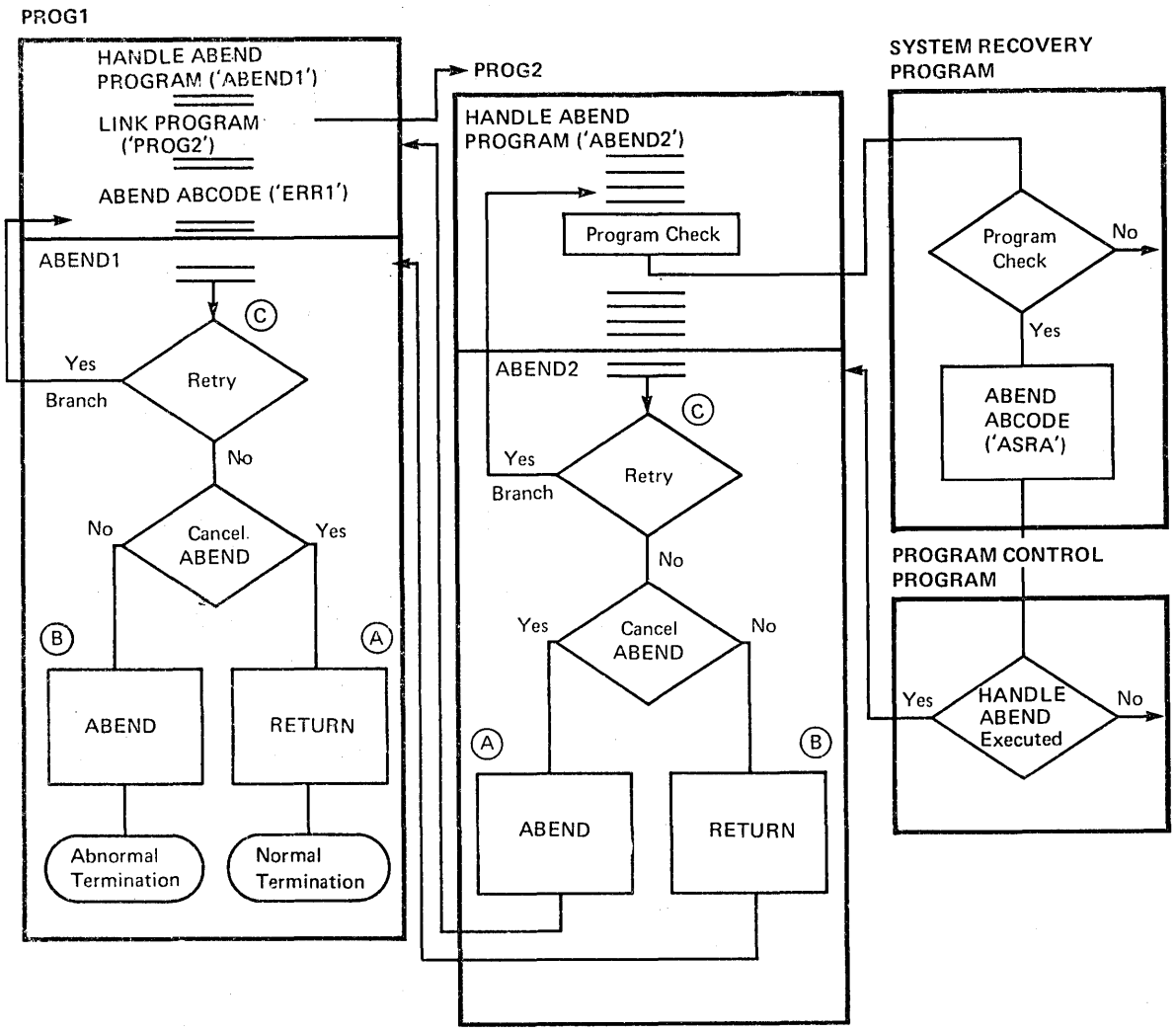


Figure 5.2-1. Program-Level ABEND Exit Processing

PCT DISABLE AND ENABLE

CICS/VS provides support for the disabling of transaction codes following their abnormal termination, and their subsequent enabling when the particular problem has been rectified.

If an application program abnormally terminates (perhaps because of a program check or an ABEND command), the user, in an ABEND routine or in the PEP, can flag the appropriate transaction code entry in the PCT as disabled. Any further attempt by terminals or programs to use that transaction code will be rejected by CICS/VS until the transaction code is enabled again. Consequently, the effect of program checks can be minimized, so that every use of the offending transaction code does not result in a program check. Only the first program check is processed, and, if the PEP indicates that the PCT entry is to be disabled,

subsequent users of that transaction code will not be accepted by CICS/VS.

Following correction of the error, the relevant PCT entry for the transaction code can be enabled by the master terminal operator, to allow terminals to use it. The master terminal operator can also disable transaction codes when transactions are not to be accepted for application-dependent reasons, and enable them again at a later time. Further information on the master terminal operator functions can be found in the CICS/VS Operator's Guide.

TRANSACTION DUMPS

Dump Data Set

If a program check or program control ABEND occurs, a program dump of all the storage associated with the task in error is automatically produced by CICS/VS. This includes any application programs that have been linked to, or any terminal I/O areas, file I/O areas, file work areas, and other working storage used by that task.

These program dumps are directed to one of two dump data sets. Relevant CICS/VS areas such as the CSA (common system area) and TCA (task control area), together with the associated terminal control table entry for that task, are also dumped.

Two dump data sets are used by CICS/VS, but only one is in active use at a time. Any program dump, resulting from abnormal termination or use of the CICS/VS dump control commands issued by application programs, is directed to the active dump data set. The master terminal operator can indicate that the second dump data set is to be made the active dump data set, and the first dump data set is to be left inactive. This switching of dump data sets is accomplished to enable dumps previously written to a dump data set to be printed from that data set while CICS/VS execution is in progress. The printing of dumps is achieved through the CICS/VS dump utility program, which may be executed in a batch partition concurrently with CICS/VS. (For further information on dump data sets, see the appropriate CICS/VS System Programmer's Guide for VSE or OS/VS.)

Operating System Region/Partition ABEND Handling

During System Initialization, the System Recovery Program (SRP) establishes a recovery function for CICS/DOS/VS (STXIT AB) or CICS/OS/VS (STAE/ESTAE). This recovery function enables a program to recover and continue from a system abnormal termination condition. The CICS/VS support for the function will trap abends and check against a user-supplied list of error codes. If the appropriate error code is listed, a routine will be invoked to recover. If recovery is possible, the transaction, rather than the CICS/VS system, will be abended.

To make use of this selective transaction abend feature, the user must generate a CICS/VS System Recovery Table (SRT). The SRT enables users to list ABEND codes for which recovery is to be or is not to be attempted. Users can supply their own recovery code or use the recovery routine supplied by CICS/VS.

Upon detecting a partition/region ABEND situation, the SRP gains control and scans the SRT to determine whether the ABEND code passed to it is in the table. If an entry for the ABEND code specified is found in the table, the associated recovery routine is invoked.

If the ABEND processing routine is resident in the SRT, control is given to that routine. After processing is complete, control can be returned to the SRP specifying whether the partition/region is to be abnormally terminated, or whether it can continue processing. Since an abnormal termination can be recovered from, CICS/VS allows the user to avoid terminating the entire partition or region.

If a non-resident ABEND processing routine is a program, control is given to that program through a program control LINK. Upon return to the SRP, the same options may be specified as in resident processing.

If no ABEND code is present in the SRT (or if recovery is unsuccessful), the task and the CICS/VS partition/region are abnormally terminated. During such task termination, any program-level ABEND exits for the task are processed. Following this, the user's installation-level program error program is given control, and a controlled shutdown is attempted.

Because a partition/region abend may occur at any time, a number of tasks may be in-flight at the time of the abend. Although the SRP attempts a controlled shutdown by recording information for a subsequent warm start, the user may wish to do an emergency restart to back out the processing carried out by the in-flight tasks.

A similar situation may occur if the master terminal operator attempts a controlled shutdown and specifies immediate termination of CICS/VS. Tasks which were in-flight at immediate termination may need to be backed out by an emergency restart.

Chapter 5.3. CICS/VS Shut-Down and Start-Up

CICS/VS Termination

The shutdown of CICS/VS may be either a controlled shutdown, an immediate shutdown, or an uncontrolled shutdown (such as following a machine check or power failure).

CONTROLLED SHUTDOWN

A controlled shutdown, resulting from CSMT SHU,NO terminal operator request, preserves certain vital information about the CICS/VS environment during normal or abnormal termination of CICS/VS. This information, recorded on the restart data set, may be subsequently used to warm start CICS/VS and reinitialize it to its status at termination. The user may, optionally, elect to warm start only certain parts of CICS/VS, and allow a cold start (complete reinitialization) of other parts of CICS/VS.

Two CICS/VS tables are used to facilitate the functions of controlled shutdown and warm start:

- Transaction list table (XLT)
- Program list table (PLT)

Transaction List Table (XLT)

On CICS/VS controlled shutdown, a transaction list table (XLT) may be loaded. This table identifies a list of transaction codes accepted by CICS/VS during termination. All other transaction codes will be rejected by CICS/VS.

Program List Table (PLT)

The program list table (PLT) is a list of programs to be executed either during controlled shutdown or during system initialization. It is generated by the user, who generally specifies two tables. One PLT identifies user-written programs that are to be executed during either the first or second stage of CICS/VS controlled shutdown (see below). These user programs may record application-dependent information, which will permit user recovery of that information on subsequent system initialization.

Another PLT can be used to identify user-written programs that are to be executed during the post-initialization phase of CICS/VS system initialization. These user-programs may locate the application-dependent information written by programs identified in the PLT during CICS/VS controlled shutdown, and use that information to reestablish the

online applications as required by the user. Several other uses of the PLT are described later in this chapter.

A normal controlled shutdown causes the status of various areas to be written to the restart data set to permit a warm start to take place. It uses the program list table (PLT), as described above, and carries out termination in two stages: the "first quiesce stage" and the "second quiesce stage." During the first quiesce stage, terminals are still active, but they are only permitted to enter transactions defined in the transaction list table (XLT). Programs defined in the first section of the PLT are also executed. During the second quiesce stage, terminals are deactivated and programs defined in the second section of the PLT are executed. The following paragraphs further describe the purpose of these two stages of termination.

- FIRST QUIESCE STAGE OF TERMINATION

  During the first quiesce stage of termination, a group of user-written programs may be sequentially executed. These programs perform special operations that are unique to the installation. All CICS/VS facilities are available to the programs during this stage. The programs to be linked to are defined in the program list table that is loaded during system termination.

  In addition, only those transactions defined in the transaction list table are accepted from terminals. Existing tasks, tasks to be automatically initiated, or ATP batches in process are allowed to continue unhampered to their normal conclusion (see Figure 5.3-1).

- SECOND QUIESCE STAGE OF TERMINATION

  At a user-defined point, termination activity waits until all system activity stops. (This defined point is the delimiter between the two parts of the shutdown PLT.) Termination then continues in the second quiesce stage without accepting any further terminal transactions.

  When all program list table programs defined to execute in the second quiesce stage have been executed, the warm keypoint is taken and CICS/VS terminates further execution (see Figure 5.3-1).

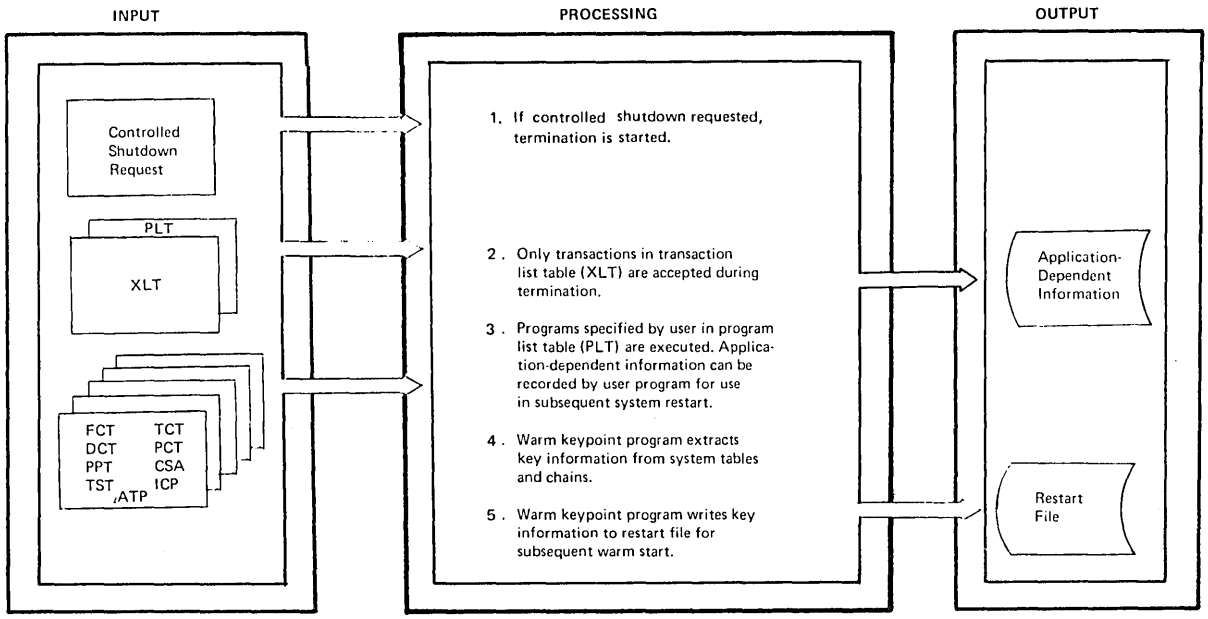


Figure 5.3-1. CICS/VS Controlled Shutdown

System Warm Keypoints

A system warm keypoint is written by the CICS/VS-provided keypoint program as part of controlled shutdown, when all system activity has been quiesced. It is used during a warm start of CICS/VS to restore the operating environment following a controlled shutdown. The following information is recorded as part of a warm keypoint:

- FCT

  File status (disabled, enabled, closed, read-only)

- TCT

  Terminal and line status and negative poll delay (for nonswitched terminals)

- DCT

  Intrapartition destination status

- TST

  Temporary storage control blocks and data set bit map

- ATP

  Asynchronous transaction processing control blocks

- ICP

  Interval control outstanding requests

- PPT/PCT

  Disabled entries

- CSA

  Information in the common system area; for example, storage cushion size, ICV, ICVS, ICVR, and MAXTASK

Note: A warm keypoint must not be confused with an activity keypoint.

The warm keypoint is taken at system termination to record the status of the quiesced CICS/VS system so that it can be reinstated at a subsequent warm restart. It is written on the Restart Data Set.

An activity keypoint is taken periodically throughout CICS/VS execution after a user-specified number of log records have been written on the System Log. It records the status of pieces of system information necessary for protected resource backout so that the scan of the System Log, should an emergency restart be necessary, can be shortened. An activity keypoint is written on the System Log.

IMMEDIATE SHUTDOWN

An immediate shutdown of CICS/VS, due to a CSMT SHU,YES terminal operator request, results in the following:

- CICS/VS is not quiesced.
- PLT programs are not invoked.
- A warm keypoint is taken but since no quiesce has been performed this may be invalid if the system has protected resources. An emergency restart should be carried out if there are any protected resources in the system.

UNCONTROLLED SHUTDOWN

An uncontrolled shutdown of CICS/VS can result from four basically different causes:

- Power failure
- Machine check
- Operating system WAIT/ABEND
- Partition/region ABEND

In each case, termination of system operation is either immediate, or so soon after appearance of the cause that insufficient time, processor resources, or system facilties are available to permit CICS/VS to complete a controlled shutdown.

System or user tasks may still be active at the time of termination, because CICS/VS is unable to quiesce system activity. Consequently, a subsequent warm start of CICS/VS is impossible. Instead, an emergency restart must be carried out.

CICS/VS Initialization

CICS/VS can be initialized either:

- With a complete cold start
- With a complete warm start
- With a partial warm start
- With an emergency restart

COMPLETE COLD START

A complete cold start results in complete reinitialization of CICS/VS and system data sets to their status as specified at system generation, without regard for any previous system activity. The system initialization table (SIT) is used to specify the particular versions of the different CICS/VS system programs and tables that are to be used for CICS/VS initialization.

COMPLETE WARM START

A complete warm start reinitializes CICS/VS to the status that existed at the previous controlled shutdown -- all system activity having been quiesced normally prior to shutdown. All CICS/VS system programs and tables are first cold-started using the SIT as described previously. If the SIT indicates that a complete warm start is to be performed, all CICS/VS system tables are then reestablished to their status as at controlled shutdown, using the warm keypoint information written to the restart data set at that time.

PARTIAL WARM START

A partial warm start is similar to a complete warm start, except that only selected CICS/VS system tables are warm-started, as specified in the SIT. Information is obtained from the warm keypoint written at controlled shutdown, only for those tables specified to be warm started. The remaining tables are cold-started. An example requiring a partial warm start is an application which requires a warm start of the DCT so that data queued to intrapartition destinations prior to a controlled shutdown may be retrieved on restart. The FCT and TCT may also need to be warm-started to reestablish file and terminal status as at controlled shutdown. The application may, however, require a cold start of temporary storage. Figure 5.3-2 illustrates a CICS/VS warm start.

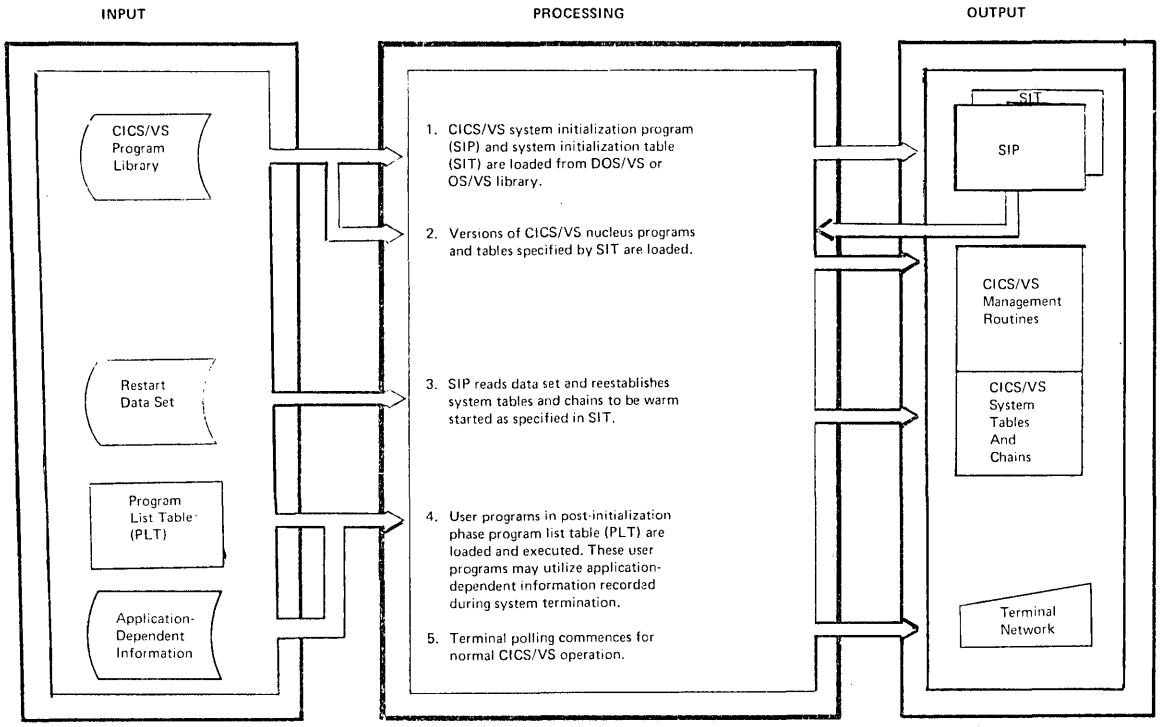


Figure 5.3-2. CICS/VS Warm Start Procedure

After all items have been initialized and control is about to be given to CICS/VS, the group of user-written programs specified in the program list table is sequentially executed. This is referred to as the "post-initialization" phase. These programs perform application-dependent functions, for the recovery of application-dependent information recorded by the user on termination, prior to complete restart of CICS/VS. All CICS/VS facilities are available except for direct terminal communication. Following post-initialization execution of programs in the program list table, the terminal control program is activated to enable terminal transactions to be received and processed.

EMERGENCY RESTART

An emergency restart restores certain CICS/VS facilities to a predefined point that existed prior to an uncontrolled shutdown. Information describing all changes, modifications, and updates made to system tables and to user data sets during previous CICS/VS execution was recorded on the system log data set. Another data set, the restart data set, is created during emergency restart. (Emergency restart is not supported by the CICS/DOS/VS Entry Level System.)

The system log contains all changes made to recoverable file control data sets, recoverable transient data intrapartition destinations, temporary storage protected destinations, and DL/I data bases. It also contains input and output messages for message-protected tasks executed by VTAM terminals.

CICS/VS emergency restart includes backout of DL/I data bases. It is not necessary to run the batch DL/I backout facility.

The restart data set is created during emergency restart, and contains system log activity and user journal records for those tasks whose processing activity had not reached a logical completion point when the uncontrolled shutdown occurred. (Such tasks are referred to as in-flight tasks in the following discussion of emergency restart.) This information can be utilized by the CICS/VS transaction backout program, for example, to remove the effect of data set modification by in-flight tasks.

On an emergency restart, the CICS/VS system initialization program (SIP) carries out a number of steps to restore CICS/VS operation to a predefined point prior to uncontrolled shutdown.

These steps are carried out by CICS/VS-provided routines. The user may wish to extend the functions carried out in emergency restart by the addition of user-written programs. To permit such user-written programs to be used, CICS/VS identifies in-flight tasks which may require user backout, and copies from the system log, in backward sequence, all system-logged records, and user journal records, for those in-flight tasks to the restart data set. To use this extension capability effectively, it is important that the system designer has an appreciation of the steps carried out be CICS/VS during emergency restart.

The remaining topics in this section provide an overview of the CICS/VS emergency restart procedure. This procedure is illustrated in Figure 5.3-3.

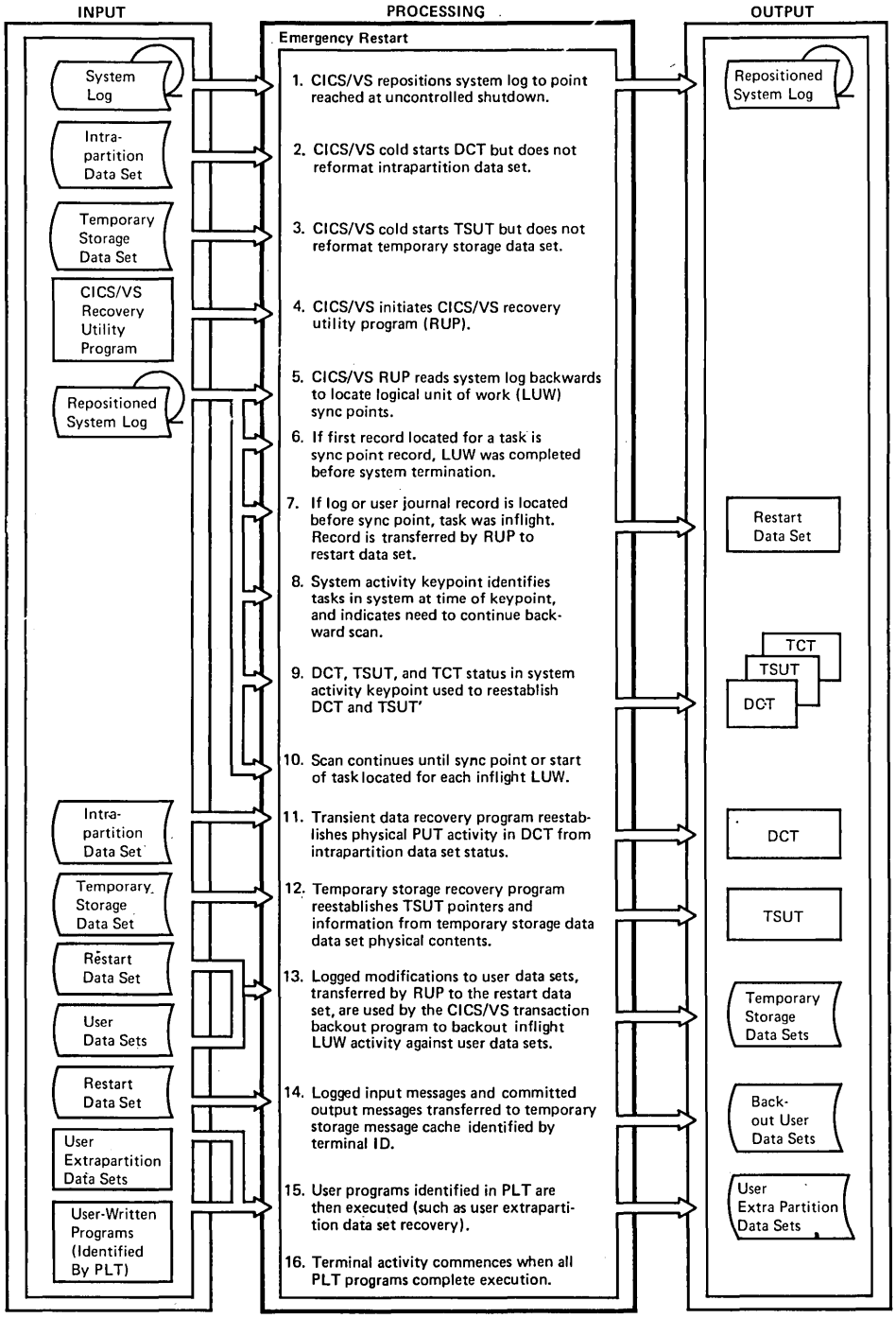


Figure 5.3-3. CICS/VS Emergency Restart Procedure

1. Reposition System Log Data Set

The system initialization table (SIT) is used by the system initialization program to identify whether the system log is on tape or on disk. If it is on disk, the system log will be repositioned to the point reached at uncontrolled shutdown when it is opened for backward processing (see the "CICS/VS Post-Initialization Processing" step later in this section).

If the system log is on tape, a separate system subtask is initiated to reposition the tape to the point reached at uncontrolled shutdown. This subtask uses a CICS/VS-provided tape end-of-file utility program. The program reads the tape system log forward and locates the last journal record written prior to uncontrolled shutdown by comparing time stamps in each journal record. The tape end-of-file utility program is executed as a system subtask to allow this tape positioning to be overlapped with the emergency restart processing described in the following steps. Once the tape is repositioned, it will subsequently be read backward to determine system activity at the time of shutdown (see the "CICS/VS Post-Initialization Processing" step later in this section).

2. Transient Data Initialization

The system initialization program performs a normal cold start function for the destination control table (DCT) at this stage, but does not reformat the intrapartition data set. The status of extrapartition data sets is lost following an uncontrolled shutdown. The status of intrapartition destinations will subsequently be recovered for those destinations identified in the DCT as being recoverable. This recovery is carried out by the CICS/VS-provided transient data recovery program (TDRP) which is discussed in the "CICS/VS Recovery Utility Program" step later in this section.

3. Temporary Storage Initialization

The system initialization program performs a normal cold start function for temporary storage but does not reformat the temporary storage data set. Temporary storage data in dynamic storage is lost following an uncontrolled shutdown.

4. CICS/VS Post-Initialization Processing

The remainder of emergency restart processing is accomplished by the CICS/VS-provided recovery utility program. This is executed as a normal CICS/VS application program, under control of the terminal control program's TCA. (This TCA is used, because normal CICS/VS operation and terminal activity have not been initiated at this time.) Upon completion of the system initialization steps outlined above, and after the system log has been repositioned (if tape) to the point reached on uncontrolled shutdown, control is then passed to the recovery utility program.

5. CICS/VS Recovery Utility Program (RUP)

The recovery utility program (RUP) reads the system log (either on tape or disk) backward, to determine system activity prior to the uncontrolled shutdown. As the log is read backward, task synchronization records (sync points) are located. (See "Logical Units of Work and Synchronization Points," earlier in this chapter.) These define the normal completion of a logical unit of work for a task. If any user data set modifications (logged either automatically by CICS/VS, or by the user) or any user journal records are located for a task before a sync point (or its initial log record) is read, this indicates that the task had not completed a logical unit of work at CICS/VS

uncontrolled shutdown (that is, was "in-flight"). The data set modifications carried out by an in-flight task may need to be backed out to the previous sync point for that task, or to the start of the task.

CICS/VS carries out this backout later using the transaction backout program. The recovery utility program identifies in-flight tasks. It collects data set or user journal records for in-flight tasks which were written after the start of the task or after a sync point, and copies them to the restart data set. As the system log is read backward, the restart data set is written forward. The restart data set therefore will reflect in-flight task activity prior to the uncontrolled shutdown. The restart data set can subsequently be read by user-written backout programs, which are executed as post-initialization programs specified in the program list table (PLT). The user-written backout programs can be automatically initiated by CICS/VS following emergency restart, and before terminal activity starts.

The first record located for a task on the backward scan of the system log may be a sync point, indicating normal completion of a logical unit of work. Transaction backout is then not necessary, and journaled or logged records for that task are not copied to the restart data set. (Note: Records output by completed tasks are collected to the restart data set only if the user specifies a special journal-type code with the high-order bit ON.)

A log record located for a task on the backward scan of the system log may have been automatically logged by the CICS/VS transient data program for intrapartition destinations identified in the DCT as recoverable. These records are used by the recovery utility programs to reestablish the DCT status for the relevant destination.

The backward scan continues until the following two conditions occur: (1) a system activity keypoint is reached, and (2) all journal and log records output by in-flight LUWs have been retrieved. The system activity keypoint contains information defining the status of intrapartition and temporary storage destinations, TCAs for in-flight tasks in the CICS/VS system at the time of the keypoint, and TCT entries for VTAM terminals with committed output messages outstanding. The status of intrapartition destinations is used to update the DCT, to back out intrapartition activity carried out by in-flight tasks, and to reflect the activity carried out by completed tasks.

The TCAs in the system activity keypoint indicate in-flight tasks at the time of the keypoint. A long-running task may have been present when the activity keypoint was taken, but may not have logged any processing activity, or written a sync point between the system activity keypoint and the point when uncontrolled shutdown occurred. Such a task had not completed a logical unit of work at the time of uncontrolled shutdown. The backward scan must therefore be continued until a sync point, or first record logged, for that long-running task is encountered. If the first record located for the task is a sync point, the task completed a logical unit of work and no backout is necessary. If a data set modification log record, or a user journal record is encountered before a sync point, those records are copied to the restart data set.

The TCT identification in the system activity keypoint of terminals which have committed output messages outstanding identifies a need to continue the backward scan until these logged output messages are located. These committed output messages are transferred to the temporary storage message cache for each relevant terminal. The TCT information in the system activity keypoint is used to prime the TCT with the VTAM sequence numbers from the last completed LUW from each VTAM terminal for subsequent message resynchronization by CICS/VS with the programmable controller.

Activity keypoints identify the necessity of continuing the backward scan until all in-flight tasks have been accounted for. Without activity keypoints, it would not be possible to identify all in-flight tasks without scanning the entire system log backward to its start.

The recovery utility program (RUP) identifies in-flight task activity, and transfers automatically logged file control activity, and user journaled activity from the system log to the restart data set. It reestablishes the DCT status of recoverable intrapartition destinations using information from the latest system activity keypoint, sync point, or end-of-task records of completed LUWs which had activity against recoverable destinations.

The transient data recovery program (TDRP) then scans physically recoverable destination queues on the intrapartition data set to locate their latest PUT activity prior to uncontrolled shutdown. This is used to establish the PUT pointer in the DCT for each physically recoverable destination.

The temporary storage recovery program (TSRP) uses status information collected by RUP from the latest activity keypoint, sync point, or end of task records of completed LUWs which had exclusive control activity against temporary storage destinations (DATAIDs). This status information reflects the logical status of each destination at uncontrolled shutdown.

Following the above RUP, TDRP, and TSRP functions, a new system log for tape, a current extent for disk, and user journal data sets are then opened by the system initialization program for output.

6. CICS/VS Transaction Backout Program (TBP)

RUP processing identifies in-flight LUWs and collects their automatically logged file control activity and user-journaled (to the log) activity on the restart data set. RUP also identifies user data sets and DL/I data bases which had in-flight activity transferred to the restart data set. Originating input messages and unresponded output messages for in-flight message-protected tasks are also written to the restart data set.

The CICS/VS transaction backout program (TBP) is executed after completion of RUP processing, and after a new system log (tape), current extent (disk), and user journal data sets have been opened for output. The purpose of TBP is to back out all in-flight activity against user data sets based on information read from the restart data set.

For messages, TBP places originating input messages and committed output messages in a temporary storage message "cache", and primes TCTTEs with the VTAM sequence numbers to be used for reestablishing message traffic. Other resources that can be backed out are CICS/VS files, DL/I data sets and Temporary Storage.

TBP provides a number of exits to permit the user to participate in data set backout. A TBP initialization exit enables the user to ignore backout against specific data sets. For example, uncontrolled shutdown may have occurred because of an unrecoverable I/O error against a data set. This data set must be recovered by user programs from a backup copy prior to emergency restart. Consequently, these data sets should not have backout activity during emergency restart.

An input exit is also provided. This is given control each time a record has been read from the restart data set and the user may choose to ignore specific records.

An error exit is also given control if an error condition is returned by file control when TBP attempts to back out data set activity. The user may specify alternative activity to be undertaken. (One instance when this exit is given control is when TBP cannot back out an add due to the file organization.) If so, the user may back out adds against VSAM entry-sequenced data sets or ISAM or DAM data sets by flagging the added record as "logically deleted." TBP then writes this logically deleted record to the relevant data set on return from the input exit. User application programs must subsequently check this flag to identify logically deleted records in the data set.

7. Message Resynchronization For VTAM Terminals

During TBP processing, input messages for in-flight LUWs or committed output messages are transferred to a temporary storage message cache identified by the relevant terminal associated with the message. Committed output messages, for which positive response had not been received from the terminal, are optionally retransmitted during emergency restart. Input messages for in-flight LUWs can be retrieved from temporary storage by user-written programs for reprocessing.

CICS/VS must establish resynchronization with those VTAM terminals that support it, on emergency restart. This is done using the VTAM sequence numbers placed by TBP in the TCT. These were established by RUP from information in the latest system activity keypoint, sync point, or end-of-task record for the last, completed, LUW against each VTAM terminal. CICS/VS issues VTAM STSN (set and test sequence number) commands to each VTAM logical unit, notifying each programmable controller of the sequence numbers known by CICS/VS. The programmable controller can compare these sequence numbers with those logged on its own disk to determine whether any messages were lost because of the uncontrolled shutdown. These may either be input messages for protected tasks, which should be retransmitted to CICS/VS by the programmable controller, or may be committed output messages.

Further information on VTAM set and test sequence number commands can be found in the appropriate CICS/VS subsystem guides.

Both CICS/VS and the programmable controllers participate in message resynchronization to ensure that no protected messages are lost because of the uncontrolled shutdown. For a discussion of message resynchronization with TCAM refer to the OS/VS TCAM System Programmer's Guide, the OS/VS TCAM Application Programmer's Guide, and the OS/VS TCAM Concepts and Applications.

8. User Post-Initialization Programs

Following execution of the recovery utility program and the opening of new journals, user-written programs identified in the program list table (PLT) are executed. These programs may carry out application-dependent recovery functions, such as repositioning extrapartition data sets but their execution may be preceded by transactions initiated by interval control or transient data control.

9. Terminal Control Activation

At this point, emergency restart and user backout are complete. The system initialization program (SIP) then initiates a system activity keypoint. Following this, terminal activity is initiated, and normal CICS/VS operation commences.

10. Controlled Shutdown Following Emergency Restart

If a controlled shutdown is then requested by the master terminal operator immediately following emergency restart, the warm keypoint

necessary for a controlled shutdown is taken, and CICS/VS terminates operation. The system may then be initialized at a later time by a normal warm start.

System Failure During Emergency Restart

System failure during emergency restart represents one of the most difficult types of failures to diagnose and correct. The user must be fully aware of the functions performed during emergency restart, the sequence in which these functions are performed, and the effect that abnormal termination during this operation has on data sets and tables.

Prior to initiating emergency restart, an analysis of the failure which caused the system to abnormally terminate should be performed. It is possible that the condition which caused the system to abend will also cause emergency restart to fail. One example of this could be a physically damaged data set which caused an uncontrolled shutdown, causing the identical failure during emergency restart if the CICS/VS transaction backout program attempts to back out modifications made to that data set.

If a file control data set has become physically damaged, a user-provided data set recovery program(s) must recover the data set prior to the user backout program attempting to back out modifications to this data set. Data set recovery involves restoring the contents of that data set from some previous copy, and then applying all modifications made to it since the copy was taken. CICS/VS automatic journaling can be used to keep track of data set modifications performed during online execution.

If the transient data intrapartition data set or temporary storage data set is physically damaged, it will not be possible for CICS/VS to emergency restart these facilities. CICS/VS recovery of these facilities is dependent upon the physical contents of the relevant data set as it existed prior to system failure. Therefore, if the data content of the data set has to be restored because of physical damage, CICS/VS may not be able to successfully reconstruct the DCT or TSUT to reflect the status of the restored data set.

User journaling may be utilized, if required, to produce an audit log of all system data set activity. This audit log can be created on a user journal data set, and utilized by user programs for subsequent reconstruction of all system data sets (such as intrapartition or temporary storage) which may have been physically damaged.

CICS/VS emergency restart is not complete until the CICS/VS transaction backout program has successfully completed, and an activity keypoint has been taken. (Optionally, a controlled shutdown could be taken at the completion of user back out, if system execution is to be terminated.) If any failure is encountered prior to this time during emergency restart, this procedure must be followed:

- Determine the cause of the failure

  The cause of the failure of emergency restart must be determined and corrected. If the transient data intrapartition data set is damaged, that intrapartition data set and the DCT must be cold started by CICS/VS. Its contents may subsequently be restored by the user, if required, by post-initialization (PLT) program processing. (This is also true for the temporary storage data set.) If a data set is damaged it must be physically recovered by user data set recovery programs.

- Restart Emergency Restart

  The emergency restart procedure is executed again using the original system log as input. The original system log is the tape or disk volume which was being used for output when the original system failure occurred. As this data set is not used for output during emergency restart, its contents are available to reinitiate the emergency restart procedure, and recover CICS/VS to its status prior to abnormal termination.

At the completion of emergency restart, the recovered status of CICS/VS has been recorded on the new system log if system execution is to proceed, or on the system restart data set as a warm keypoint if the system is to be terminated. This status represents the predefined point to which the system is recovered; system table, system data set, and user data set status are all logically synchronized. If restart becomes necessary from this point on, the new system log must be used for restart.

If the system was terminated upon completion of emergency restart, without an intervening system failure, the system restart data set contains the fully recovered CICS/VS status in the form of a warm keypoint. A CICS/VS warm start may be performed using this data to initiate CICS/VS execution with the recovered system status.

Chapter 5.4. User Journaling

Journaling provides a generalized facility for reporting and reviewing modifications to data bases and other important data sets.

The journal control program is table-driven from information defined by the user in the journal control table (JCT). Application programs may issue journal control commands to request specific journaling activity. (See CICS/VS Application Programmer's Reference Manual (Command Level).)

Journaling

Data may be journaled synchronously or asynchronously, thus providing for application processing overlap with journaling operations. In the synchronous mode, the requesting task is put in a WAIT state until the journal write has completed successfully, to guarantee that a journal copy of data exists on auxiliary storage before user processing continues. This mode of operation is similar to the way in which most CICS/VS management modules function: the user task receives control only when the requested operation has been performed.

The asynchronous mode allows the requesting task to retain control. The journal write is not synchronized unless and until the task requests synchronization either directly by use of the WAIT JOURNAL command, or indirectly by issuing a synchronous journal request. There is no guarantee that the journal copy of the data is written to auxiliary storage until the user task performs synchronization.

Journal records consist of a system prefix, an optional user prefix, and user data. Information placed in the system prefix includes:

- Task identification (that is, transaction code)
- Task sequence number allocated by CICS/VS to uniquely identify this task
- Terminal identification associated with the task
- Time of day
- Journal data set identification

In addition to that detailed above, the following information is provided by the automatic journaling option of file control.

- Data set identification (internally generated by CICS/VS)
- Record identification supplied by the application program

The user prefix is generally application dependent and is defined by the user. For automatic logging, the user data may comprise the record read from the data set for updating, the record to be deleted, or the record to be added to the data set. It may instead be data supplied by the user task, such as terminal messages.

SPECIFICATION OF JOURNALING

As indicated previously, automatic logging may be specified by the user in the file control table for each required protected data set. Automatically logged information is used by the TBP to back out data set modification activity initiated by in-flight tasks. The user may also specify additional journaling activity in the FCT. This is called automatic journaling, and it enables the user to request that activity (beyond that logged by file control for backout purposes) also be journaled by file control. This activity may include journaling of records after an update is completed; for example, for user programs to recover a data set following an unrecoverable I/O error based on a previous backup copy of the data set. Automatic journaling activity can be directed to the system log, or to a user journal data set. (Refer to the CICS/VS System Programmer's Reference Manual for further information on specification of automatic logging and automatic journaling.)

Journal requests may be made directly by user tasks, through the use of journal control commands. User journal records issued in this way may comprise data for audit purposes, for example, or may contain data to assist in subsequent application-dependent recovery, such as terminal messages.

USE OF JOURNALS AT SYSTEM INITIALIZATION

On a warm start of CICS/VS, no data set backout is necessary. All transaction activity was completed and the system was quiesced when the previous controlled shutdown took place.

On an emergency restart, the CICS/VS-provided recovery utility program (RUP) identifies all in-flight tasks at uncontrolled shutdown. It transfers all automatically journaled data set modification journal records (and other user journal records), written to the system log by the in-flight tasks to the restart data set. The CICS/VS transaction backout program backs out protected data set activity initiated by in-flight tasks. The post-initialization phase is then entered.

During this phase of system initialization, user programs identified in the program list table for use during post-initialization are executed. These user programs may issue journal control commands to access user journal data sets, or may read data set journal records from the restart data set.

JOURNAL REQUESTS

The following types of journal requests can be requested during normal execution of CICS/VS by application programs.

JOURNAL
: Create a logical record for the relevant journal data set.

WAIT JOURNAL
: Synchronize request.

JOURNAL WAIT
: Creates record and synchronizes request.

NOTE
: Note current logical record positioning of journal data set.

HANDLE CONDITION
: Handle any exceptional conditions.

POINT
: Position journal to a specified logical journal record.

GETF
: Get (forward) next logical journal record.

GETB
: Get (backward) next logical journal record.

For further discussion of the JOURNAL, WAIT, and HANDLE CONDITION journal requests, refer to the CICS/VS Application Programmer's Reference Manual (Command Level). For further discussion of the NOTE, POINT, GETF and GETB journal requests, refer to the CICS/VS System Programmer's Reference Manual.

TRANSACTION JOURNALS

The journaling facility of CICS/VS can be utilized by user programs to journal any application-dependent information, which can subsequently be retrieved by user-written post-initialization programs. This journaled information may include terminal transactions. A transaction journal produced in this way provides a record of every terminal transaction received by the system, and may also include every output message sent to terminals. A terminal input message may be written by the user to a unique user journal data set and/or the system log, immediately after a task is given control to start processing the input message. The journaling of the input message should be the first activity carried out by the initial program which operates on that transaction.

The transaction journal may be used for audit and for transaction recovery purposes. Transaction journals developed in this way may avoid the need for TCAM terminal operators to retransmit transactions on system restart, if those transactions had been entered completely before system termination.

The first input message received from VTAM terminals for each logical unit of work carried out by message-protected tasks (as specified in the PCT) is automatically logged by the VTAM terminal control program. Similarly, committed output messages are logged. CICS/VS uses these logged messages to establish message recovery and resynchronization with VTAM programmable controllers on emergency restart. Input messages belonging to in-flight LUWs or committed output messages for which a positive response had not been received, are transferred during emergency restart to temporary storage. In-flight input messages in temporary storage can be utilized to resubmit transactions for reprocessing, after their activity has been backed out. The final decision as to whether transactions must be resubmitted on restart depends upon the application requirements.

Preparation of User Journals

Disk extents that receive journal data set output must be allocated and preformatted prior to their use in CICS/VS execution. A journal format utility program is supplied by CICS/VS to complete this formatting. (See the appropriate CICS/VS System Programmer's Guide.) Once formatted, extents can be (and are) reused for successive CICS/VS executions. The system operator is notified when an extent is full, to enable the scheduling (concurrently with CICS/VS) of a user-written batch program to copy the journal extent to an archive data set on disk or tape before allowing the extent to be reused, if required by the user. A PAUSE option is provided for the user to prevent reuse of the extent by CICS/VS until the archive copy is completed. More than one journal extent may be specified by the user to permit CICS/VS to make an extent switch when one extent is full, and to continue CICS/VS operation while the full extent is being copied to the archive data set.

Tape journals may utilize either one or two tape drives. If one tape drive is used, CICS/VS directs the system operator when another tape must be mounted, and provides its own tape label management. If two tape drives are used, it automatically switches to the other tape drive and directs the operator to dismount the previously used tape.

The CICS/VS-provided tape end-of-file utility program may be used offline to reposition and close correctly a journal tape after an uncontrolled shutdown. For the tape system log, this function is performed by CICS/VS during an emergency restart. In order to enable the repositioning program to operate properly, the tapes must be formatted (with the CICS/VS-provided tape format utility program) the first time they are used for journaling.

User Journaling as a Means to Extend CICS/VS Recovery

There are five main features by which the user may extend CICS/VS recovery procedures:

- Journaling

  This is a means whereby the user may log changes made to data bases and other important resources as well as any other information he may wish to preserve.

- Program List Table (PLT)

  The program list table is a user generated table of program names. There are normally two tables, one contains a list of programs to be executed in the event of a controlled shutdown and the other a list of programs to be executed during system initialization.

- Recovery Utility Program (RUP)

  The recovery utility program processes the system log and backs out any changes that may have been made to protected resources.

- User Exits

  Program level ABEND exit points are provided by CICS/VS so that user exit programs may handle individual errors in unique ways.

- Restart Exit Program (RTY)

  Program to control which transactions are to be restarted after an abnormal termination and subsequent dynamic transaction backout.

Extrapartition Data Set Recovery

CICS/VS does not provide for recovery of extrapartition data sets. If this is significant to the online application, the system design team must develop procedures to enable that information to be recovered for continued execution on restart, following either a controlled or uncontrolled shutdown of CICS/VS.

There are two areas which must be considered in recovery of extrapartition data sets:

- Input data sets
- Output data sets

INPUT DATA SETS

The main information required on restart is the number of records processed up to the time of system termination. This may be recorded during processing using the journaling capability of CICS/VS, as described in the following paragraphs.

Each application program which reads records from extrapartition input destinations should first enqueue exclusive access to those destinations. This will prevent interleaved access to those same destinations by other concurrently executing tasks, and so enable the user's extrapartition recovery technique to operate correctly.

READQ TD commands are then issued by the application program, to read and process extrapartition input records. The application programs accumulate the total of input records read and processed during execution for each destination. The total number of READQ operations is journaled by the program to a user journal data set, together with the relevant destination identifications. This journaling is only carried out at completion of a logical unit of work, which may be at end of task or at a user sync point, such as on a conversational terminal operation.

Following output of the user journal record, the application program dequeues itself from the input destinations, to permit other application programs to access those extrapartition input destinations.

If uncontrolled shutdown occurs prior to this user journaling, no records will appear on the user journal data set for that logical unit of work, and the effect of that in-flight task is therefore automatically backed out on emergency restart. However, if the user journal record is written before uncontrolled shutdown, this completed input data set processing will be recognized on emergency restart.

On a controlled shutdown, CICS/VS will wait until the application program completes execution normally. This will enable the input data set activity to be recorded by the user for subsequent warm start.

On emergency restart following uncontrolled shutdown or on a warm start following a controlled shutdown, the following procedure may be

utilized. This will reposition the extrapartition input data sets, to reflect the input and processing of their records during previous CICS/VS operation.

An uncontrolled shutdown does not permit a tape journal data set to be closed normally. This may be achieved through the use of the CICS/VS tape end-of-file utility program prior to execution of the user recovery program.

A user-written extrapartition input recovery program may be identified in the PLT for execution during the post-initialization phase. This program reads the user journal data set forward. Each journaled record indicates the number of READQ operations performed on the relevant extrapartition input data set during previous execution of application programs. The same number of READ TD commands is issued again by the recovery program, to the same input destination as referenced previously.

On reaching the end of the user journal data set, the intrapartition input data sets are positioned at the same point they had reached prior to initiation of tasks which were in-flight at uncontrolled shutdown. The result is the logical recovery of these input data sets with in-flight task activity backed out.

OUTPUT DATA SETS

The user recovery of output data sets is somewhat different from the recovery of input data sets.

For a tape output data set, a new output tape should be used on restart. The previous output tape can be utilized if necessary to recover information recorded prior to termination.

To avoid the loss of data in tape output buffers on termination, it may be desirable to write unblocked records. Alternatively, data may be written to an intrapartition disk destination (which is recovered by CICS/VS on a warm start or emergency restart) and periodically copied to the extrapartition tape destination by an automatically initiated task. In the event of termination, the data is still available on restart to be recopied.

If a controlled shutdown of CICS/VS occurred, the previous output tape is closed correctly, and a tapemark is written. However, on an uncontrolled shutdown such as on a power failure or machine check, a tapemark will not be written to indicate the end of the tape. This may be achieved by execution of the CICS/VS tape end-of-file utility program prior to use of that tape for subsequent recovery.

For a line printer output data set, if it is satisfactory to continue output from the point reached prior to system termination, no special action need be taken. However, if it is desired to continue output from a defined point, such as at the beginning of a page, it may be necessary to use a journal data set. As each page is completed during normal CICS/VS operation, that fact may be noted by writing a record to a journal data set. On restart, the page which was being processed at the time of failure can be identified from the journal data set, and that page reprocessed to reproduce that same output again, from the beginning of the page. Alternatively, an intermediate intrapartition destination may be used (as previously described) for tape output buffers.

Transaction Recovery and Restart

The following sections discuss message recovery and transaction restart considerations for VTAM and BTAM terminals.

RECOVERY OF MESSAGES ASSOCIATED WITH VTAM TERMINALS

CICS/VS automatically logs input messages for transactions specifed in the PCT as "message-protected," and logs committed output messages and responses indicating their subsequent positive receipt by the terminal. Automatic logging only applies to message-protected transactions associated with VTAM terminals which can support message recovery. It is a user responsibility to carry out this journaling for transactions associated with BTAM terminals. Inquiry transactions (which do not update data sets) should not be specified as "protected" in the PCT.

The CICS/VS recovery utility program (RUP) identifies in-flight tasks during emergency restart and transfers logged input messages or committed output messages (for which receipt acknowledgement is outstanding) to temporary storage. A temporary storage message "cache" is defined for each terminal which had an in-flight task at uncontrolled shutdown. This message cache has a unique temporary storage queue name of DFHMXXXX (where XXXX=four-character terminal identification). This cache contains either the input message for the in-flight LUW, or, if the LUW completed normally but definite receipt of a committed output message was not acknowledged by the terminal, the committed output message.

A committed output message can optionally be automatically retransmitted by CICS/VS on emergency restart to establish message resynchronization. Thus, committed output message integrity is preserved.

An input message for an in-flight LUW is not automatically reprocessed by CICS/VS on emergency restart. CICS/VS will back out all in-flight activity for that LUW. The decision to reprocess the input message is a user responsibility based on factors such as application requirements and security of information. A technique for user-activated transaction restart is discussed later in this chapter.

RECOVERY OF MESSAGES ASSOCIATED WITH BTAM TERMINALS

To provide for implementation of a user-written transaction restart program, each user task must journal its terminal input message as soon as it commences execution. This can be achieved either by each task LINKing to an installation-level, user-written transaction journal program, or by specifying this transaction journal program in the PCT for each transaction code for which input messages are to be journaled. In this latter case, after journaling the input message the user-written transaction journaling program examines the transaction code in the input message, and transfers control to the relevant application program.

Alternatively, journaling of terminal input messages may be carried out by user-written code in a terminal control program user exit.

The information which would be included by the user in the transaction journal record is detailed below. Some of this information is automatically provided in the system prefix by journal control.

- Terminal identification
- Operator identification
- Transaction code used to attach the task (extracted from the TCA)
- Time of day when the journal record was written
- Task priority from the task control area (TCA)
- Other information required to further identify that transaction

The above information is inserted in the system and user prefixes of the journal record, and the original input transaction from the terminal input area may be placed by the user's transaction journal program in the data section of the journal record.

The user-written transaction journal program may journal terminal input messages to a user journal data set and/or the system log. Input messages written to the user journal data set can be used for audit purposes, if required. Input messages written to the system log will be transferred to the restart data set by the CICS/VS recovery utility program, for in-flight tasks when an uncontrolled shutdown occurred. This is carried out during emergency restart.

Following RUP processing, the in-flight activity of that task against data bases, intrapartition, and temporary storage destinations is backed out by CICS/VS. User-written programs identified in the program list table (PLT) are then executed.

A user-written PLT program should read terminal input messages from the restart data set, where they are transferred by RUP. This program should construct a temporary storage record, using the same temporary storage message cache format as that established by CICS/VS for VTAM terminals. This contains the terminal identification, transaction code, and task sequence number. This information can be extracted from the system prefix which was constructed for the record on the restart data set when it was originally written by journal control to the system log. The program should flag the message as a task-originating input record and then write the input message (now in the VTAM temporary storage message cache format) to temporary storage using a DATAID of DFHMXXXX, where XXXX=four-character terminal identification.

The input message is then available to the user for reprocessing based on application requirements and security information. By using the same format as that used for the VTAM temporary storage message cache, reprocessing of input messages from both BTAM and VTAM terminals can be handled consistently.

As BTAM terminals are unable to indicate definite receipt of specific output messages as can VTAM terminals, the VTAM concept of committed output messages is not applicable to BTAM terminals. Consequently, output message journaling by the user cannot ensure output message integrity. User application programs should delay issuing BTAM terminal control writes until the completion of the logical unit of work. It is a user responsibility to identify to terminal operators on emergency restart the transactions which were in-flight at uncontrolled shutdown and subsequently backed out. This is necessary to ensure that the backed out transactions (whose application and security requirements permit) are reprocessed. The following is a technique for user-activated transaction restart following emergency restart.

RESTART TRANSACTIONS (BTAM AND VTAM TERMINALS)

At the completion of emergency restart, input messages for in-flight tasks which have been backed out are available in the temporary storage message cache for each relevant terminal. An inquiry program should be written by the user so that terminal operators can determine whether their last transaction was fully processed prior to uncontrolled shutdown, or whether it was backed out on emergency restart. This program should issue a READQ TS command, using as the queue name DFHMXXXX, where XXXX=four character identification of the enquiring terminal. If that terminal had no in-flight task, an QIDERR error indication will be returned to the program. (See the appropriate CICS/VS Application Programmer's Reference Manual.) If the task was in-flight (or had a VTAM-committed output message), a temporary storage record will be returned to the program. The program should check the temporary storage message cache record flag for presence of an input message, and then present that input message and associated information to the terminal operator. The terminal operator can then decide whether the transaction is to be reprocessed. A reprocessing request by the terminal operator should only be accepted if the terminal operator has the necessary authority (based on security codes, or operator class in the TCT) to make that decision for the particular transaction. Processing then proceeds as if the transaction has just been entered from the terminal.

If the input message is associated with a VTAM programmable controller, the inquiry program may be automatically initiated by the controller after message resynchronization and recovery have been completed. The in-flight input message (transmitted back to the controller by the inquiry program) may be presented automatically to the relevant terminal operator for a reprocessing decision. Alternatively, if application and security considerations permit, the controller itself may automatically make the reprocessing decision and notify the enquiry program accordingly.

TERMINAL OPERATOR RESTART

The previously described technique enables the terminal operator to determine, on emergency restart, the status of transactions submitted prior to uncontrolled shutdown. By initiating that transaction inquiry program, the terminal operator is notified whether his last transaction was completely processed, or was in-flight. If it was in-flight (and therefore backed out), he is presented with the original input message to decide whether reprocessing is necessary. If a committed output message had not been received by his VTAM terminal prior to uncontrolled shutdown, it is retransmitted on emergency restart.

Thus, the terminal operator is presented with sufficient information on emergency restart to allow him to identify the point reached in previous communication with CICS/VS, and reestablish application processing activity.

Part 6. Performance Design

Chapter 6.1. Introduction to Performance Considerations

Introduction

The title of this Part could equally have been "Cost and Efficiency Design" in that performance is equivalent to cost and efficiency, and is probably the key to determining whether an online system is viable.

In general terms, an online computing system is composed of a number of resources, each essential to the system. Not the least of these is the human resource necessary to operate the terminals and indirectly the external resources controlled by them.

A major design objective of the design process should be to optimize the use of the total system resources, thus minimizing the cost of each unit of work.

To achieve this, a balanced system is required, which makes maximum use of the most expensive resources. It is expected that the first stage of design will be to set response, load, and cost objectives for the computing system. These are often based on a previous feasibility study and they should take into consideration the functions and cost of the associated external resources (people, and the functions they carry out). For example, in an online system controlling the movements of a fleet of aircraft or ships, the computer may well be the least expensive resource, whereas in an online mail order system the cost and efficiency of the system may well have a significant impact on profit margins.

Even after a system has been successfully designed and implemented, it is essential that the performance is carefully monitored and tuned to ensure that efficiency is maintained as the inevitable changes in the function and load on the system occur.

Performance information in CICS/VS publications is concentrated in three areas. This chapter discusses the performance aspects that should be considered when designing the system and writing the applications. When the system has been designed, the system programmer will have the responsibility of generating and maintaining the system within the constraints of the design. There are a significant number of generation parameters which will affect performance. These parameters are discussed in the appropriate (VSE or OS/VS) CICS/VS System Programmer's Guide.

Both the designer and the system programmer will need to estimate the approximate performance (response and resource utilization) of the system. Data, which will allow approximate calculations of the host processor performance to be made, is presented in the CICS/VS System Programmer's Reference Manual. Calculation of the total system performance is affected by such items as the communications controllers, TP links, and terminal devices. While quantitative discussion of the performance of such items is outside the scope of CICS/VS publications, some qualitative discussion is given.

The main topics discussed in the remainder of this chapter are about how the main (host processor) performance characteristics of CICS/VS, namely, response, maximum load, processor utilization, and storage utilization, can be affected by system design and application design.

Performance Aspects of Design

The main performance characteristics that should be considered when designing the system are:

- Response Time
- Maximum Load
- Virtual and Real Storage Utilization
- Pathlength and Processor Utilization
- Physical Database Utilization
- Network Utilization

RESPONSE TIME

The response time of the system is the sum of the response times of all the individual components of the system. These will usually include disks, channels, processor, communications controllers, and TP links. At low load (when only one transaction is processed at a time) the component response times are independent of each other and can be summed to give the total response time. It is usually possible to calculate the response of each component and summing them will give the minimum response. Normally it is inefficient to run a system this way because most resources will be heavily under-utilized.

As the load increases the response times will rise slowly, because of queueing, until one component becomes overloaded. Generally this increase will become a major part of the overall response and will constitute a bottleneck. The characteristic shape of the Response versus Load (or resource utilization) graph is shown in Figure 6.1-1.

Removal of the bottleneck (by either redesign of the application or allocating more of the bottleneck resource) will allow higher utilization of the other resources, thereby either reducing the cost of each transaction (by increasing the load) or reducing the response, thus allowing the productivity of the operator to rise. A basic objective is to achieve the desired load consistent with maintaining the required response, thus minimizing the cost of each unit of work.

If a system response is degraded, the problem is to locate the bottleneck or bottlenecks. If the bottleneck is in one of the more expensive or fixed resources (for example, the processor), the load must be reduced or the application redesigned. For most systems the change in the critical resource to go from acceptable to unacceptable response is quite small. This is illustrated in the graph shown in Figure 6.1-1. This factor allows "tuning" to be an acceptable technique in achieving or maintaining the response of a system. Tuning is a technique whereby small changes in resource utilization can be achieved by making changes to software (mainly CICS/VS) parameters. Because of the sensitivity of the response/load curve, relatively large changes in response can be achieved around the maximum load point. However, in normal circumstances, tuning is <u>not</u> an answer for inadequate resources and poor design and will <u>not</u> allow a significant increase in the maximum load.

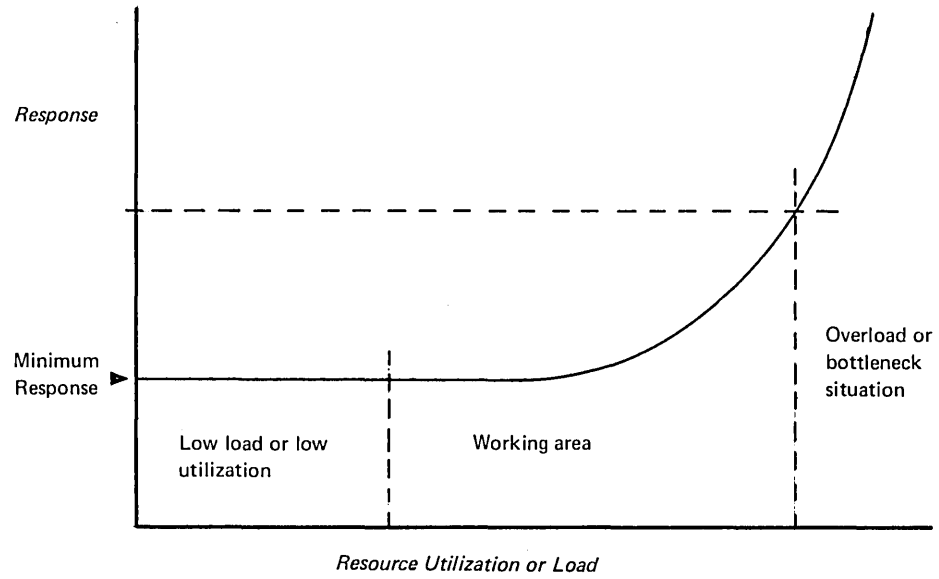


Figure 6.1-1. Typical Response Characteristic

MAXIMUM LOAD

The maximum load the system will sustain is dependent on the limiting resource. It is usual for the limiting resource to be the most expensive resource to obtain the best value for money. Often this resource is the processor.

A knowledge of the maximum achievable load consistent with the required response is important. Attempts to load the system still further will usually result in longer response times and even lower effective throughput, because the system brings into action complicated regulatory mechanisms that use up resources.

It is essential that the Master Terminal Operator controlling the system is aware of the expected loading limits, so that he can regulate the load before unacceptable response occurs. The design team/system programmer should provide the Master Terminal Operator with these load limits.

VIRTUAL AND REAL STORAGE UTILIZATION

In an online TP system such as CICS/VS, some fundamental parts of the system code will be executed many thousands of times at comparatively regular intervals. Although peaks and troughs in the load will occur and occasionally exceptional conditions will arise, there is, for any one installation, a fairly well-defined set of program code, control blocks, and data areas that are always in use. To achieve the best response, enough real storage should be available to accomodate this reference or working set of programs and data areas.

Prior to the availability of virtual storage, the normal technique was to have the main program and data area resident in main storage and to dynamically load the less frequently used programs as required. This

feature is still available in CICS/VS. The aim of the designer should be to ensure that the reference or working set is small enough to reside in real storage and allow the system to page-in the other parts of the application as and when necessary.

PATHLENGTH AND PROCESSOR UTILIZATION

In most systems processor power is the most expensive resource. For overall system cost efficiency it is generally true that the processor utilization should be high, consistent with a low instruction pathlength per transaction. Under normal circumstances, the time taken to queue for use of the processor and execute the instructions that make up a transaction represents a small proportion of the total response time. This means that processor utilization can reach relatively high levels (80% or over) without causing significant response degradation.

A factor that can be significant in achieving a high processor utilization is the ability to overlap processor processing with I/O activity. This can normally be achieved by multitasking (as provided by CICS/VS).

A measure of the cost of a transaction is its average instruction pathlength or the time taken to execute these instructions. Examples and the data necessary to calculate processor utilization are given in the CICS/VS System Programmer's Reference Manual.

PHYSICAL DATABASE UTILIZATION

The I/O hardware associated with the database is another significant resource in the computer system. To achieve low cost per transaction consistent with good response, it is necessary to perform the required logical operations with the minimum number of physical database accesses and have either a sufficiently high transaction rate or be able to reduce the amount of I/O equipment to achieve high utilization of the database.

The main components can be considered to be the channel and the device. Although this discussion is headed "Data Base Utilization", it applies to all I/O hardware, covering devices holding all files and the page dataset for paging I/O.

NETWORK UTILIZATION

The network is another significant resource in the computer system. In some large systems dedicated to teleprocessing it may be the most expensive resource. The main components of the network are the communication controllers, communications lines and associated modems, terminal control units, and the terminals or logical units.

The main considerations in determining the efficiency and utilization of the network are the transaction rate and the amount of data transmitted per transaction.

In absolute terms, the network is usually responsible for the biggest component of the response time. In an efficient interactive system, the network component of the response time usually lies somewhere between a

quarter of a second and ten seconds, whereas the host processor component of the response time will probably be between one twentieth and one half of a second.

This may mean that a low-cost high-utilization network design may have to be modified to allocate more resources in order to achieve the required response, thus obtaining a better total system efficiency.

Design Criteria

Response and utilization information is essential to ensure that effective decisions are made when designing an online teleprocessing system. Many decisions made solely on functional grounds can have significant consequences in terms of the completed system being able to meet its basic objectives.

Each major functional decision should be considered in terms of quantitative estimates of the performance and cost of the different alternatives.

Data is presented in the CICS/VS System Programmer's Reference Manual to enable the designer to estimate the performance of the host processor. This is obviously concentrated on CICS/VS, although data pertaining to the different software facilities used by CICS/VS is discussed. This data is intended to be used to:

- Check the overall feasibility of the desired application load in relation to the hardware
- Allow the designer to evaluate trade offs between various functions

This section discusses the performance aspects of the following topics:

- Application design
- System design
- Communications design
- Database design
- Program design
- Human factors
- Performance monitoring
- Recovery, security, and debugging
- Online control and modification of the system

APPLICATION DESIGN

The "application" consists of all those functions required and used to satisfy the needs of the users of the computing system. These functions will be partly satisfied by the application programs and partly by calls to CICS/VS which, in turn, will use other programs and systems facilities.

The requirement for a particular type of function may imply a particular CICS/VS function, which may, in turn, imply a need for another system resource. For example, a requirement for a scratch-pad type of facility could be satisfied by CICS/VS Temporary Storage Facility. This in turn may imply the need for VSAM and adequate real and virtual storage to support it. In a large system this would not be of consequence, while in a small system this choice could have significant impact on the overall performance if VSAM was not required for other purposes.

In addition to this type of functional judgement, it is essential to have firm estimates of the expected load on the system and any variation of the load at different periods, together with estimates for the inevitable expansion of the system. This expansion, whether in new applications or increased load, will inevitably have an impact on system response. It is recommended that the application is designed with these objectives in mind. For example, in many installations there will be well-defined peaks and troughs in activity. It may be beneficial to design automatic procedures to initiate what are essentially off-line programs as low-priority CICS/VS tasks during slack periods to make use of the spare capacity. The technique of top-down design and programming can be used to performance estimation and simulation. As each level of design is reached, estimates of the performance of each "box" (preferably upper and lower limits) should be made. These can be used to predict performance, detect bottlenecks, and provide objectives for the program coder. "Boxes" that are used very frequently can be given special attention to achieve optimum performance.

When the top-down approach is used to write the application, parts of the program not available can be crudely simulated by instruction loops, so that a better idea of likely performance can be obtained.

Application programs should be kept as simple as possible. Sophisticated features invariably have an impact on performance and sophistication should be avoided unless it can be cost-justified.

In summary, it is essential to know what the application design implies in terms of performance, not only in the host processor, but in all aspects of the total system.

SYSTEM DESIGN

This area covers the design of the total system, including CICS/VS generation, choice of operating system and access method parameters, hardware, and network configuration. This is usually done under the constraints of the application design, available hardware (or funds), and programming expertise.

Choice between different features should depend on both functional capability and performance. For example, a choice between TP access methods VTAM and BTAM should not only take into consideration the functional aspects (such as the terminal types supported) but also the relative performances. In crude terms, CICS/VS BTAM requires much less

real storage than CICS/VS VTAM, but CICS/VS VTAM requires less processor power, especially for larger networks. Hence constraints on the type of processor and available real storage could have a significant impact on the choice of TP access method, or vice versa.

The main alternative software choices to be made are as follows.

Operating Systems - VSE, OS/VS1, OS/VS2

In general terms, VSE is the simplest and fastest operating system and OS/VS2 (MVS) is the slowest and most sophisticated system. Where the proposed CICS/VS system is only one of a number of systems running concurrently on the same processor, it may mean that the more sophisticated operating system will give better overall system performance. Alternatively, if the installation is running a dedicated CICS/VS system with a comparatively simple application, VSE will probably be the cheapest and most efficient choice, irrespective of processor size. However, no hard and fast rule should be adopted. Each case should be evaluated on its own merits.

TP Access Methods - BTAM, VTAM, TCAM (CICS/OS/VS only)

These access methods have significantly different performance characteristics which impact the choice of host processor. The ability of VTAM to support intelligent terminals and sub-systems (such as the IBM 3790 Communication System) also has significant performance consequences because some of the work normally done in the processor can be off-loaded. In some cases, a significant percentage of transactions can be completely handled by the intelligent terminal (and its controller) without the need for a transmission to the host processor.

Unless there are very special reasons, it is advisable to only use a single access method.

Database Access Methods

The main alternatives in this area are:

- Hardware-oriented access methods (DAM, ISAM)
- Virtual storage access method (VSAM)
- Complex database access method (DL/I)

The choice between DAM, ISAM, and VSAM from the performance point of view is basically that DAM and ISAM are slower than VSAM but require less real storage. If the database is on a fixed block architecture (FBA) device, neither DAM nor ISAM can be used. DL/I requires an additional amount of real storage and overhead in processor power. If the database is very complex and is to be interrogated by many different types of keys, or if complex relationships are necessary, the need for extra storage and power can be offset by better performance by the terminal operator and program coder, and quicker access to the actual physical data. However, it is very important to cost-justify the decision.

PL/I Shared Library (CICS/OS/VS)

The PL/I Optimizing Compiler has a facility whereby those resident library modules likely to be used in more than one program simultaneously can be stored together in the link-pack area, whence they can be invoked from any region. This facility, known as the PL/I Shared Library, is available to PL/I programs running as CICS/OS/VS applications, provided that they were compiled by the PL/I Optimizing Compiler. The PL/I Shared Library is another facility that helps the user to conserve storage.

CICS/DOS/VS Shared Library (PL/I)

The CICS/DOS/VS Shared Library facility enables some of the PL/I Resident Library modules to be used by more than one PL/I program running under CICS/DOS/VS. This avoids duplication of code. For further details see DOS PL/I Optimizing Compiler Installation.

CICS/VS Options

The main alternatives should be thoroughly examined and cost-justified. It is especially important to examine which system resources are implied by the use of any particular CICS/VS option. Some major CICS/VS options, which have an impact on system performance, are discussed in the latter part of this chapter. These include the High Performance Option (MVS only) and the CICS/DOS/VS Entry Level System.

When these major design decisions have been made, the system programmer should investigate the individual options for each major component. A discussion of the performance implications of the major options is given in the appropriate CICS/VS System Programmer's Guide.

CICS/VS Modules in Shared Area of Operating System

Certain CICS/VS management modules may occupy the link-pack area (LPA) on OS/VS systems or the shared virtual area (SVA) on VSE systems. Eligible modules are listed in the CICS/VS System Programmer's Guide for the appropriate operating system.

Modules using these areas may be shared between two or more CICS/VS systems in the same processor system. This reduces real storage requirements and the amount of paging. Two or more such CICS/VS systems may intercommunicate, but they will benefit from sharing modules whether they communicate with each other or not.

System integrity is improved for single and multiple CICS/VS systems, because the shared areas have protection keys of 0. The CICS/VS management modules therefore cannot be overwritten by other programs, such as CICS/VS applications.

COMMUNICATIONS DESIGN

Data communications design has a significant effect on the success or failure of the project, because it is the interface between the user and the host processor, and because it has significant impact on the response and throughput of the system.

Network Design

It is not intended to enter detailed discussions on design of communications networks. It is sufficient to recommend that each connection between the host processor and the terminal user should have adequate capacity to cope with peak demands. As far as other constraints allow, the network should be balanced to match expected traffic to the capacity of each part of the link. This includes ensuring that the processor can deal with the peak traffic rate that the network can present to it. Hence an overall smooth flow will occur as messages move through the network. Normally it is reasonable to expect small queues to occur at various points because in practice, it is impossible for all links to cope with all the variations in load. Queues allow minor fluctuations in load to be smoothed out. A system with no queues can be achieved, of course, when the system is more powerful than is necessary, but this results in a higher cost per transaction than is necessary.

Message Design

Another important consideration is the amount of information passed between the terminal and the host processor. In many cases, there can be a distinct trade-off between the cost per transaction and the ease with which the terminal operator can enter or use the data.

For example, always using a formatted screen where both the actual data and the format are always transmitted can have a significant effect on performance and cost. This causes an impact both in the host processor software and the capacity of the network. Of course, other considerations are necessary. For example, it would be unreasonable to expect the general public to use an unformatted screen or to use abbreviated symbols to communicate with the computer.

Think Time

Many teleprocessing systems are designed with the underlying concept that, for a large proportion of elapsed time, the operator is either entering data or thinking. During the remainder of the time the system is processing the message. This means that although the system may only be capable of processing a small number of transactions simultaneously, it can support a large number of terminals. For example, suppose the system is capable of supporting five transactions simultaneously with an average response time of under two seconds. If the operator only enters one transaction every 60 seconds, the system would be capable of supporting up to 150 active terminals. If, however, the operator thinks and enters the data in four seconds, the system is only capable of supporting 15 active terminals. This kind of consideration is essential when designing the network (and determining the host processor requirements).

CICS/VS has several powerful and sophisticated features, which allow the user to initiate transactions with a very short think time and just one or two keystrokes. An example of this is the use of BMS Terminal Paging. This can be used so that the application will prepare more output than can be displayed at the screen at any one time. This output is split up into various pages by CICS/VS and is stored on disk by the Temporary Storage Program. By the use of paging commands, the operator can select each page of the output for display. This means that a single operator, by indiscriminate use of this function, can put a significant load on the system, because each display of a page may be equivalent to a normal transaction (terminal input, disk retrieval, terminal output).

DATABASE DESIGN

In a DB/DC system such as CICS/VS, the database can be considered to be part of the system and, as such, the general principles applicable to the system can be applied. Included in the database for the purposes of this discussion are the page dataset (for page I/O) and any other datasets (for example, CICS/VS statistics and temporary storage).

Primary considerations in database design are as follows:

- Datasets should be arranged so that traffic to the different physical devices is equalized as far as possible, especially under peak load conditions.
- The design of the database should be such that the minimum amount of data is transferred between the database and the terminal operator. A more complex database design may allow easier extraction of the pertinent information, thus allowing economies to be made in the amount of data transmitted through the network. This must of course be balanced against the extra handling necessary by the database access method and possibly a requirement for more real storage.
- The host resource requirements (storage, processor power) should match the access method requirements.
- For systems with medium or high transaction rates, the impact of extensive updating or adding of records, which normally requires exclusive use of a portion of the dataset, should be considered in some detail.

PROGRAM DESIGN

The main factors to be considered in CICS/VS program design are:

- Choice of programming language
- Module size and program structure and the method of transferring control from one program to another
- Splitting into working set and non-working set
- Resident or non-resident programs
- Conversational and non-conversational programs

Choice of Programming Language

The choice of programming language (COBOL, PL/I, RPG II (for CICS/DOS/VS only), or assembler) will primarily depend on the relative amount of programming activity and production work, together with the skills available. The higher programmer productivity of the high-level languages may more than compensate the increased resource utilization required. Another factor is the use of either the CICS/VS Macro Level Interface or the CICS/VS Command Level Interface. The command level gives higher programmer productivity than the macro level at the expense of some extra resource utilization.

In order of decreasing overhead and decreasing productivity the selection is:

1. Command Level Interface with COBOL, PL/I, or RPG II (for CICS/DOS/VS only)
2. Command Level Interface with assembler
3. Macro Level Interface with COBOL, or PL/I
4. Macro Level Interface with assembler.

The overheads take the form of increased instruction execution and a larger real storage requirement.

An idea of the relative costs of the various alternatives is given in the CICS/VS System Programmer's Reference Manual.

Note that RPG II programs (CICS/DOS/VS only) are not reentrant; therefore to allow more than one transaction to concurrently make use of a program, a separate copy of the program for each transaction will be loaded into storage.

Module Size and Program Structure

Using the normal criteria for defining module sizes and program structure will generally lead to comparatively small modules, each satisfying a particular function. This can sometimes lead to poor performance. Modules should be written to minimize the use of real storage; that is, they should occupy the minimum storage for the smallest amount of time. Because the operating system deals with storage in pages (2048 bytes for VSE and OS/VS1, 4096 for OS/VS2), any module smaller than the size of a page will still effectively utilize the whole page while it is executing. Typical maximum sizes are 12K for COBOL and PL/I programs, and 4K for Assembler programs.

CICS/VS can be used to pass control from one program module to another. While the overhead to do this is comparatively small (see the CICS/VS System Programmer's Reference Manual for details), large numbers of calls via the CICS/VS Program Control interface during a single transaction can add a significant overhead. If this is the case, it is recommended that either programs are amalgamated to form larger modules. Typically a maximum of 3 or 4 calls should be aimed at.

The overhead is both in instruction pathlength and real storage requirements.

Working Set

The concept of working set or reference set is very important, especially in systems with limited real storage. Programs (and data areas) should be split into code that is used frequently and code that is only used infrequently (for example, for error conditions). It is recommended that code be arranged so that the infrequently-used code can be in a different page from the frequently-used code. This can be done by splitting the code into different modules, or for very large modules the code can be separated into high-use and low-use sections. When considering the code packaging, all constants, control blocks, and data areas must be included.

Resident and Non-resident Application Modules

CICS/VS allows programs to be loaded into virtual storage when the system is initialized. They stay in virtual storage until the CICS/VS system closes down. They will be paged into main storage as required and the length of time they stay in main storage will depend on factors such as their frequency of use. Alternatively, CICS/VS will dynamically load application programs into the CICS/VS dynamic storage area on request. Although this process is basically the same as that carried out at initialization time, there is an important difference. It is an expensive piece of processing, and hence it can have an impact on online performance. It is far better to absorb this overhead during initialization when there are no demands for quick response. It should also be remembered that loading a program does not necessarily prevent page faults occurring when the particular program is referenced.

Conversational and Non-conversational Transactions

CICS/VS transactions can be written in two distinct modes:

- Conversational
- Non-conversational

In the conversational mode, the transaction is initiated as a result of the first piece of input from the terminal. After that, a number of inputs and outputs to and from the terminal will occur before the transaction is terminated. For certain application types (for example, text processing) this is the obvious way to construct the application. It has the disadvantage that storage resources can be held by the transaction for the whole time the transaction is operational. It is quite possible that for most of the time the operator is thinking or entering data.

Non-conversational tasks do not have this overhead because the transaction consists of a single input and one or more outputs to the terminal.

Applications that require a "conversational" approach can be coded using a sequence of non-conversational transactions. When a wait for the operator occurs, the transaction is terminated and the resources released. The next input from the terminal will then cause another transaction to be initiated and the next part of the same application can be executed as required. While this approach requires more application programming, because application data must be preserved and the application logic is probably more complex, it can make a significant difference to performance if there is insufficient real storage to match the working set. The storage overhead of conversational tasks can be alleviated by using the Anticipatory Paging feature of CICS/VS (see the appropriate CICS/VS System Programmer's Guide), thus saving the cost of continually reattaching and terminating the same transaction.

HUMAN FACTORS

There are at least two significant areas where human factors aspects of system design and performance aspects meet. These are concerned with the interaction of the terminal operator and the computing system.

Firstly, it should be remembered that facilities that are easy to use will tend to get used more often, sometimes for purposes for which they were not designed. There is often a tendency for the easily-used facilities to require more resources, and the designer may find that a projected transaction mix used to estimate system performance may become biased towards the easily-used and more expensive transactions. For example, in a text processing application it may be necessary to correct a number or misspelled word which occurs several times. This could either be implemented such that the operator will display each line containing the word and correct it separately, or by searching the whole document and automatically correcting and displaying the relevant lines. Obviously the latter is easier to use but could require substantial use of the system's resources. This could be wasteful if there are only a few occurrences of the word to be corrected.

Secondly, it is often desirable to keep the terminal user occupied. For example, in an order-entry application it may be beneficial from a resource-utilization point of view to require the operator to enter

several orders together (up to the limitation of the screen size) and then process them in a batch. If the amount of processing per batch is significant, the response time may be quite large. This could mean that the operator may get bored and, having his or her concentration disturbed, may be more prone to errors. It may be better to design the application so that less work is entered at a time and a quicker response is obtained.

PERFORMANCE MONITORING

Although an online system may be operating efficiently when it is installed, the characteristics of the system usage may change and the system will not run so efficiently. This inefficiency can usually be corrected by adjusting various controls.

The usual symptoms of inefficient usage are decreased throughput and increased response time. If these occur it leaves a feeling that the system is inefficient and poorly designed. Monitoring the performance of the system will enable the person responsible to see the trend and hopefully identify the problem before it shows up as poor response and throughput.

Most performance monitoring is done by sampling information in certain control blocks at regular intervals, and by counting the occurrence of certain critical events. CICS/VS provides certain monitoring features, and there are also features in the various access methods and operating systems that enable this monitoring to be carried out. An outline of the facilities and techniques is given in the appropriate CICS/VS System Programmer's Guide.

It is essential that the system designer lays down monitoring procedures to accumulate the desired information and procedures to make changes according to the data obtained.

Because the monitoring is usually done by sampling, it will itself have an impact on the system performance. There is a definite trade-off between the cost of over-enthusiastic monitoring and the amount of control needed. It is recommended that the user establishes the impact on system performance of using various levels of monitoring. A suitable level may well be a general monitoring of the most critical parameters, together with a detailed investigation of each new feature or application. It is recommended that performance testing of new applications be made part of the normal functional testing, which each new program will receive.

CICS/VS Monitoring Facility

This CICS/VS facility will provide monitoring information at various levels of detail.

The accounting class provides the minimum information to enable accounting routines to associate particular transactions with particular users or terminals.

The performance class provides data to enable the performance of the systems to be observed. Information like response time, CPU time, and data base calls can be collected. Charging algorithms that depend on resources actually used would use this class of data collection.

The exception class provides data on certain exception conditions that the task has encountered.

The data is collected at various event monitoring points in the CICS/VS system. These may be system or user defined. The user-defined monitoring points, the monitoring classes, and the data sets to be used for collecting the monitoring data are recorded in the Monitoring Control Table (MCT), which is generated by the DFHMCT macro.

Full details of the monitoring facility are given in the System Programmer's Reference Manual.

The CICS/VS monitoring facility is a data collection tool. CICS/VS provides no data reduction or analysis programs although the sample program DFHXMOLS will print the monitoring data.

RECOVERY, SECURITY, AND DEBUGGING

Recovery and security are very desirable features for any online system. The level of recovery and security required depends very much on the application. However, because such features can be very expensive in the resources they require, their impact on response time and on the throughput of the system, it is essential that they should be cost-justified and several alternative techniques considered.

If a program error occurs during normal online running, it can sometimes be very difficult to recreate the problem in an artificial or controlled environment. The problem can sometimes be alleviated by using such tools as CICS/VS Trace and Dump Facilities to automatically monitor the path of each transaction. This kind of facility can add a significant processor and storage requirement overhead and should be used with care.

ONLINE CONTROL AND MODIFICATION OF THE SYSTEM

The load on an online system will normally vary, often considerably, with the time of day. During peak periods it may be necessary, for example, to limit certain kinds of transactions or change certain system parameters. It is also possible to automatically initiate low-priority or "batch-like" transactions during slack periods to use spare capacity. These controls can be carried out dynamically by the system or master terminal operator by using the appropriate CICS/VS transactions.

It should be part of the system design to lay down procedures defining controls to be instituted to ensure adequate response times at peak periods and utilization of spare resources during slack periods.

Major CICS/VS Performance Options

There are two major performance oriented CICS/VS options:

- CICS/DOS/VS Entry Level System
- High Performance Option for OS/VS2 (MVS)

Other significant features which can have an impact on performance are:

- Intercommunication
- Recovery and Integrity features.

CICS/DOS/VS ENTRY LEVEL SYSTEM

This pregenerated CICS/VS product gives a system which is optimized for the situation when only one task is processed at a time. This system will give optimal performance for small installations with networks of up to about 30 terminals.

The Entry Level System provides significant economies in real storage and processor requirements by eliminating the sophisticated control mechanisms of CICS/DOS/VS.

However, if the smaller system grows and the load increases, it may be found that upgrading to CICS/DOS/VS will give better performance.

HIGH PERFORMANCE OPTION (CICS/OS/VS ONLY)

In a CICS/VS installation there is generally a requirement for the same terminal network and data to handle both simple (low-function) and complex (full-function) transactions. For many applications a high transaction-rate is required for the simple transaction types, while the complex transaction types are able to tolerate slower rates. Although it provides an improvement for all users, the CICS/OS/VS High Performance Option (HPO) is designed specifically to support the low-function, high transaction-rate requirement for MVS users.

In the case of simple transactions, HPO provides improved performance over the base CICS/OS/VS system by reducing the transaction path length (instructions). For transactions involving 1 input message, 1 output message, and 3 physical DASD accesses, the reduction is likely to be in the order of 50%. This has the effect of increasing the transaction rate that CICS/OS/VS can support, or reducing the processor utilization, or both. It makes use of the MVS Service Request Block (SRB) facility and so provides better utilization of a multiprocessor.

The main characteristics of HPO are:

- Primed storage: improved storage management.
- VTAM authorized path: improved path through VTAM.
- VSAM fast path: improved file accesses.
- Improved logging and journaling.

These characteristics are described in detail under the following headings.

The HPO function is available to MVS users via the SRBSVC=number operand (MVS only) of the DFHSG TYPE=INIT macro (for full details refer to the CICS/VS System Programmer's Reference Manual). This specification provides the VTAM authorized facility and the potential to use the VSAM fast path facility.

Primed Storage

Normally, CICS/OS/VS minimizes the amount of storage in active use by acquiring and releasing it dynamically. Storage held by a task is returned to the common storage pool when that task terminates.

The primed storage facility of HPO (obtained via the PRMSIZE= operand of the PCT macro) allows the user to specify that, at task termination, the storage be retained for later use by another task of the same transaction type. Although the amount of storage in use at any one time is increased, performance is improved by the use of this suballocation scheme for acquiring and releasing storage.

Note: Under OS/VS1, the primed storage facility may be used even if no other HPO facilities are specified.

VTAM Authorized Path

Under OS/VS2 an alternative path is available in VTAM to authorized programs. HPO uses this shorter path by scheduling Service Request Blocks (SRBs) to perform VTAM operations where possible.

File Access

Under OS/VS2 VSAM provides a fast path which gives improved performance for the accessing of control intervals. This is known as "improved control interval processing" (ICIP). This support provides the HPO user with fast path access to his data base for both ESDS and KSDS files. Those files which are to be accessed using VSAM ICIP are indicated via the MODE= operand of the DFHFCT TYPE=DATASET macro (for full details refer to the CICS/VS System Programmer's Reference Manual). The following restrictions apply to ICIP files:

- The control interval size must equal the physical block size.
- New records may not be added to a file while it is being accessed using ICIP.
- Browsing or segmenting is not supported.
- Sharing of resources (especially buffers) is not supported.
- Indirect Access and Alternate Indexes are not supported.
- Relative Record files are not supported.

Support for both ESDS and KSDS files is provided by using ICIP interface. To achieve a consistent interface with ESDS the deblocking is performed by CICS/OS/VS. For KSDS files the user specifies the level of indexes that are to be kept in core. When using the KSDS interface via ICIP both the index and data file must be defined for the file, otherwise ICIP will not be used.

The same file may be accessed using either ICIP (simple transactions) or normal VSAM (complex transactions), but not at the same time. The DFHOC TYPE=OPEN, MODE= macro specification allows the required mode to be selected. Full details of the DFHOC TYPE=macros can be found in the CICS/VS System Programmer's Reference Manual.

Statistics

HPO will maintain statistics on primed storage utilization. These include the average amount (bytes) by which the task type exceeded its initial primed allocation.

Logging and Journaling

To maintain the high performance characteristics required by HPO, the logging and journaling functions in CICS/OS/VS allow the delaying of journal I/O until either the buffer is full or after 1 second. Consequently the number of I/O operations executed for a given amount of file content is reduced.

Intercommunication

This feature of CICS/VS allows processing to be carried out, or data to be accessed, on systems different from the one on which the transaction was entered or to which the terminal is connected. In overall system efficiency terms this will represent an improvement because previously it would have either been necessary to maintain more than one copy of a database or limit the application programs that the individual systems could run.

This facility is not designed to be used to offload work from an overloaded system to an under-used system; the resources consumed by doing so are likely to be at least as great as the resources required to do the function being offloaded.

CICS/VS provides two sets of facilities that allow applications to access resources on remote systems: distributed transaction processing (DTP) and function request shipping. Both are described in Part 7. DTP is likely to prove the more efficient in terms of the use of computing resources (particularly the use of the intersystem link), but is likely to be the more costly in terms of programming effort.

Recovery and Integrity Features

CICS/VS provides a significant number of recovery and integrity features. The overhead associated with them can be divided into two parts. Firstly there is the overhead associated with the actual recovery process. Usually this is of little concern because it will only occur at very infrequent intervals.

The other part is the recording of data with every occurrence of a recoverable operation such as an update to a data set. This has an immediate impact on the total throughput and the transaction response.

Whether either CICS/VS facilities or user's application code is used to provide recovery features, care is essential in minimizing usage if costs need to be limited. Indiscriminate use has been known to degrade throughput by up to 50%, while careful selection and implementation of an effective system can easily limit degradation to less than 10% of the throughput.

Part 7. Intercommunication Design

Chapter 7.1. Introduction

A CICS/VS system may require resources owned by other CICS/VS systems, or may need to distribute processing to them. The systems may be region-remote, that is in the same processor system, or they may be domain-remote, that is in different processor systems (or, strictly, in different SNA domains). For instance, an application running in a branch office processor may need to access data owned by a head office processor. Or, if one CICS/VS system is run for production work while a second is run, in the same processor system, to test new applications, it may be desirable to execute an application in the test system from a terminal connected to the production system. Such requirements are met by the CICS/VS intercommunication facilities.

There are three types of facility:

CICS/VS function request shipping
: An application requests use of a resource owned by another CICS/VS system. The resource can be:

  a file or DL/I database

  a transient data or temporary storage queue

  another transaction

  The application program is designed and coded as if the resource were owned by the system in which the transaction is to run. Tables are set up by the system programmer indicating to CICS/VS where the resource is located, and CICS/VS ships a request for its use to the appropriate system at execution time. Function request shipping is available through the command level interface only of CICS/VS, the command level interface of DLI (CICS/DOS/VS only), and the CALL level interface of DL/I.

CICS/VS transaction routing
: A terminal owned by one CICS/VS system runs a transaction in another CICS/VS system. The transaction that is invoked may execute programs that use either the command or the macro level interface.

Distributed transaction processing (DTP)
: A CICS/VS transaction running in one system communicates synchronously with a transaction running in another system. Both programs are designed and coded explicitly to communicate with each other, and thereby utilize the intersystem link with maximum efficiency. DTP is available through the command level interface only of CICS/VS.

The mechanisms by which communication takes place are:

an SNA access method, such as VTAM. This is used for domain-remote connections.

the multiregion operation (MRO) facility of CICS/VS. This is used for region-remote connections. MRO uses an interregion SVC to pass information between regions.

DTP is available via VTAM but not via MRO; transaction routing is available via MRO but not via VTAM; CICS/VS function request shipping is available via VTAM and MRO. The situation is summarized in Figure 7.1-1.

Facility	Connection	
	Domain-remote (via VTAM)	Region-remote (via MRO)
Function request shipping	Yes	Yes
Transaction routing	No	Yes
Distributed transaction processing	Yes	No

Figure 7.1-1. Availability of Intercommunication Functions

Where a facility is available via VTAM and MRO, the performance is likely to be better via MRO.

VTAM application-to-application facilities can be used to implement CICS/VS function request shipping or DTP between CICS/VS systems running in the same processor system.

The DTP facilities can be used to communicate, via an SNA access method such as VTAM, between CICS/VS and other systems implementing suitable subsets of SNA logical unit type 6 (LU6) protocols. IMS/VS is one such system.

Note about terminology: One of the pair of CICS/VS systems using intercommunication is known as the local system, the other as the remote system. The local system is the one from whose point of view the discussion takes place. The remote system is the other one: the one that owns the a data resource for which a function request is shipped, for instance. In the case of function request shipping, the remote system can be region-remote (in the same domain) or domain-remote (in a different domain). In the case of transaction routing, the remote system is always region-remote. In the case of DTP, the remote system is always domain-remote (except if VTAM application-to-application facilities are used). Communication between two domain-remote CICS/VS systems is known as intersystem communication (ISC).

Chapter 7.2. Function Request Shipping and Transaction Routing

Function Request Shipping

Use of the function request shipping facility allows a CICS/VS application program to:

- Access files and DL/I data bases managed by other CICS/VS systems by shipping requests for file control or DL/I functions.
- Transfer data to or from transient data and temporary storage queues in other CICS/VS systems, or other non-CICS/VS systems that implement the SNA LU type 6 protocols, such as IMS/VS, by shipping requests for transient data and temporary storage functions.
- Initiate transactions in other CICS/VS systems, or other non-CICS/VS systems that implement SNA LU Type 6 protocols, such as IMS/VS, by shipping requests for interval control functions.

Applications may be written without regard for the location of the requested resource, for example, a file. Entries in the CICS/VS resource definition tables (for example, the file control table), allow the system programmer to specify that the named resource is not on the local (or requesting) system but in a remote (or owning) system.

When a command level request is made for a resource owned by another system, CICS/VS turns the request into a suitable transmission format and ships it to the remote system identified in the relevant table. On receiving the request, the remote CICS/VS system passes it to a special transaction, known as the mirror transaction, which is responsible for executing the shipped request and sending the reply to the originating system. The mirror transaction recreates the original request and issues it on the remote system.

CICS/VS recovery and restart facilities allow resources in remote systems to be updated and ensure that when the requesting application program reaches a synchronization point, the mirror transactions updating protected resources also successfully reach the synchronization point, and that changes to protected resources in remote and local systems are consistent.

Transaction Routing

The transaction routing facility allows terminals connected to one CICS/VS region to run transactions in another CICS/VS region within the same processor system. CICS/VS handles all routing of requests and replies. Transactions can be designed and coded without regard to the fact that the terminal is connected to another CICS/VS region.

Details of transactions on other CICS/VS systems are specified by the system programmer in the Program Control Table. Details of terminals on other CICS/VS systems are specified by him in the Terminal Control Table. When a terminal operator enters a transaction code for a transaction that must be executed on another system, a transaction

referred to as the relay transaction is attached on the local system. The transaction executes a CICS/VS-provided program, the relay program, that establishes a connection with the remote system.

The relay transaction sends the request to the remote system to attach the user transaction. When the user transaction issues a request to its principal terminal, that request is intercepted by the CICS/VS terminal control program and shipped back to the relay transaction. The relay transaction then issues the request to the terminal. Similarly terminal status and data are routed to the user transaction via the relay transaction.

To allow recovery after an error, the relay transaction takes a synchronization point (if necessary) whenever the user transaction does so.

Applications of Region-remote Intercommunication

INTRODUCTION

MRO enables two regions running CICS/VS in the same processor system to communicate with each other. Function requests can be shipped between them, and terminals connected to one can run transactions in the other. Some possible applications of these facilities are described in subsequent sections.

Conversion from Single Region System

Using MRO, existing single-region CICS/VS systems can generally be converted to multiregion CICS/VS systems without reprogramming, provided the command-level application programming interface has been used. The restrictions that may necessitate reprogramming are documented in the CICS/VS System Programmer's Reference Manual. CICS/VS function request shipping will allow an existing application to continue accessing existing data resources after either the application or the resource has been transferred to another CICS/VS region, provided the applicaton uses the CICS/VS command level interface. CICS/VS transaction routing will allow an existing application to be run from an existing terminal after either the application or the terminal has been transferred to another CICS/VS region, in this case whether the application uses the command level or macro level interface.

SYSTEM DEVELOPMENT

If a system is to be upgraded by the installation of VTAM, communications controllers, network control programs and TP lines, function shipping application that are to use these facilities may be tested in parallel with the installation. This is likely to allow the applications to be put into production sooner than would be possible if testing were not started until VTAM became available.

PROGRAM DEVELOPMENT

The testing of newly-written programs can be isolated from production work by running a separate CICS/VS region for testing. This enables the reliabilty and availabilty of the production system to be maintained during the development of new applications, because the production system remains up even if the test system terminates abnormally.

By using function request shipping, the test transactions can access resources of the production system, such as files or transient data queues. By using transaction routing, terminals normally connected to the production system can be used to run test transactions.

The test system can be brought up and taken down as required, without interrupting production work. During the cutover into production of the new programs, terminal operators can run transactions in the test system from their regular production terminals, and the new programs can access the full resources of the production system.

TIME-SHARING

If one CICS/VS system is used for compute-bound work, such as APL or ICCF, as well as regular DB/DC work, the response time for the DB/DC user may be unduly long. It can be improved by running the compute-bound applications in one region and the DB/DC applications in another. Transaction routing allows any terminal to access either CICS/VS system without the operator being aware that there are two different systems.

RELIABLE DATA BASE ACCESS

An installation may have two sets of programs, one of which needs to be kept running even if a program in the other set causes the system to abnormally terminate. For instance, it may be desirable to continue running batch programs even if an online program terminates the CICS/VS system. Or an installation may divide its programs into reliable and unreliable ones, and wish to be able to run the reliable ones even after an unreliable one has brought CICS/VS down.

With MRO, two CICS/VS regions can be defined, one of which owns the online or unreliable applications, the other the batch or reliable applications and the data base. The more applications that run in the region that does not own the data base, the more reliable the data-base-owning system will be. On the other hand, the cross region traffic will be greater, so performance is degraded. The system programmer can trade off performance against reliability.

DEPARTMENTAL SEPARATION

MRO allows various departments of an organization to have their own CICS/VS systems. Each can bring up and take down its own system as it requires. At the same time, each can have access to other departments' data, with access being controlled by the system programmer. A department can run a transaction on another department's system, again subject to the control of the system programmer. Terminals need not be allocated to departments, since, with transaction routing, any terminal could run a transaction on any system.

MULTIPROCESSOR PERFORMANCE

A single CICS/VS system will generally be unable to take advantage of a multiprocessor system. With MRO, using several CICS/VS systems, the user can take full advantage of a multiprocessor, and allow any terminal to access the transactions and data resources of any of the systems. Transaction routing presents the terminal operator with a single system image; the operator need not be aware that there is more than one CICS/VS system.

The system programmer can assign transactions and data resources to any of the connected systems so as to balance the load and achieve optimum performance.

Applications of Domain-remote Intercommunication

VTAM enables domain-remote CICS/VS systems to be connected together so that function requests can be shipped from one to the other. Some possible application of these facilities are described here.

Figure 7.2-1 illustrates some combinations of systems that may be set up using the CICS/VS function request shipping facilities and VTAM.

CONNECTING REGIONAL CENTERS

Many users have computer operations set up in each of the major geographical areas in which they operate. Each system has a data base organized towards the activities of that area, and its own network of terminals able to inquire or update the regional data base. When requests from one region require data from another, without intersystem communication, manual procedures have to be used to handle such requests. The intersystem communication facilities allow these "out-of-town" requests to be automatically handled by providing file access to the data base of the appropriate region.

The application program can be written so that it is independent of the actual location of the data, and can run in any of the regional centers. An example of this type of application is the validation of credit against customer accounts.

For example, if two systems each have a dataset called ACCT, containing data for their area, a single application program for execution in both systems can be written to handle inquiries against its local ACCT file and the remote ACCT file. To do this, each system has

an FCT entry for the file named "ACCTR", specifying that this file is located on the other system, and is known there by the name "ACCT". Each system also has a normal FCT entry for the file "ACCT".

With the following logic, the application program may first search its own file for the required record, and if not found, it can then search the "ACCT" file on the connected system.

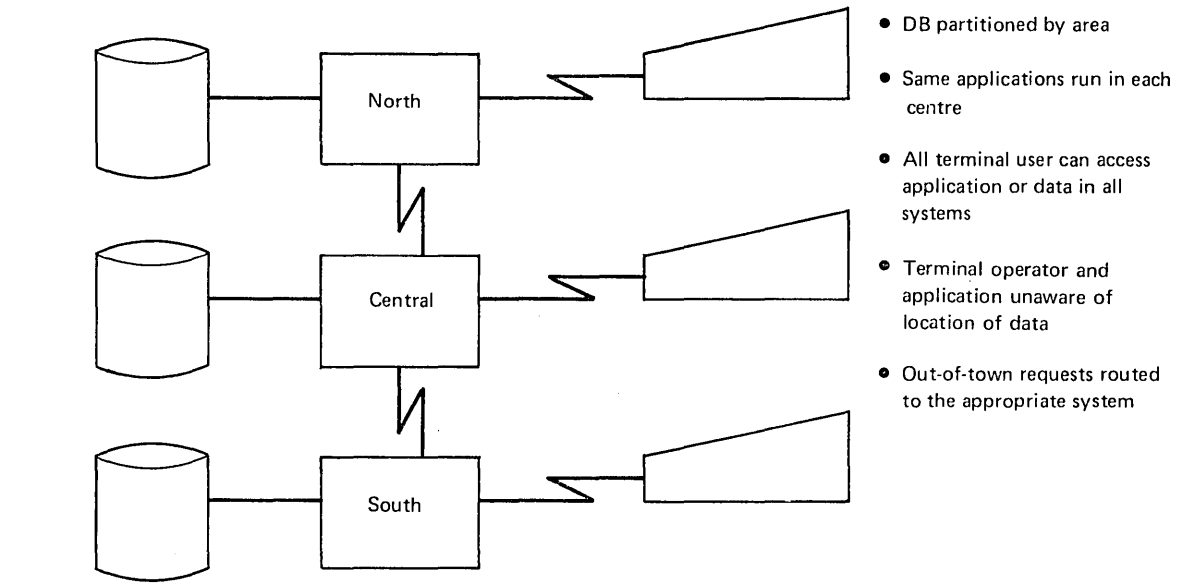
```
LCLFILE
                    /*PREPARE FOR RECORD NOT FOUND ON LOCAL SYSTEM*/
          EXEC CICS HANDLE CONDITION NOTFND (RMTFILE)
          EXEC CICS READ DATASET('ACCT')
          GO TO PROCESS

RMTFILE
                    /*PREPARE FOR RECORD NOT FOUND ON REMOTE SYSTEM*/
          EXEC CICS HANDLE CONDITION NOTFND (UNKNOWN)
          EXEC CICS READ DATASET('ACCTR')
          .
          .
          .
UNKNOWN
                    /*RECORD NOT FOUND ON LOCAL OR REMOTE SYSTEM*/
```

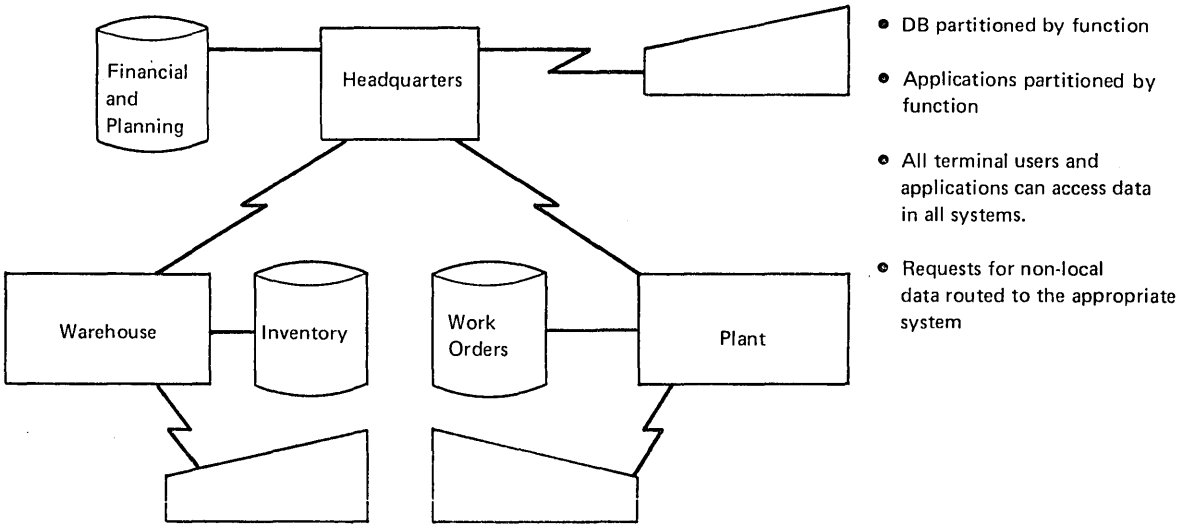
CONNECTING DIVISIONS WITHIN AN ORGANIZATION

Some users are organized divisionally, with separate systems, terminals, and data bases for each division: for example, Engineering, Production and Warehouse divisions. Connecting these divisions to each other and to the headquarters location improves access to programs and data, and thus may improve the coordination of the enterprise.

The applications and data may be hierarchically organized, with summary and central data at the headquarters site and detail data at plant sites. Alternatively, the applications and data may be distributed across the divisional locations, with planning and financial data and applications at the headquarters site, manufacturing data and applications at the plant site, and inventory data and applications at the distribution site. In either case applications at any site can access data from any other site, as necessary, or request applications to be run at a remote site (containing the appropriate data) with the replies routed back to the requesting site when ready.

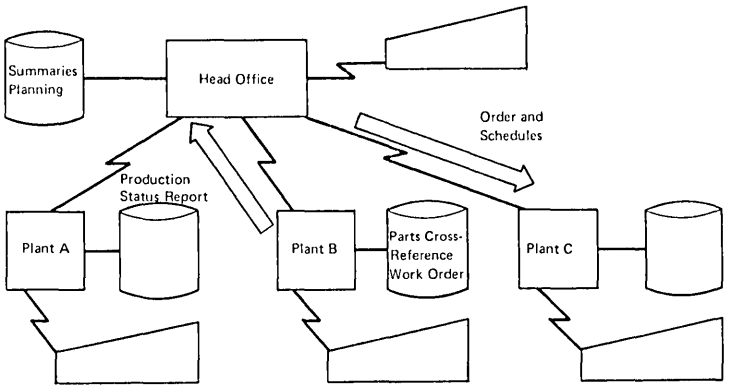


Connecting regional centres



Connecting divisions: distributed applications and data

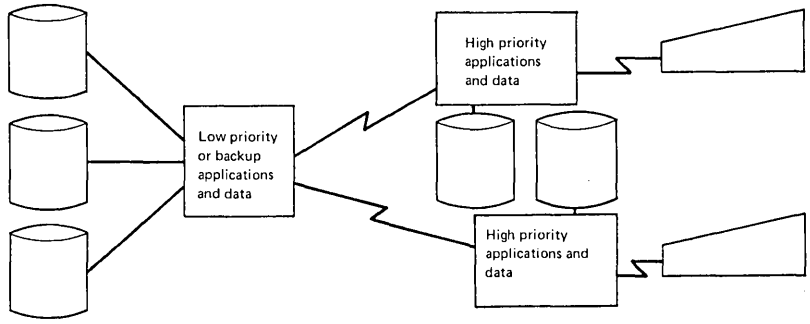
Figure 7.2-1. (Part 1 of 2) Possible Application Configurations



Hierarchical division of DB

- Summaries and central data at HQ, detail data at plant location
- Order processing at HQ: orders and shedules transmitted to plants of production status
- Plants send summaries of production status to HQ (for example, overnight)
- Access to data from HQ or Plant possible if required.

Connecting divisions: hierarchical distribution of data and applications



Connecting divisions: hierarchical distribution of data and application

- Improved response through distributed processing

Figure 7.2-1. (Part 2 of 2) Possible Application Configurations

Design Considerations

User application programs may run in a CICS/VS intercommunication environment and make use of the intercommunication facilities without being aware of the location of the file or other resource being accessed. The location of the resource is defined by the system programmer in the appropriate CICS/VS table: DCT for transient data; FCT for file control; PSB directory list (PDIR for OS/VS and DLZACT for VSE) for DL/I; TST for temporary storage; and PCT for interval control. (Details of these tables are in the CICS/VS System Programmer's Reference Manual.) The entry may also specify the name of the resource as it is known on the remote system, if that is different from the name by which it is known locally. When the resource is requested by its local name, CICS/VS substitutes the remote name before sending the request. This facility is useful when a particular resource exists with the same name on more than one system but contains data peculiar to the system on which it is located.

An operand, namely SYSID, is also available to allow the application program to select explicitly the system on which a particular command is to be executed. This optional parameter allows requests to be routed to a remote system without defining that remote resource in the local CICS/VS system. If this optional parameter is coded the resource definition tables on the local system are not used.

The local system can also be specified in the SYSID operand, so the decision whether to access a local resource or a remote one can be taken at execution time.

FILE CONTROL

Intercommunication allows access to BDAM, ISAM or VSAM files located on a remote system. Segmented records, and OPEN or CLOSE are not supported. Both inquiry and update requests are allowed, and the files may be defined as protected in the system on which they reside. Updates to remote protected files will not be committed until the application program issues a sync point request or terminates successfully. Linked updates of local and remote files may be performed within the same logical unit of work, even if the remote files are located on more than one connected CICS/VS system.

Caution is needed when designing systems where remote file requests using physical record identifier values are employed, such as BDAM, VSAM RBA, or files with keys not embedded in the record, because of the need to ensure that all application programs in remote systems have access to the correct values following addition of records or reorganization of these types of files.

DL/I

Intercommunication allows access to DL/I DOS/VS or IMS/VS DB data associated with a remote CICS/VS system. A CICS/DOS/VS transaction may access data from an IMS/VS DB data base associated with a CICS/OS/VS system running in another processor system. Remote access to DL/I DOS/VS ENTRY is not supported.

As with File Control, updates to remote DL/I data bases are not committed until the application reaches a sync point. It is not possible to schedule more than one PSB per logical unit of work, even when both PSBs are defined to be on remote systems. Hence linked DL/I updates on different systems cannot be made in a single logical unit of work.

The PSB directory list (PDIR or DLZACT) is used to define a PSB as being on a remote system. The remote system owns the data base and the associated PCB definitions. When DL/I access requests are made to another processor system by a CICS/OS/VS system but no local requests are made, it is not necessary to install IMS/VS DB on the requesting system. For CICS/DOS/VS it is necessary to install DL/I DOS/VS, even if there is no DL/I database associated with the requesting system. This is because the decision to handle the request in a local or remote system is contained within DL/I DOS/VS.

Interval Control can be used to schedule transactions in remote connected systems. The connected system may be any one that implements a suitable subset of SNA LU Type 6 protocols. As well as CICS/VS, IMS/VS supports a suitable protocol.

Entries for transactions located on other systems are made in the Program Control Table (PCT). The CICS/VS START command allows a transaction name, and optionally a terminal name, to be specified on the request. Thus a transaction can be initiated in a remote system with or without specifying an associated terminal. Additional operands on the START command are available to:

- Name a Temporary Storage or Transient Data queue containing information previously set up for processing by the nominated transaction
- Name another transaction, to be optionally invoked by the remote transaction before completion
- Name a terminal to be associated with that transaction when it executes.
- Indicate that an SNA function management header is included in the data. This allows a data description to be passed with the data.

Use of these operands is optional. They may be used when initiating local as well as remote transactions. There is also an operand that improves the performance of intersystem START requests at the expense of detecting certain types of error; it is described later under "NOCHECK Operand of START Command".

The names of the queue and the reply transaction specified in the additional operands must be defined in the system on which they are to be used.

With these operands it is possible for a terminal connected to the local system to initiate a transaction that in turn initiates a transaction in the remote system to run concurrently with other work initiated by the terminal in the local system. The application program at the local system issues a START request for the required remote transaction. The request specifies the identity of the invoking terminal, and the name of a reply transaction in the requesting system that is to be used to deliver the reply to the terminal. The requesting program then terminates, so releasing the terminal to enter other transactions.

In the remote system, multiple copies of the single transaction may execute concurrently as a result of requests from several other systems. On completion of its processing, the transaction sends the reply to the appropriate system via a START command naming the reply transaction and originating terminal. Each requesting system must specify a system unique name for its reply transaction, and the remote system must have all these names in its PCT.

If the data to be passed between the transactions exceeds that allowed by the START command, the data can be added to remote Transient Data or Temporary Storage queues by naming the queue in the START command queue name operand.

The PROTECT operand indicates that the remote transaction must not be scheduled until the local one has successfully completed a synchronization point. It can take the synchronization point by either

issuing a SYNCPOINT command or termination. If the synchronization point fails, the START request is backed out. If PROTECT were not specified, the results of a failed synchronization point would be undefined. This facility allows data intergrity across the link to be maintained. If the remote system is IMS/VS, no message must cross the link between the START command and the synchronization point.

A transaction initiated by a START command can access the operands by issuing the RETRIEVE command. A full discussion of the START and RETRIEVE commands may be found in the appropriate command level CICS/VS Application Programmer's Reference Manual.

If a transaction initiated by a START request is to be recoverable on failure of the remote system, the request identifier specified on the START command must be defined as recoverable in the Temporary Storage Table (TST) on the remote system.

Only the START and CANCEL commands are supported for function request shipping. The START command may specify a time of day at which the named transaction is to be scheduled. If the remote system is domain-remote and in a different time zone the requesting application program must take this into consideration, or use the interval form of time specification.

Local Queueing

When the local application is ready to ship a request, the intersystem facilities may be unavailable, either because the remote system is not active or because a connection cannot be established. An attempt to proceed with the request at that time would result in the SYSIDERR condition being raised. A solution is to store the request locally, and forward it when the link is back in service. The storing and forwarding may be carried out by user-written transactions or by the CICS/VS local queueing facility.

CICS/VS will provide local queueing for START commands intended to initiate transactions on domain-remote systems. The commands must include the NOCHECK operand, and local queueing must be specified by means of either a user exit invoked from the CICS/VS routine DFHISP or the LOCALQ operand of the DFHPCT TYPE=REMOTE macro for the remote transaction. The user exit can specify local queueing for all requests from the local system, and the LOCALQ operand can specify local queueing for all requests from the local system for the particular remote transaction defined in the DFHPCT TYPE=REMOTE macro. Details are given in the System Programmer's Reference Manual

If a required link is out of service when a program executes a START command with the NOCHECK operand, then if local queueing has been specified, the request will be added to the queue. CICS/VS will automatically begin forwarding the queue when the link returns to service.

Applications may be designed to communicate with remote systems on a store-and-forward basis. For instance, requests may be accumulated during the day and forwarded at night. A user-written transaction may be used to forward requests from a user-created queue. Alternatively, if the requests can be made in the form of START commands, and if the error checking suppressed by the NOCHECK operand is not required, then the CICS/VS local queueing facilities may be used instead.

The CICS/VS NOINTLOG terminal status may be used on the sessions across the link to control when requests are forwarded. For instance,

if the NOINTLOG status is specified on the TRMSTAT operand of the DFHTCT TYPE=SYSTEM macro defining the link, then CICS/VS will not attempt to open sessions across the link in response to function shipping requests. START requests will then be queued. When they are to be forwarded, the master terminal operator can acquire sessions or set the status to INTLOG, at which time CICS/VS will begin opening sessions and shipping the requests.

TEMPORARY STORAGE

Intercommunication enables application programs to send data to, or retrieve data from, Temporary Storage queues located on remote systems. A Temporary Storage queue is specified as being remote by means of an entry in the local TST. If the queue is to be protected, its queue name (or remote name) must also be defined as recoverable in the TST of the remote system. The queue on the remote system may be defined as either an auxiliary or main storage queue.

TRANSIENT DATA

An application program can access intrapartition or extrapartition transient data queues on remote systems. The Destination Control Table (DCT) in the requesting system defines the named queue as being on the remote system. The DCT entry for the queue in the remote system specifies whether the queue is protected, and whether it has a trigger level and associated terminal. Extrapartition queues may be defined (in the owning system) as having fixed, variable, or undefined length records.

Many of the uses currently made of transient data and temporary storage queues in a stand-alone CICS/VS system can be extended to an interconnected processor system environment. For example, a queue of records can be created in a system for processing overnight. Queues also provide another means of handling requests from other systems while freeing the terminal for other requests. The reply can be returned to the terminal as soon as it is ready, and delivered to the operator when there is a lull in entering transactions.

If a transient data destination has an associated transaction, the named transaction must be defined to execute in the system owning the queue; it may not be defined as remote. If there is a terminal associated with the transaction, it may be connected to another CICS/VS system within the same processor system and used via the transaction routing facility of CICS/VS.

The remote naming capability enables a program to send data to the CICS/VS service destinations, such as CSMT, in both local and remote systems.

TRANSACTION ROUTING

A terminal operator signed on to a terminal connected to one CICS/VS system can a enter a transaction code for a transaction owned by a connected CICS/VS system, and have CICS/VS route the request to the owning system. The transaction will then run exactly as if the terminal were attached to the transaction-owning system.

CICS/VS will route information between the terminal and the transaction by means of a relay transaction running in the system to which the terminal is connected.

To communicate with the terminal, the application program may use the terminal control, BMS or batch data interchange facilities of CICS/VS. Mapping and data interchange functions are performed in the system running the user's transaction. BMS paging operations will be performed on the terminal-owning system. BMS routing may be used subject to restrictions described in the System Programmer's Reference Manual.

Both conversations and pseudo-conversational transactions are supported. The various transactions that make up a pseudo-conversational transaction may run on different systems.

CICS/VS supports automatic transaction initiation between connected systems: the initiator transaction, the initiated transaction and the associated terminal (if any) may all be in different systems.

Applications written for single-region CICS/VS systems may make use of transaction routing without, in most cases, any reprogramming. System errors associated with cross-regional traffic will cause the application to abend.

A user's transaction can be in session with only one relay transaction at a time (since a transaction can converse with only one principal terminal). But it can be in session with several mirror transactions as well as a relay transaction (it may have several function shipping requests outstanding). A mirror transaction may be in the same CICS/VS system as the relay or a different one; in the former case, the user's transaction will be using two simultaneous sessions between the two systems. A relay transaction session, which results from a transaction routing request, must be within a processor system, but a mirror transaction session, which results from a function shipping requests, may be within a processor system or across processor systems.

Transaction routing requests may be routed through more than one connected CICS/VS system, if the transaction requested by the terminal is not available on the first system to which the request is routed. This 'daisy chaining' of requests is not recommended, however, because of difficulties in recovering from errors (see Chapter 7.5).

Automatic Transaction Initiation

There are two ways in which transactions may be initiated automatically:

> by the transient data program, when the number of items on a queue reaches the trigger level;
>
> by the interval control program, when the time specified on the initiating request has expired.

In either case, the transaction may require CICS/VS to connect a particular terminal to it. The terminal may be connected to another CICS/VS system within the same processor system. If the two CICS/VS systems are connected to each other by MRO, CICS/VS will start a remote scheduler transaction in the other system, and route the connection request to it. This transaction issues an automatic transaction initiation request for the relay transaction to run on the required terminal. If the terminal is not available (it may be connected to another transaction), CICS/VS will wait until it is free before starting the relay transaction. When the relay transaction has been started and connected to the terminal, CICS/VS starts the user's transaction and routes information between it and the terminal via the relay transaction, in the same way as when the transaction is initiated from a terminal.

Basic Mapping Support

The mapping operations of BMS are performed in the system on which the user's transaction is running. The mapped information is routed between the terminal and this transaction via the relay transaction, as for terminal control operations.

For BMS page building and routing requests, the pages are built and stored in the user transaction's system. When the logical message is complete the pages are shipped to the terminal owning region (or possibly regions if they were generated by a routing request) and deleted from the user transaction system. Page retrieval requests are processed by a BMS program running in the system to which the terminal is connected.

THE MIRROR TRANSACTION

The mirror transaction is supplied by CICS/VS to reissue function shipping requests transmitted to it by other systems. It executes as a normal CICS/VS transaction and uses the CICS/VS terminal control program facilities to communicate with the requesting system. Its transaction identifier is CSMI. In the requesting system, the command level EXEC interface program (for all except DL/I requests) determines that the requested resource is on another system, formats the request for transmission, and calls on the intercommunication component to send the request to the appropriate connected system.

The intercommunication component uses CICS/VS terminal control program facilities to send the request to the mirror transaction. The first request to a particular remote system on behalf of a transaction will cause the communication component in the local system to precede the formatted request with the mirror transaction identifier, in order to attach this transaction in the remote system. Thereafter it keeps

track of whether or not the mirror transaction terminates, and reinvokes it as required.

The mirror transaction decodes the formatted request and executes the corresponding command. At completion of the command the mirror transaction constructs a formatted reply and returns this to the requesting system. On that system the reply is decoded and used to complete the original command level request made by the application program.

If the mirror transaction is not required to update any protected resources, and no previous request updated a protected resource in its system, the mirror transaction will terminate after sending its reply. However, if the request causes the mirror transaction to change or update a protected resource, or the request is for any DL/I PSB, it will not terminate until the requesting application program issues a synchronization point request or terminates successfully. When the application program issues a synchronization point request, or terminates successfully, the intercommuniction component sends a message to the mirror transaction which causes it also to issue a synchronization point request and terminate. The successful synchronization point by the mirror transaction is indicated in a response sent back to the requesting system, which then completes its synchronization point processing, so committing changes to any protected resources. If DL/I requests have been from another system, CICS/VS issues a DL/I TERM call as a part of the processing resulting from a synchronization point request made by the application program and executed by the mirror transaction.

The application program is not constrained in the order in which it accesses protected or unprotected resources, nor is it affected by the location of protected resources (they could all be in remote systems, for example). When the application program accesses resources in more than one remote system, the intercommunication component invokes a mirror transaction in each system to execute requests on behalf of the application program. Each mirror transaction follows the above rules for termination, and when the application program reaches a synchronization point, the intercommunication component exchanges synchronization point messages with those mirror transactions that have not yet terminated (if any). This is referred to as the multiple-mirror situation, and is illustrated schematically in Figure 7.2-2.

The mirror transaction uses the CICS/VS command level interface to execute CICS/VS requests and the DL/I CALL interface to execute DL/I requests. The request is thus processed as for any other transaction and the requested resource is located in the appropriate resource table. If its entry defines the resource as being remote, the mirror transaction's request is formatted for transmission and sent to yet another mirror transaction in the specified system. This situation is referred to as "chained-mirror." It is strongly recommended that the system designer avoids defining a connected system in which chained mirror requests will occur, except when the requests involved do not access protected resources, or are inquiry-only requests. Refer to Chapter 7.5, "Recovery and Restart" for further discussion of the effects of using chained mirror transactions.

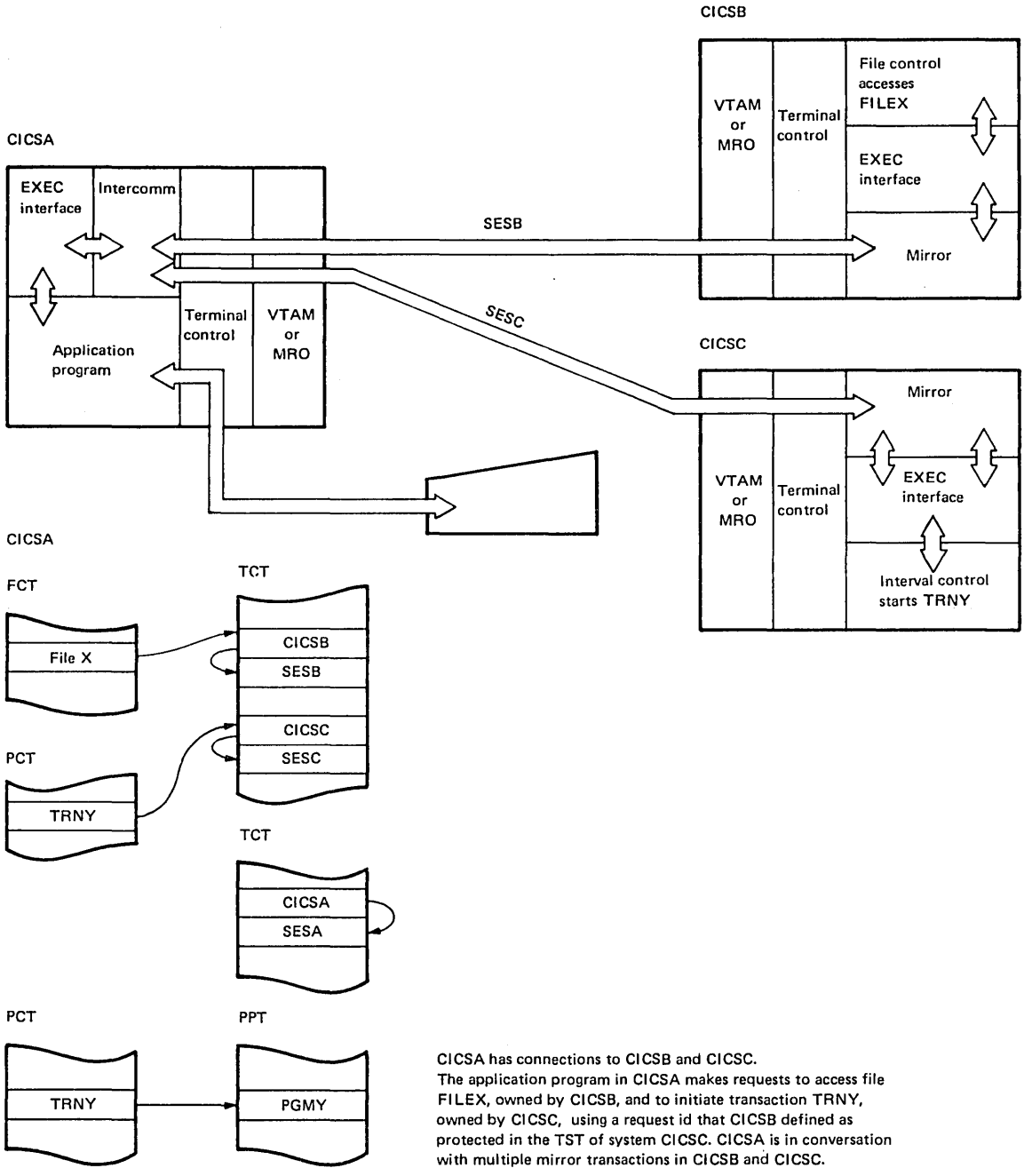


Figure 7.2-2. Multiple and Chained Transactions

THE RELAY PROGRAM

When a terminal operator enters a transaction code, CICS/VS checks the program control table and decides whether that transaction is owned by the system to which the terminal is connected or some other system. In the latter case, a transaction is started in the terminal-owning system and connected to the terminal. This transaction is frequently called the relay transaction, though in fact its name and attributes are those of the user's transaction in the remote system, as described by the system programmer in the DFHPCT TYPE=REMOTE macro for the terminal-owning system. The program that runs as this transaction is, however, the CICS/VS-supplied relay program. For the sake of simplicity, therefore, the transaction is often called the relay transaction rather than the relay program.

If an automatically initiated transaction requires a terminal that is connected to another system, a relay transaction is started in that other system. If and when the terminal is free, the relay transaction is connected to it.

In both cases (operator and automatic initiation), after the relay has been connected to the terminal, it sends a request to the other system to attach the user transaction. From then on, any request or output issued by the user transaction to its principal terminal is intercepted by the CICS/VS terminal control program and shipped to the relay transaction. The relay transaction then issues the request or output to the terminal. In a similar way, terminal status and input are shipped via the relay transaction to the user's transaction.

The relay transaction remains in existence for the life of the user's transaction and has exclusive use of the session to the remote system during this period. When the user's transaction terminates, an indication is sent to the relay transaction, which then terminates and frees the terminal.

When the user's application takes a synchronization point, then if it was defined as a protected task requiring committed output messages, an indication is sent to the relay transaction, which then takes its own sychronization point. Each of the two CICS/VS systems maintains its own log. Committed output messages are logged on the terminal owning system.

A user transaction initiated by a relay transaction may in turn initiate one or more mirror transactions. This is illustrated in Figure 7.2-3.

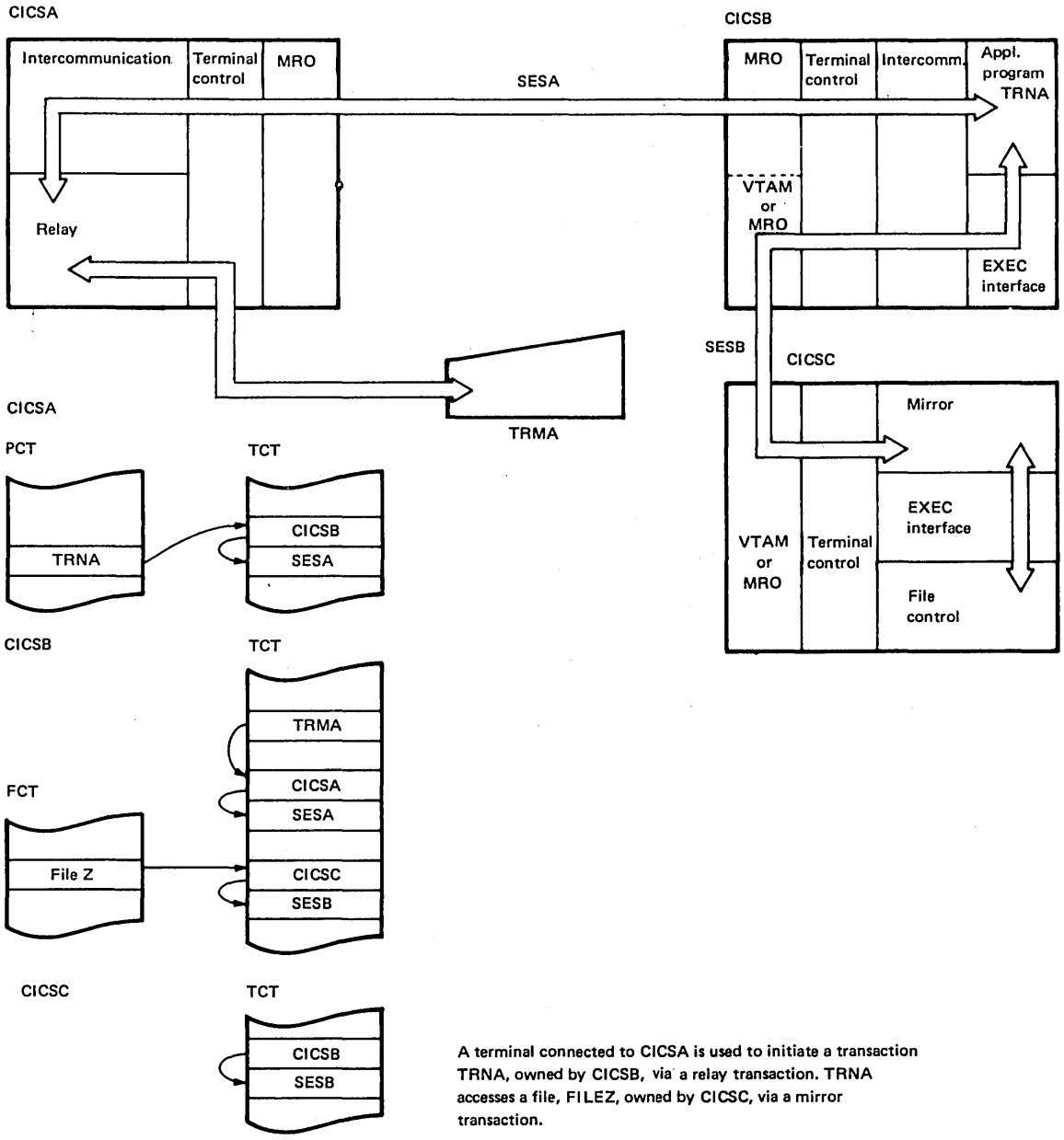


Figure 7.2-3. Relay and Mirror Transactions

The additional flexibility in operation that is afforded by the CICS/VS function shipping and transaction routing facilities must be offset against additional storage and processor usage that may be experienced when making remote requests. Details of the instruction overhead in the local processor of making any remote request are to be found in the CICS/VS System Programmer's Reference Manual.

Where existing transactions are made to access their files remotely or existing terminals made to access their transactions remotely, terminal operators may experience an increase in response time due directly to the additional elapsed time taken to transmit the request, attach the mirror or relay transaction, and transmit the reply. The actual delay will depend upon the volume of data to be accessed and the speed of the communication facility. In addition, the delay may be aggravated in particular instances; for example:

- Waiting for allocation of a session when the intersystem or interregion traffic exceeds the capacity of the available sessions;
- Waiting for sufficient resources in the remote system to schedule the mirror or user transaction (for example, when the remote system has scheduled its maximum allowed number of tasks).
- In the case of domain-remote requests, waiting while VTAM establishes the session between the systems on the first request that requires a particular system, when the session was not previously established;

The system designer must ensure that sufficient resources are available in a system to handle mirror and relay transactions required by intercommunication requests. These transaction must be given due consideration when defining system initialization table values for the "maximum number of tasks" (MXT) and "maximum number of active tasks" (AMXT) operands of the DFHSIT macro, described in the CICS/VS System Programmer's Reference Manual).

For CICS/OS/VS users, the instruction pathlength for the mirror transaction may be reduced by specifying a primed storage allocation by means of the PRMSIZE operand of the DFHPCT TYPE=ENTRY statement for the mirror. The task control area (TCA), journal control area (JCA), EXEC interface block (EIB), and any VSAM work area (VSWA), will be allocated from this area during execution of the mirror transaction. The exact allocation required may be found from the primed storage statistics collected for the mirror transaction. The allocation depends on the services requested by the mirror on behalf of remote transactions.

For domain-remote connections additional instruction pathlength reductions may be gained by MVS users through by use of the VTAM Authorized Path with the CICS/VS High Performance Option.

NOCHECK Operand of START Command

In many enquiry-only applications, and in many update applications where recoverable data is entirely contained within one system, sophisticated error-checking and recovery procedures are not justified. Where the transactions make enquiries only, the terminal operator can retry an operation if no reply is received within a certain time. In such a situation, the number of messages to and from the remote system can be substantially reduced by means of the NOCHECK operand of the START command. Where the connection between the two systems is via VTAM, this can result in considerably improved performance. The price paid for better performance is the inability of CICS/VS to detect certain types of error in the START command.

A typical use for the START NOCHECK command is to make enquiries of a data base that spans two or more CICS/VS systems. The transaction attached as a result of the terminal operator's enquiry has to analyse the information supplied by the operator, and decide where to route the request. It then issues an appropriate START command with the NOCHECK option, which causes a single message to be sent to the appropriate remote system to start, asynchronously, a transaction that makes the enquiry. The command should specify the operator's terminal identifier. The transaction attached to the operator's terminal now terminates, leaving the terminal available for either receiving the answer or initiating another request. If the operator is to issue further requests, the application program must make all answers self-identifying, since the operator may receive them in random order.

The remote system performs the requested enquiry on its local data base, then issues a START command for the originating system. This command would pass back the requested data, together with the operator's terminal identifier. Again, only one message passes between the two systems. The transaction that is then started in the originating system must format the data and display it at the operator's terminal.

An example of intercommunication using the NOCHECK option is given at the end of this chapter (Example 6).

If a system or session fails, the terminal operator must re-enter his enquiry, and be prepared to receive duplicate replies. To aid him, either a correlation field should be shipped with each request, or all replies should be self-describing.

It is not possible to carry out simultaneous linked updates to data split between two systems, since the transaction that is the subject of the START command is executed asynchronously with the one issuing the command.

The NOCHECK operand is always required when shipping of the START command is queued pending the establishment of links with the remote system (see "Local Queuing" earlier in this chapter), or if the request is being shipped to IMS/VS.

Application Programming Considerations

CICS/VS application programs can use the function request shipping facilities only through the CICS/VS Command Level Interface or through DL/I CALLS. Transaction routing is possible for transactions that use the macro or command level interface. Application may be coded so that they are unaware of the location of the resources or terminals being accessed. This common application program interface minimizes the difficulties of developing applications in connected systems by placing the burden of system definition and resource location on the system programmer. The command level interface is available for programs written in PL/I, COBOL, RPG II, or assembler language. Since the program is unaware of location or means of connection, data resources, terminals and programs may be moved between systems without impact to the application programmer.

Optional extensions of function request shipping (the SYSID operand) allow the application program to select a specific system on which to execute a request. In this case the resource definition tables on the local system are not referenced in order to make the request. If in this case the request refers to transient data destinations or files defined as having fixed length record format, the application program must supply the record length as part of the request; this information is required by the local CICS/VS system in order to transmit the record correctly in the case of a WRITEQ or WRITE request. The capability to specify a specific system is not available with DL/I.

The START and RETRIEVE command parameters for "reply transaction identifier," "reply terminal identifier," and "processing queue name" are particularly useful in writing application programs to be initiated in remote systems as a means of distributing function among connected CICS/VS systems.

A new error condition may be experienced by application programs that ship function requests to remote systems. It indicates that the resource is defined as remote, but the local system is unable to ship the request because the remote system cannot be contacted (for example, it is temporarily out of service) or the remote system is not defined to CICS/VS. An application program that does not contain code to explicitly handle this condition will be abended if it occurs. Details of the return code are in the appropriate command level CICS/VS Application Programmer's Reference Manual.

An application request against a remote resource may cause an abend in the mirror transaction (for example, the requested Transient Data destination has been disabled by the remote CICS/VS master terminal operator). Similarly, an error may occur which abends the relay transaction handling transaction routing in a system remote from the application program. In these situations, the application program will also be abended, but with an abend code of ATNI (for domain-remote systems) or AZI6 (for region-remote systems). The actual error condition will be logged by CICS/VS in an error message sent to the CSMT destination. Any HANDLE ABEND command issued by the application will not be able to identify the original cause of the condition and take explicit corrective action (which may have been possible if the resource was local). (An exception is the mirror transaction in a region-remote system abending with a DL/1 program isolation deadlock; in this case, the application will abend with the normal deadlock abend code.)

The START command is supplied with an operand, NOCHECK, that can reduce the traffic between systems connected via VTAM. It indicates to CICS/VS that the result of checking the validity of the request in the remote system will not be returned to the request system; an error in the START command could therefore cause the remote transaction to abend

without the program that issued the command being notified. The facility is described earlier under "NOCHECK Operand of START Command".

The NOCHECK operand is required if the remote system is IMS/VS, or if the START request is to be queued pending the establishment of links with the remote system (see "Local Queueing Option" earlier in this chapter).

System Programming Considerations

The following is a summary of the points to be considered when generating a CICS/VS system that is to use function request shipping or transaction routing. (In the latter case, the system programmer is concerned with the terminal control and program control tables only). Details of generation macros and system requirements are in the CICS/VS System Programmer's Reference Manual or CICS/VS System Programmer's Guide respectively.

There are certain restrictions that apply when designing a system using transaction routing and function request shipping; these restrictions are listed in the System Programmer's Reference Manual.

TERMINAL CONTROL TABLE

For each system to which CICS/VS is to ship function requests or to distribute transaction processing or to route transactions, there is a system entry in the TCT (the TCTSE). Associated with this is a terminal control table terminal entry (TCTTE) for each session with the other system. The TCTSE is generated by the DFHTCT TYPE=SYSTEM macro (or TYPE=ISLINK, which is exactly equivalent), and the TCTTE by either of these or by the DFHTCT TYPE=TERMINAL statement.

The local system also has a TCTSE. This allows local resources to be accessed using the SYSID operand of application programmer commands. The program can therefore use the same command to access both local and remote resources.

The ACCMETH operand specifies whether the communication is to be via the interregion communication facility (ACCMETH=IRC) or via VTAM (ACCMETH=VTAM). The former is used when the other CICS/VS system is in the same processor system, the latter when it is in another processor system, or when it is in the same processor system and the VTAM application-to-application facility is to be used.

The SYSIDNT operand identifies the system entry, and is the name used in the other tables, such as the FCT and PCT, when defining resources as being on this particular remote system.

For communication via VTAM, the TRMIDNT operand provides an identity for an associated TCTTE, which allows a specific session to be referenced in CICS/VS master terminal commands that do not refer to the remote system as a whole. For communication via IRC, the names of TCTTEs are generated from the parameters supplied on the SEND and RECEIVE operands (described below).

The name of the logical unit representing the other system must be specified in the NETNAME operand, unless it is equal to the SYSIDNT value. This name is the same as that in the APPLID operand of DFHTCT

TYPE=INITIAL or DFHSIT TYPE=CSECT statements for the remote CICS/VS system.

Where there are multiple parallel VTAM sessions with a particular remote system, the system programmer may code one DFHTCT TYPE=TERMINAL statement for each session. Alternatively, where the attributes of the VTAM sessions are to be all the same, he may use the SEND or RECEIVE operands of the DFHTCT TYPE=SYSTEM statement. With IRC sessions, he must use SEND or RECEIVE. These specify the maximum numbers of sessions that a system may initiate and have initiated with it by other systems. They cause multiple TCTTEs to be generated without the system programmer having to code multiple DFHTCT TYPE=TERMINAL statements. They allow two-character prefixes for the TRMIDNT names of the generated TCTTEs to be specified together with a range of numbers, each number being used in character form as the other two characters of the TRMIDNT name. The number of sessions specified in a SEND operand and in the corresponding RECEIVE operand on the remote system must be the same.

A number of other operands, normally associated with the DFHTCT TYPE=TERMINAL statement, may also be specified with the DFHTCT TYPE=SYSTEM statement to define specific CICS/VS processing options to be applied to the session with the remote system. These include operating priority (OPERPRI) and security (OPERSEC). These values should be chosen bearing in mind that mirror and routed transactions may operate on this session to handle requests from the remote system, and so must have the appropriate authority to access resources in this system. The processing priority of the mirror transaction or a routed transaction is established by means of the TRMPRTY operand.

Two operands apply solely to START commands shipped to a remote system, both concerned with the intersystem data stream. RECFM indicates whether the data format follows the SNA VLVB specification, and DATASTR indicates whether the contents of the data stream are in SNA Character Set (SCS), 3270 data stream, Structured Field, Logical Mapping or user-defined form.

Transaction Routing

To permit transaction routing, the system programmer must provide a DFHTCT TYPE=REMOTE macro for each terminal on a remote system that may be connected to transactions on the local system. The SYSIDNT operand specifies the name of the remote system. It is the name given in the SYSIDNT operand of the DFHTCT TYPE=SYSTEM macro. The TRMIDNT operand specifies a four-character name by which the terminal is to be known to the local system. The RMTNAME operand specifies the name by which the terminal is known to the system to which it is connected, if it is different from the TRMIDNT name. Other operands must be coded to describe the physical characteristics of the terminal and the way in which it is to be accessed.

Alternatively instead of coding a TYPE=REMOTE macro for each terminal, it is possible to group the definitions of the terminals used in the local system under a macro specifying TYPE=REGION, SYSIDNT=system-identification, thereby defining the terminals as belonging to the named system. Attributes of the terminals that are not required or are meaningless for a remotely owned terminal are ignored. A single set of terminal definitions may therefore be copied into the DFHTCT macros for the remote system and for the local one. This facility greatly simplifies the terminal definition requirements for an MRO system.

FILE CONTROL TABLE

Entries are added to the FCT to define remote files in the DFHFCT TYPE=REMOTE macro, which names the owning system, and optionally provides the name by which the file is known on that remote system. The remote system is specified by the same name as in the SYSIDNT operand of DFHTCT TYPE=SYSTEM. For files that have fixed-length record format, the LRECL operand must also be specified on this entry.

DESTINATION CONTROL TABLE

Entries to the DCT for remote Transient Data queues are made in the DFHDCT TYPE=REMOTE macro. As for fixed-length files, the record length for fixed-length record format queues must also be specified in the LENGTH operand. Such items as trigger level and associated transaction can be specified only in the queue definition macros of the remote system. It is not permitted to define a local transient data queue with an associated transaction defined to be in a remote system.

TEMPORARY STORAGE TABLE

A DFHTST TYPE=REMOTE macro is used to define a set of temporary storage queues in remote systems in a similar manner to that in which DFHFCT defines a file. The queues are defined in the owning (remote) system by means of the DFHTST TYPE=RECOVERY macro if they are recoverable. No definition is required in the remote CICS/VS system if the queues are not recoverable.

PROGRAM CONTROL TABLE

Remote transactions are defined by means of the SYSIDNT operand of the DFHPCT TYPE=REMOTE macro. If the data to be passed to a transaction is to be protected, the request identifiers specified on the requests must appear as protected entries in the TST of the system owning the transaction.

When transaction routing is to be used, the PCT of the system to which the terminal is connected must include an entry for each remote transaction that may be invoked via the relay transaction. The transactions are specified in the DFHPCT TYPE=REMOTE macro. This macro specifies the name of the remote transaction and the name of the system on which it is to be executed. In addition, it describes the attributes of the relay transaction that is to control communication between the terminal and the user transaction; these attributes are usually the same as those specified for the user transaction in the remote system.

A program specification block (PSB) in CICS/OS/VS is defined as being remote by means of the SYSIDNT operand of the DFHDLPSB TYPE=ENTRY macro, which names the owning system. For DL/I under VSE, additional specifications on the DLZACT statement are required. The MXSSASZ operand of the DFHDLPSB TYPE=ENTRY statement is also required for remote PSB definition to inform CICS/VS of the maximum segment search argument that could be used with this PSB.

In CICS/DOS/VS, the DOS/VS DLI DLZACT macro defines local and remote PSBs.

When defining a system that may access domain-remote DL/I resources, care must be taken in the choice of the maximum request or reply message length value for the TIOAL operand of the DFHTCT TYPE=SYSTEM statement. This value must encompass the greatest of the maximum length request or reply that may be expected to be used with any of the remote PSBs on that system. This will be the greater of either an output request, or the reply to a schedule request. The reply to the schedule request consists of a header constructed by CICS/VS and a list of view descriptors corresponding to the PCBs associated with the requested PSB. The output request consists of a request header, an I/O area, and a list of segment search arguments. Its maximum length is the sum of the maximum possible length I/O area required by the PSB and the maximum number of SSAs of maximum SSA length. The maximum possible I/O area size for a particular PSB is passed to a requesting system by the owning system as a field within the reply to the schedule request.

If DL/I requests are to be made to a CICS/DOS/VS system running in another processor system, DL/I DOS/VS must be installed with that system, even if there are no local DL/I data bases. This is not the case for CICS/OS/VS.

GENERAL CONSIDERATIONS

If CICS/VS intercommunication facilities are to be used, the following macros have to be specified:

```
DFHSG PROGRAM=ISC
DFHPCT TYPE=GROUP,FN=ISC
DFHPPT TYPE=GROUP,FN=ISC
DFHSIT ISC=YES
DFHSG PROGRAM=EIP and related code
```

The DFHSG PROGRAM=TCP macro must include ACCMETH=VTAM, VTAMDEV=LUTYPE6 for a domain-remote connection or ACCMETH=IRC for a region-remote connection. The CICS/VS System Programmer's Reference Manual describes the PPT and PCT entries that must be specified for intercommunication. The system initialization table, DFHSIT, has an ISC operand to control loading of the intercommunication group. The APPLID parameter of DFHSIT must be specified if that operand was not included in the DFHTCT TYPE=INITIAL statement. The APPLID of the remote system must match the value in NETNAME or SYSIDNT in the DFHTCT TYPE=SYSTEM statement referring to that system. The two systems may support different levels of CICS/VS function, provided they both contain at least those functions that are to be used.

A transaction, program or map may exist on more than one system.

For users of OS/VS2 (MVS), the CICS/VS High Performance Option (HPO) may be specified with CICS/VS intercommunication facilities. When this option is used, specification of VTAM Authorized Path for connections between remote systems will reduce the pathlength for shipping requests from one system to another.

The CICS/VS System Programmer's Reference Manual contains information on how to generate the High Performance Option. CICS/VS files using VSAM fastpath facilities of HPO may be accessed by transactions in remote connected CICS/VS systems, but those transactions cannot open or close these files, so they cannot switch between fast path and normal mode access to the file. The CICS/VS High Performance Option is outlined in Part 1 of this manual.

The DFHPCT TYPE=ENTRY statements for any existing or new transactions that may request access to remote data resources should be examined, particularly to ensure that the DTB operand adequately specifies the backout options to be used in the event of a session failure during transaction sync point processing. The DTIMOUT (deadlock timeout) operand in DFHPCT TYPE=ENTRY provides a timeout value to guard against waiting too long for a remote request to be satisfied. The deadlock timeout is activated either when the transaction is waiting for the reply to a request on the mirror, or when it is waiting for allocation of the session in order to send a request. In the former case the deadlock timeout value will be exceeded when the mirror is enqueuing upon resources in a manner that deadlocks it with another transaction (which could be a mirror) that is also enqueuing on those resources in a different order. The value will also be exceeded if the definition of resources in remote systems has caused the mirror transaction to chain ultimately back to one in the requesting system. Note that such PCT operands as RTIMOUT and OPTGRP apply to the session between the transaction and the terminal, and not to any associated intercommunication sessions.

The DFHPCT TYPE=ENTRY statement for the mirror transaction may also have a DTIMOUT value to guard against excessive waiting due to deadlocks experienced by the mirror when it makes remote requests as in the case of chained mirror transactions. The RTIMOUT value for the mirror is not specified because it is usual for the mirror to wait for further requests from the application program when protected resources have been accessed. By monitoring CICS/VS statistics for the mirror transaction, a value may be assigned to PRMSIZE to meet the primed storage requirements of the transaction (CICS/OS/VS only). The stall purge operand, SPURGE, should take a value that best suits the operating characteristics of the system. The TPURGE and DTB operands of DFHPCT must always be specified as YES. The PROTECT option of OPTGRP is not required for the mirror transaction since all necessary logging is invoked through the synchnization point for CICS/VS intercommunication. INBFMH=ALL must be specified. Other PCT options for the mirror transaction may be used under the same criteria that govern selection of options for other transactions within an installation.

Certain CICS/VS management modules may be installed in the link-pack area (on OS/VS) or shared virtual area (on VSE). These can be shared by all the CICS/VS regions within a processor system. Using the LPA or SVA has the advantages of reducing paging and improving reliability (since the modules then cannot be overwritten). For information on installation see the CICS/VS System Programmer's Guide for your system.

CRTE ROUTING TRANSACTION

Before a transaction on a remote system can be invoked via the transaction routing facility, the system programmer must, in the ordinary way, update the program control table. He must use a DFHPCT TYPE=REMOTE macro to specify the remote transaction name and the name by which it is to be known locally. CICS/VS provides a transaction, CRTE, that allows a terminal operator to invoke a transaction on a remote system without the need for any table updates on the local system (though the terminal from which CRTE is invoked must have been defined on the remote system.) It may be used from any 3270 display device and, under CICS/DOS/VS only, the console.

To use CRTE, the terminal operator enters:

```
CRTE SYSID=xxxx
```

where xxxx is the name of the remote system, as specified in the SYSIDNT operand of the DFHTCT TYPE=SYSTEM macro. The transaction then indicates a routing session has been established, and the user enters input of the form:

```
yyyyzzzzzz...
```

where yyyy is the name by which the required remote transaction is known on the remote system, and zzzzzz... is the initial input to that transaction. Subsequently, the remote transaction may be used as if it had been defined locally and invoked in the ordinary way. All further input is directed to the remote system until the operator terminates the routing session by entering CANCEL.

The CRTE facility is particularly useful for invoking a master terminal transaction (CSMT or CEMT) on a particular remote system. It avoids the necessity of defining the remote CSMT or CEMT in the local PCT with a different name. CRTE is essential if EDF is to be used to debug a transaction that runs on a remote system, and is useful for testing transactions without using EDF and for occassional invocations of remote transactions that do not warrant updating the PCT.

STATISTICS

CICS/VS maintains two sets of statistics to help evaluate the operation of intercommunication facilities. The first set relate to usage of the available sessions and indicate the level of contention experienced for a session. The second set provides the number of remote requests made for each type of remote resource, and the number of transaction routing requests. Refer to the CICS/VS Operator's Guide for methods of statistics collection and the CICS/VS System Programmer's Guide for details of statistics output.

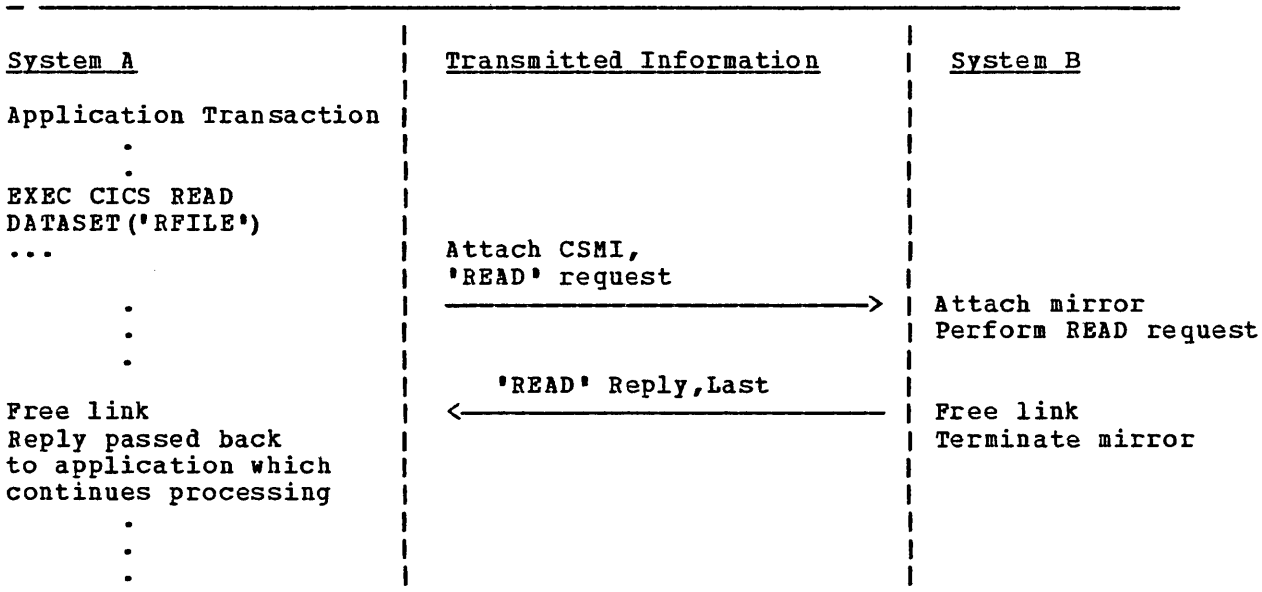
RECOVERY

When sessions between connected systems fail, or one of the connected systems fails, operator messages may be issued to indicate that the failure occurred during a critical point in the synchronization point process between the application program and its associated remote transaction. Upon restart of the system or the failing session, further messages will be issued to indicate whether or not a resource integrity exposure exists as a result of the failure. If the message specifies that there is an exposure, arrangements must be made to execute the appropriate reconcilliation procedure, as specified by the system designer. For further information, see Chapter 7.5.

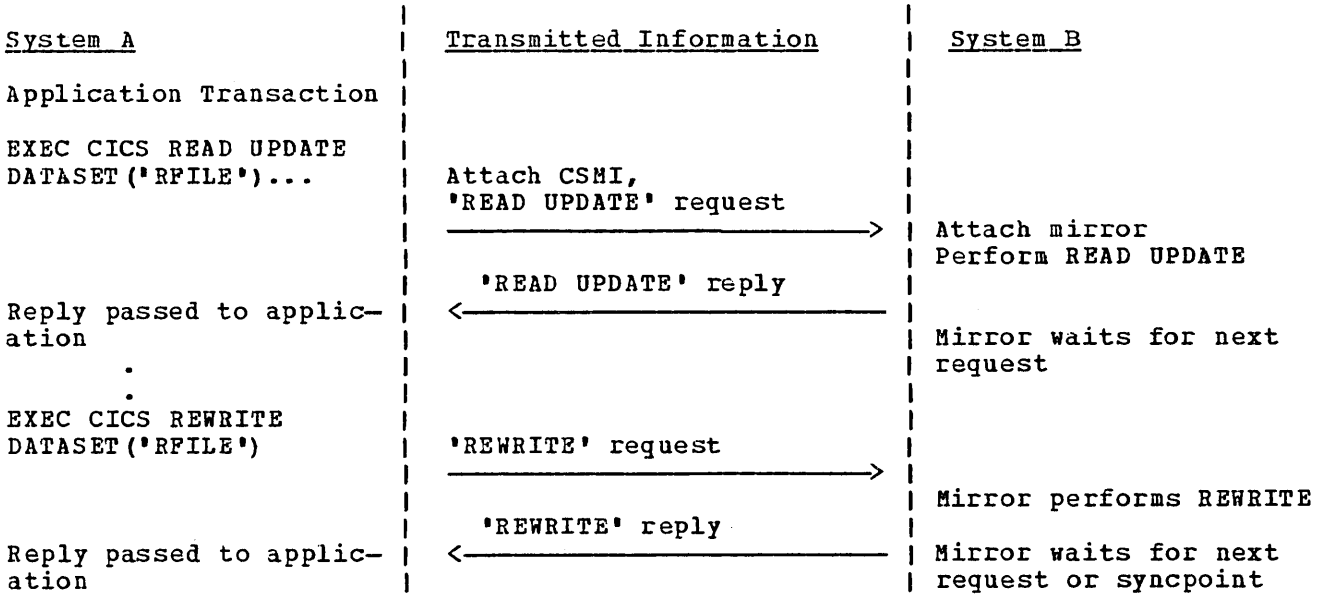
Function Request Shipping — Examples

This section gives some examples to illustrate the life time of the mirror transaction and the information flowing between the application and its mirror (CSMI). The examples contrast the action of the mirror transaction when accessing protected and non-protected resources on behalf of the application program. Example 5 illustrates a complete series of transactions and the points at which a session between two CICS/VS systems is made available for other requests. Example 6 shows the same thing, using the message performance option. Further details can be found in the CICS/VS Diagnosis Reference Manual.

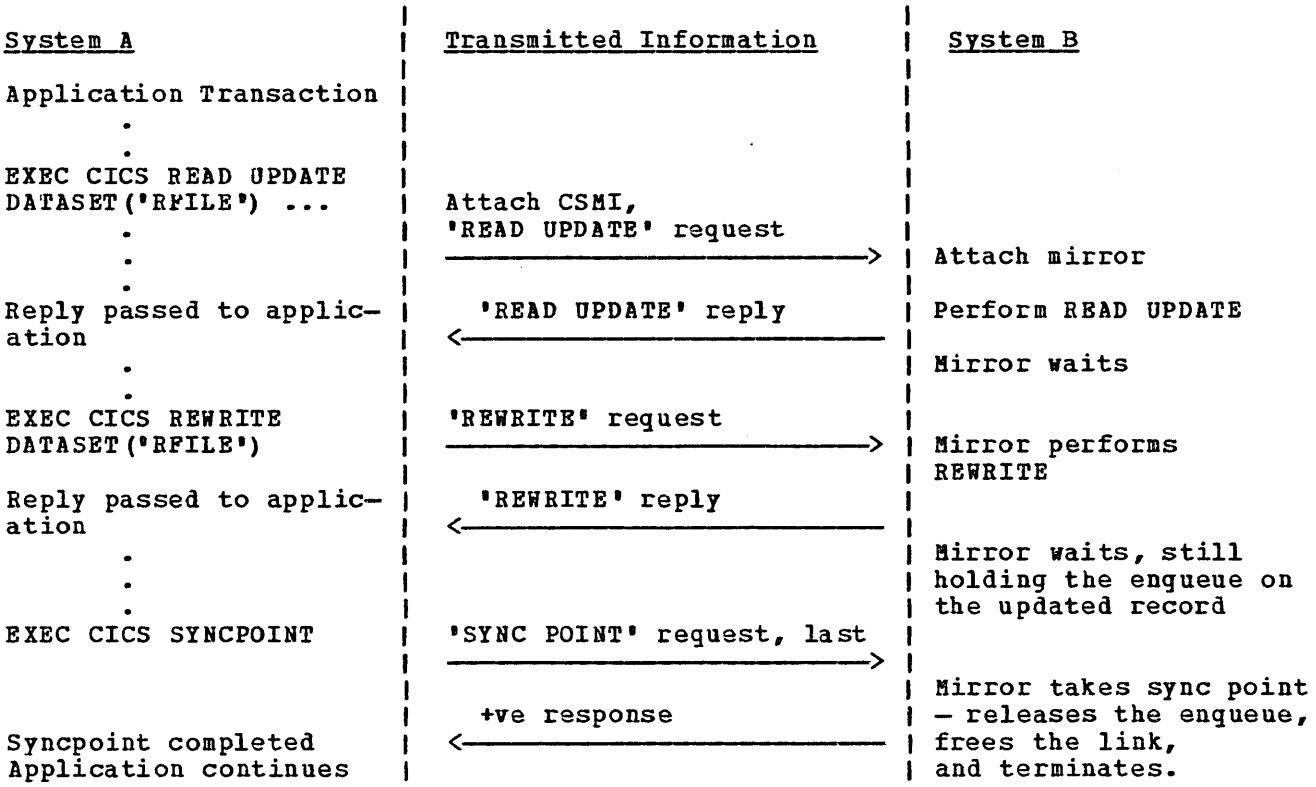
1. Simple Enquiry



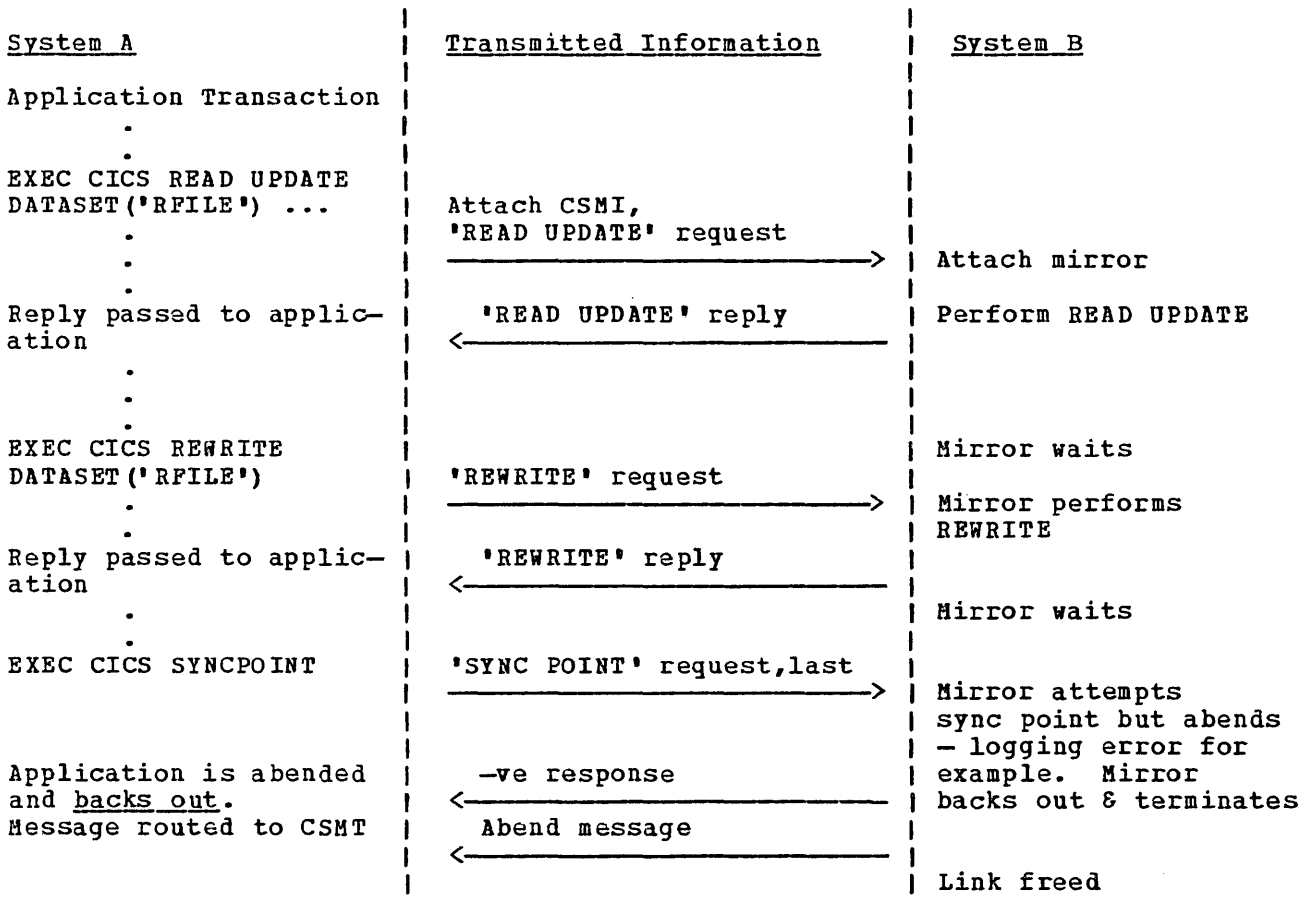
2. Remote Update (Unprotected)



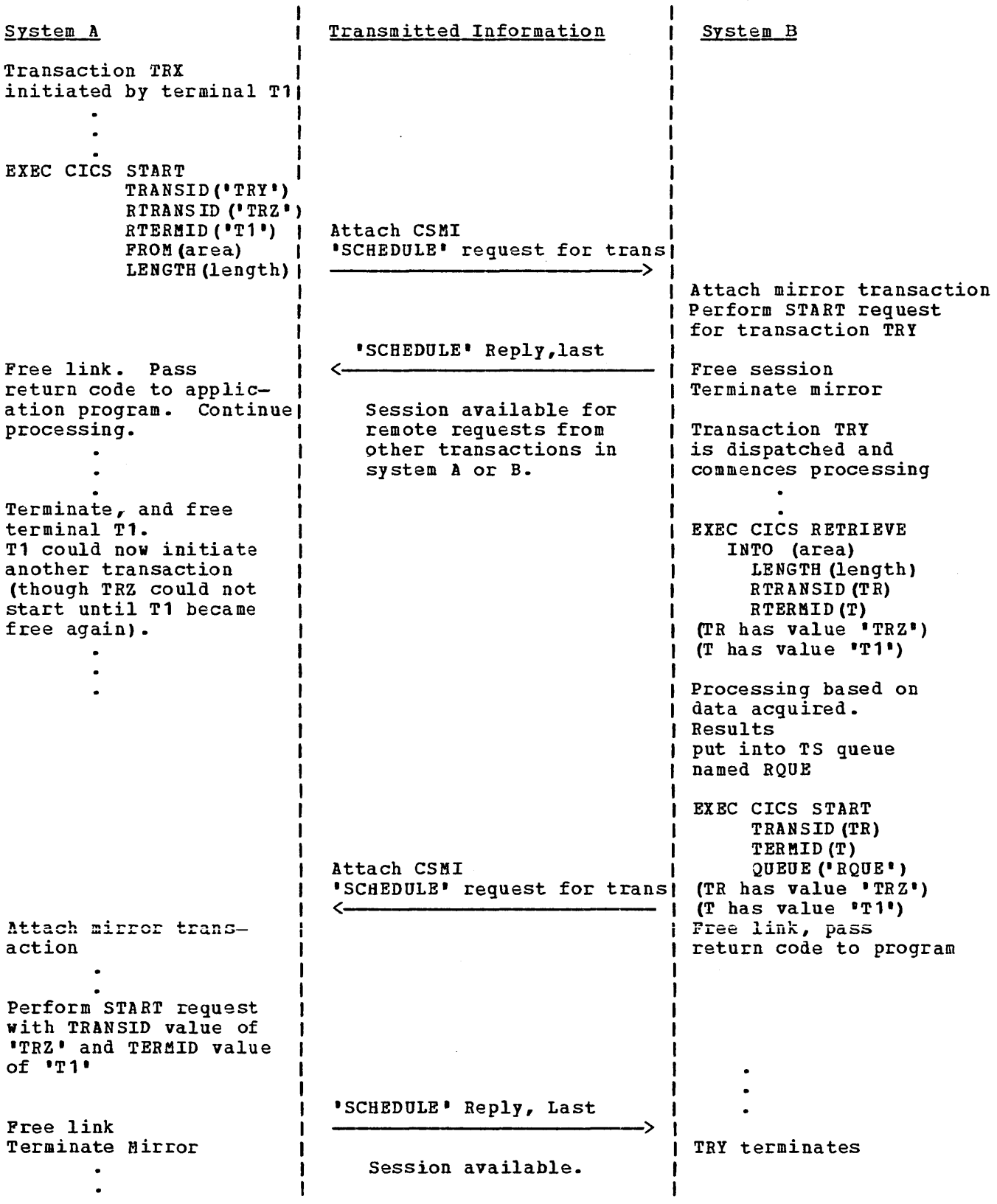
3. Remote Update (Protected)

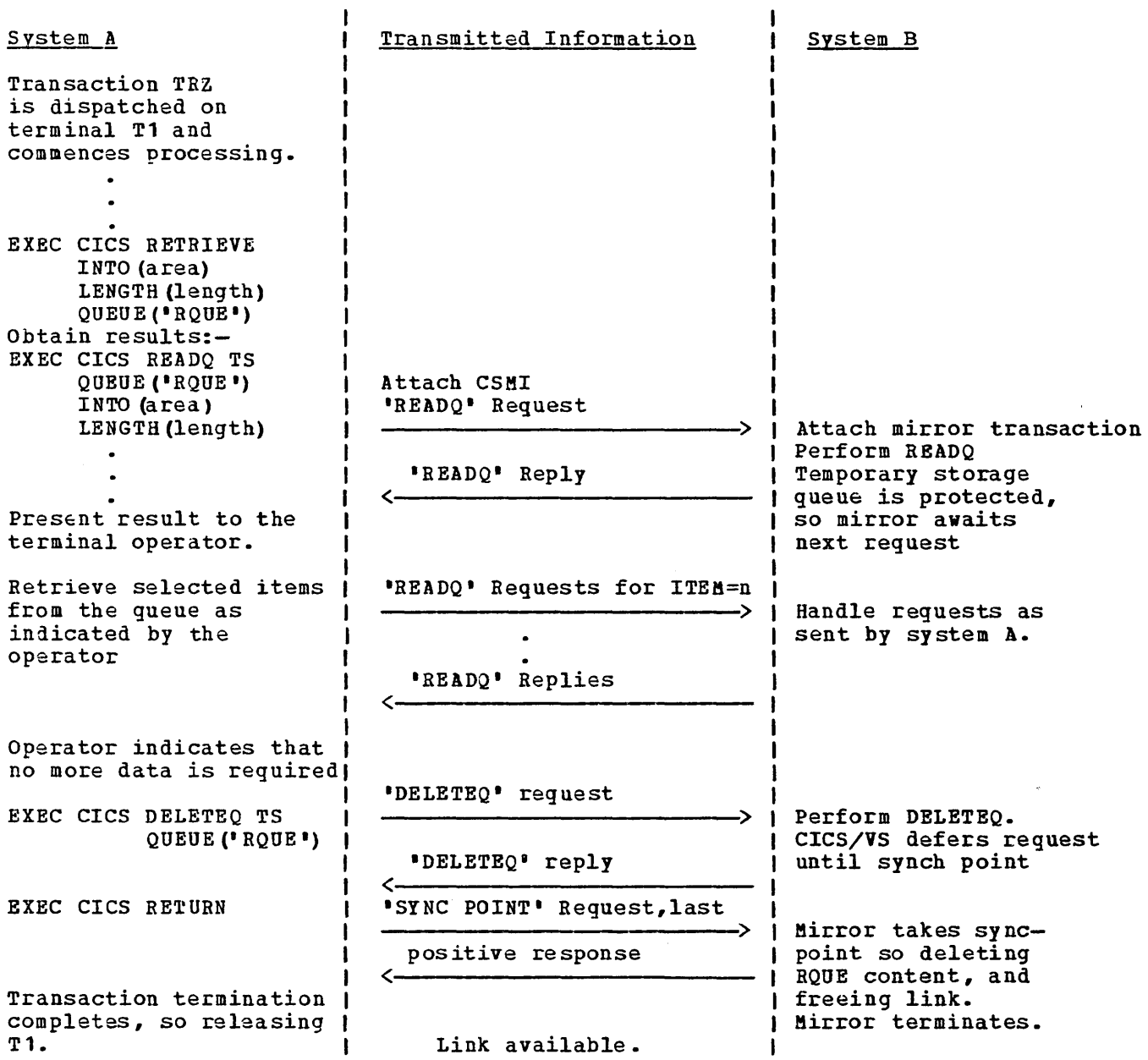


4. Remote Update (Protected) with ABEND

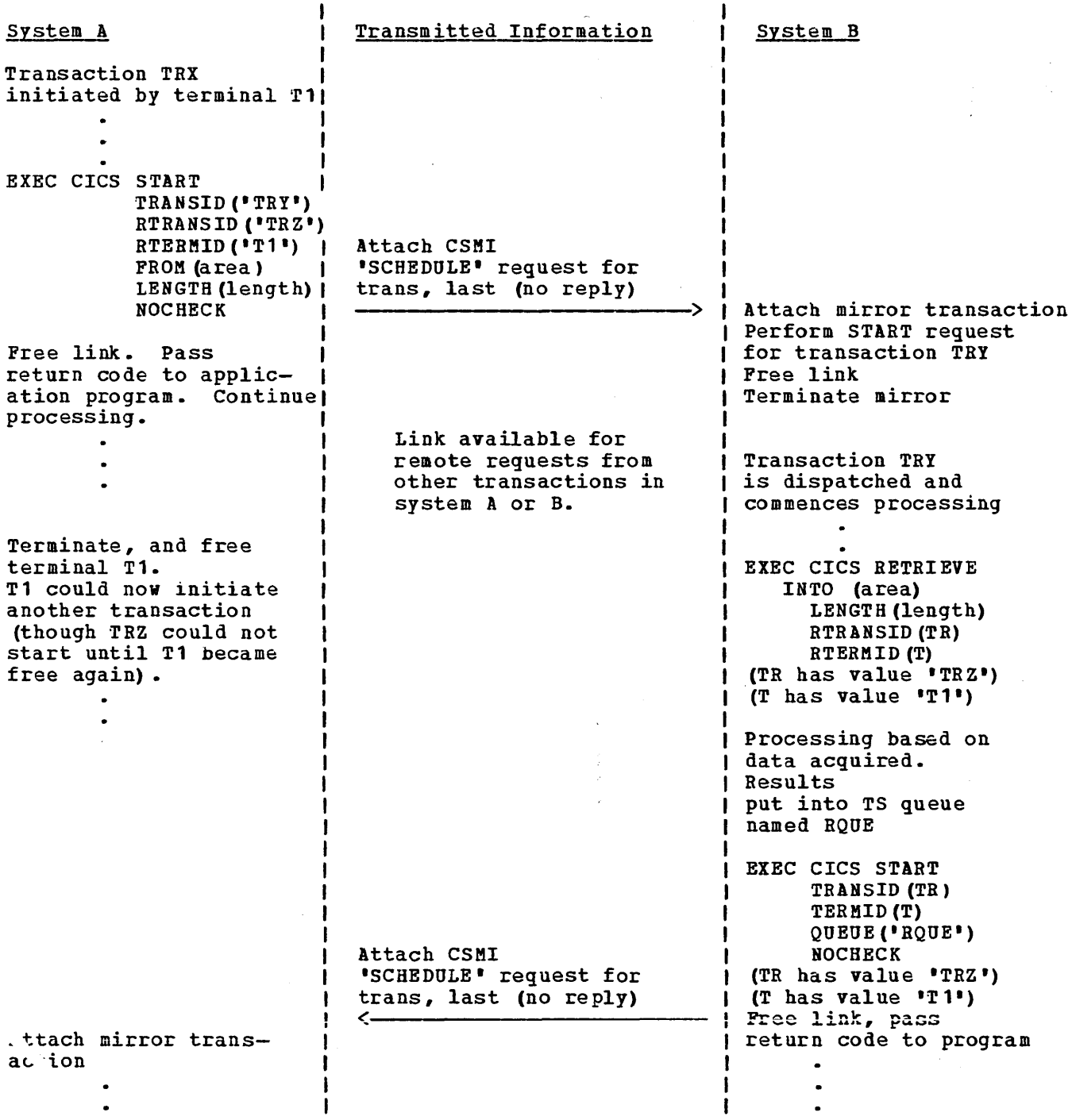


5. Remote Transaction Initiation




System A
Transmitted Information
System B
Transaction TRZ is dispatched on terminal T1 and commences processing.
EXEC CICS RETRIEVE INTO(area) LENGTH(length) QUEUE('RQUE')
Obtain results:-
EXEC CICS READQ TS QUEUE('RQUE') INTO(area) LENGTH(length)
Attach CSMI 'READQ' Request
Attach mirror transaction
Perform READQ
'READQ' Reply
Temporary storage queue is protected,
Present result to the terminal operator.
so mirror awaits next request
Retrieve selected items from the queue as indicated by the operator
'READQ' Requests for ITEM=n
Handle requests as sent by system A.
'READQ' Replies
Operator indicates that no more data is required
'DELETEQ' request
EXEC CICS DELETEQ TS QUEUE('RQUE')
Perform DELETEQ.
CICS/VS defers request
'DELETEQ' reply
until synch point
EXEC CICS RETURN
'SYNC POINT' Request,last
Mirror takes sync-point so deleting RQUE content, and freeing link.
positive response
Transaction termination completes, so releasing T1.
Mirror terminates.
Link available.

6. Remote Transaction Initiation Using NOCHECK Option



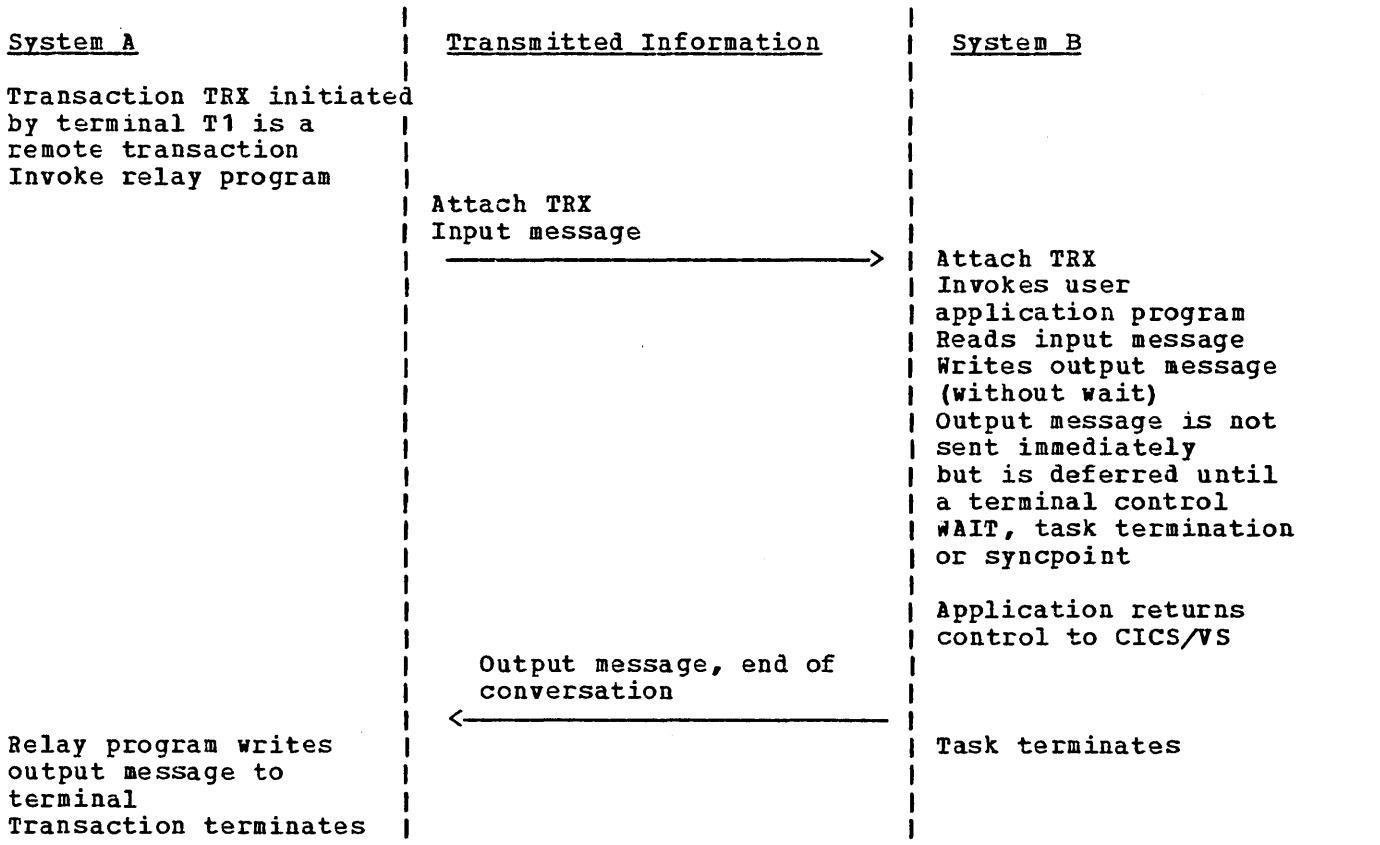
6. Remote Transaction Initiation Using NOCHECK Option (continued)

System A	Transmitted Information	System B
Perform START request with TRANSID value of 'TRZ' and TERMID value of 'T1' Free session		TRY terminates
Terminate Mirror . . . Transaction TRZ is dispatched on terminal T1 and commences processing.	Link available.	

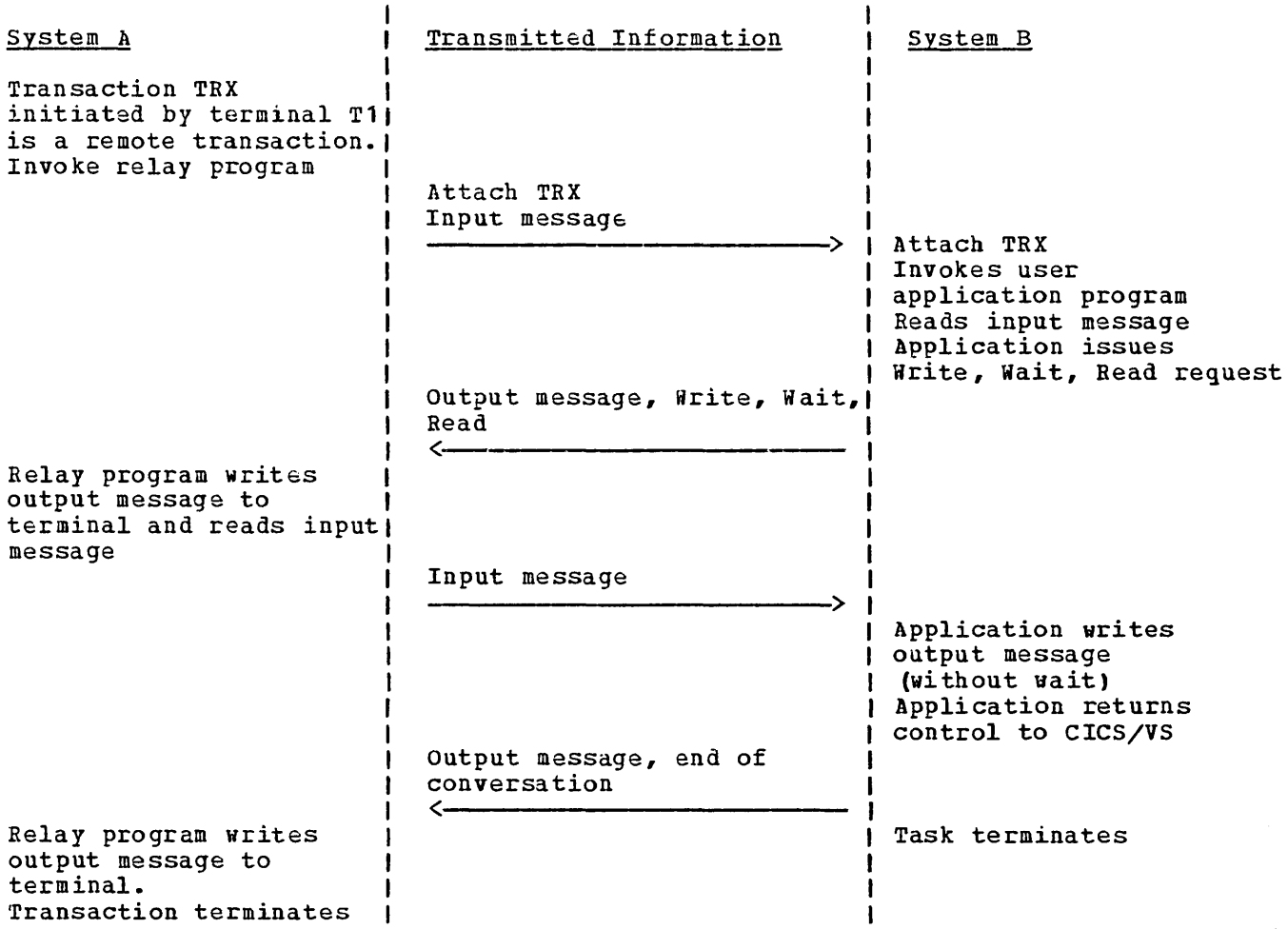
TRANSACTION ROUTING - EXAMPLES

1. Non-Conversational Transaction

This section gives some examples to illustrate the way in which remote transactions are attached and the information flowing between the relay transaction and the user transaction.



2. Conversational Transaction



Chapter 7.3. Distributed Transaction Processing

Introduction to Distributed Transaction Processing

The distributed transaction processing facility (DTP) allows the user to distribute the processing required by any particular application between two or more processor systems. Some of the user's own code, which might otherwise have to be executed on the local system, may be executed remotely. The distribution of processing is achieved by allowing a transaction to initiate and communicate synchronously with transactions running in remote systems.

DTP is provided by the command level interface of the COBOL, PL/I and assembler languages.

Compared with function request shipping (see Chapter 7.2), DTP allows the design of much more flexible and efficient applications. For instance, if it is necessary for an application to browse through a remote file, then if file control function requests were used, at least one request followed by the return of one record would have to flow between the two systems for each record inspected. Using DTP, the local transaction would ship sufficient data to allow the required record to be identified, then a user-written transaction in the remote system would carry out the browsing, and send back only that record.

Without DTP, processing can be distributed between systems by shipping a START function request to the remote system to initiate a user-written transaction. But in that case the remote CICS/VS interval control program schedules the transaction asynchronously. DTP allows synchronous communication between the local and remote systems, which means that the two transactions can hold a conversation and each can carry out processing that depends on the results of a previous stage of processing performed by the other.

The DTP facilities are provided through the command level interface of the CICS/VS Terminal Control program. No special DTP facilities are provided at the macro level. CICS/VS uses SNA logical unit type 6 (LU6) protocols for communication between processor systems. This means that the remote system could be any one that conforms to a compatible subset of the LU6 specifications. One such system, apart from CICS/VS itself, is IMS/VS.

Intercommunication for DTP requires an SNA access method such as VTAM. This means that the remote system is generally in another processor system, not in another region of the same processor system. The exception would be a processor system using the VTAM application-to-application facility.

More than one session may exist in parallel between the two CICS/VS systems. In this case, both function request shipping and DTP may be used simultaneously to communicate between the systems. Any transaction may simultaneously use several sessions for function request shipping and several more for DTP.

DTP allows resources, such as databases, on two or more systems to be accessed synchronously; changes made to local and remote resources can be co-ordinated. Synchronisation points taken in one system will force corresponding synchronization points in the other. Similarly, abends in

one system are communicated to the other, and CICS/VS will back out any uncommitted changes to protected resources on both systems following a transaction abend in either one.

Applications of Distributed Transaction Processing

DTP is used in situations similar to those described for CICS/VS domain-remote function request shipping in Chapter 7.2. It is used in preference to function request shipping when the application needs greater flexibility or better performance through a more sophisticated control of the requests and data that flow between the two systems.

A typical transaction that would be a candidate for distribution is one whose processing can be divided into two logically distinct parts: terminal handling and processing remote data. The terminal handling code would execute in the local system, and the data processing code in the remote system that owned the data resource (see Figure 7.3-1).

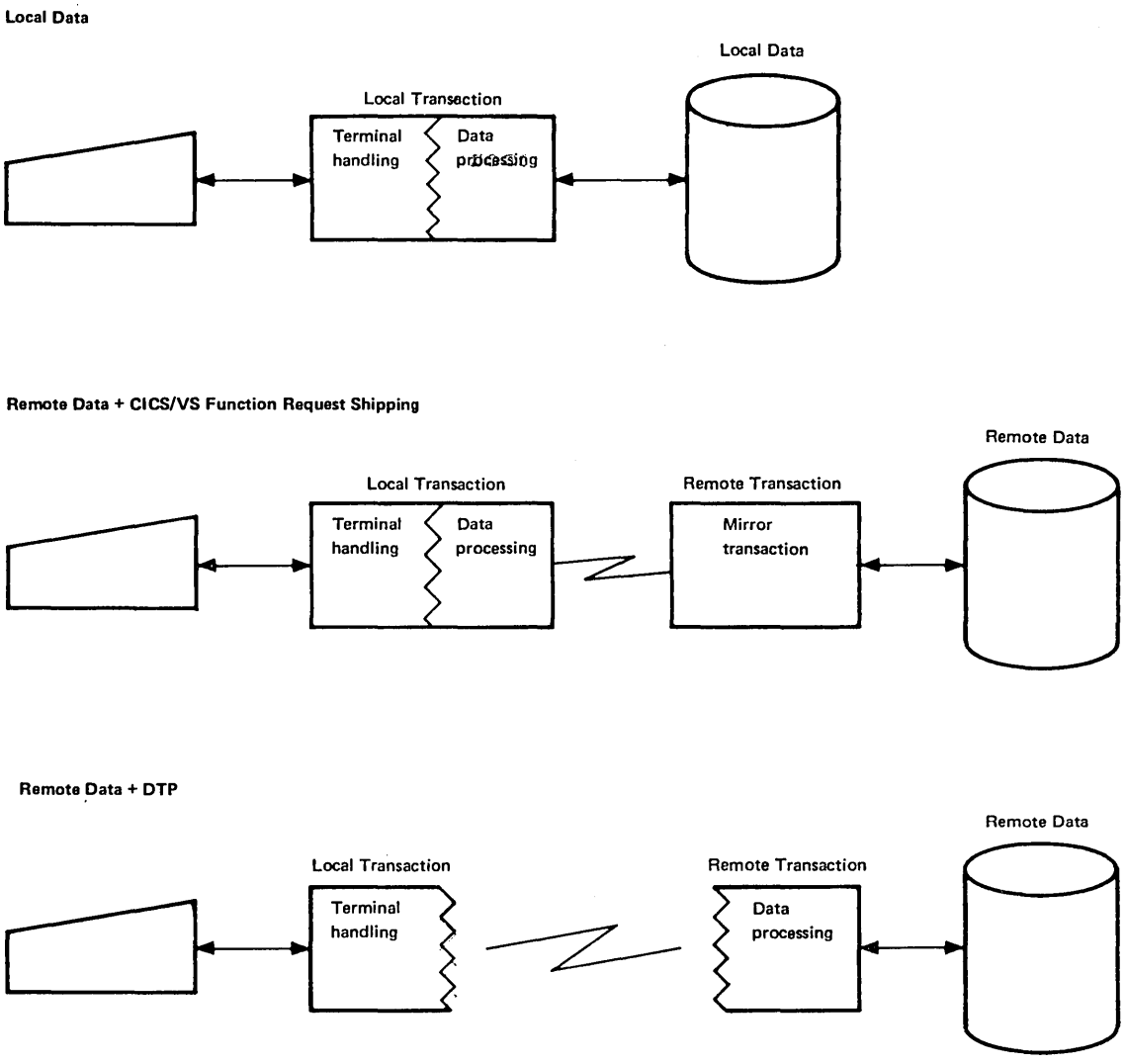


Figure 7.3-1. Development of a Network using Function Request Shipping and DTP.

Transactions can acquire and free sessions according to their needs. They can initate transactions in remote systems and pass sufficient data to allow all processing involving remote resources to be completed remotely; only the results need be transmitted back to the local system. Alternatively, very flexible processing can be implemented, with, for instance, the terminal operator effectively conversing with the remote system.

Design Concepts

DISTRIBUTED TRANSACTION PROCESSING CONCEPTS

To distribute processing, a transaction must acquire a session connecting two systems and start a transaction in the remote system. The first transaction is known as the front-end transaction, and the second as the back-end transaction. The former must always exist as an active task before the latter.

After the connection has been established, the two transactions may be treated as peers: there is nothing that one may do that the other may not. In practice, however, the back-end transaction is usually treated as subordinate to the front-end transaction; it is usually best thought of as a subroutine. Attempts at true peer relationships are likely to lead to complex design problems.

In general, a CICS/VS task is initiated from a terminal, or a logical unit acting as an "intellegent" terminal. This terminal is known as the principal facility of the task. It is the one whose name is found in the EXEC Interface Block field EIBTRMID, and is always connected to the task at its initiation (including at automatic task initiation).

DTP allows the connection of another facility during execution of the task; this is known as the alternate facility. The alternate facility must be a session with another CICS/VS system or logical unit that conforms to a suitable subset of the SNA specifications for an LU Type 6. It is used in CICS/VS-to-CICS/VS intercommunication to send messages to and receive them from remote transactions. A task may have only one principal facility, but several alternate facilities.

Before the back-end transaction can be initiated, the front-end transaction must acquire a session with the remote system by means of the ALLOCATE command. Either the name of the system with which communication is to take place, or the name of a particular session with that system, is specified.

OVERVIEW OF APPLICATION PROGRAMMING INTERFACE

The front-end transaction is always connected to the back-end transaction when the latter is initiated. The session with the front-end CICS/VS system is therefore the principal facility of the back end transaction. The back end transaction communicates with the front end transaction using the basic terminal control facilities.

Once the the back-end transaction has been initiated, it is indistinguisable from the front-end transaction: both can use all the DTP facilities. For instance, the back-end transaction can initiate synchronization points in both systems, free the session, or itself start a session with a remote system (either the one running the front-

end transaction or a third one) and initiate a transaction in it. In the last case, the back-end transaction becomes a front end transaction as well, and the the remote system with which it started a session becomes its alternate facility.

There is one way in which a pair of transactions may be initiated to make the intersystem session into the principle facility of both. The front-end transaction is started by automatic transaction initiation, and a connection with the intersystem session is made at initiation. This makes the connection with the remote CICS/VS system into its principal facility. The automatically initiated transaction uses the session to initiate a transaction in the remote system, making the session the principal facility of the remote transaction as well as the local one.

Since a remotely-initiated transaction may itself initiate a remote transaction, hierarchical trees of transaction may be set up. However, no connection can be made with an already active task: the setting up of a session must always be followed by the initiation of a task. It is not possible, therefore, to create a closed loop (a "daisy-chain") by making a connection back to an active transaction.

After the session has been allocated, the front-end transaction can transmit messages to the remote system by means of the SEND or CONVERSE commands. The front-end transaction initiates the back-end transaction by sending the latter's name either as the first bytes of the data in the first message across the session or as part of an SNA-defined field, called the function management header (FMH), that is transmitted with the first message. The name-prefix is the simpler method, and will suffice in most circumstances. The FMH is commonly used only for communication with logical units other than CICS/VS, such as IMS/VS.

Applications receive messages from each other by executing RECEIVE or CONVERSE commands. CICS/VS transfers the contents of the output data area specified in a sending transaction's SEND or CONVERSE command into the input data area specified in the receiving transaction's RECEIVE or CONVERSE command.

To communicate with an alternate facility, SEND, CONVERSE, and RECEIVE commands must always specify the name of the session that is providing the connection. If no session is specified, the principal facility is assumed.

Use of a session is ended in one of three ways: by one or other transaction terminating, by the execution of a FREE command, or by the execution of a SEND command with the LAST option.

PROTOCOLS

The transactions at either end of a session each own a TCTTE for the session, since communication is via the Terminal Control program. The TCTTE defines the logical unit (always an LU6) at the other end of the session. The session follows the SNA half-duplex flip-flop protocol. This means that at no time can both transactions attempt to send messages together. One transaction must be in send state while the other is in receive state. The state is recorded in the TCTTE. The front end transaction is always in send state when it acquires the session, and the back-end transaction is always in receive state when it is initiated.

A change of state can be initiated only by the transaction currently in send state. It can do so by executing a CONVERSE or SEND INVITE

command, which allow a message to be sent to the remote system prior to changing states, or, less efficiently, by executing a RECEIVE command. In all cases, CICS/VS reverses the states of both transactions.

A transaction in receive state can request the one in send state to reverse the states by executing an ISSUE SIGNAL command. This causes CICS/VS to transmit an SNA signal command to the other transaction. CICS/VS indicates receipt of the command by raising the SIGNAL condition in the SEND state transaction. A transaction may change to receive state following receipt of the signal command, though it is not an error not to do so.

Either transaction may initiate a synchronization point when it is in send state, by executing a SYNCPOINT command. The other transaction may not then send any data until it too has issued a SYNCPOINT command. As an alternative, it may terminate, since termination causes a synchronization point to be taken; when it terminates, the transaction must then either be in send state, or have freed the session with a FREE command. A transaction that uses more than one DTP session must be in send state with all of them when it issues the SYNCPOINT command.

Application Programming Considerations

IDENTIFYING THE REMOTE SYSTEM

There are application program commands for allocating and freeing sessions with other LU6 systems, and for sending data to and receiving data from them via these sessions. The other system is identified either by the session name, or, alternatively on ALLOCATE commands only, by the system name. The identification is supplied by the SESSION(session-name) or, on the ALLOCATE command, the SYSID (system-name) operand. On commands other than ALLOCATE, the SESSION operand may be omitted, in which case, the command is assumed to refer to the principal facility. The system name is supplied by the system programmer in the SYSIDNT command of the DFHTCT TYPE=SYSTEM statement. The session names are defined by him explicitly in the TRMIDNT operand of the DFHTCT TYPE=TERMINAL statement, or are generated by CICS/VS from information supplied in the SEND and RECEIVE operands of the DFHTCT TYPE=SYSTEM statement.

There can be only one session between any pair of transactions at any one time. However, a front-end transaction can invoke several back-end transactions (or a particular back-end transaction several times as several different tasks) and communicate with them in parallel.

A transaction may use CICS/VS function request shipping as well DTP. The remote DTP transaction may execute on the same CIVS/VS system as the function request or on a different one. In both cases, CICS/VS handles the session selection for CICS/VS function request shipping, and the DTP functions need be coded no differently from when DTP is used alone.

An application acquires a session by executing an ALLOCATE command. The name of either the required system or a particular session is specified. If the system name is specified, the name of the session can be found in the field EIBRSRCE of the Exec Interface Block (EIB) after execution of the command has completed.

The PROFILE operand can be used on the ALLOCATE command to specify a set of terminal control processing options. These are generated by the system programmer in the Program Control Table. They determine such factors as whether an FMH received from the other system is to be included in the application program's input data area, and whether automatic journaling is to be used.

Generally, a function management header is not required for communication between CICS/VS systems: the name of the transaction to be attached is specified as the first bytes of user data in the first message to be sent.

For communication with non-CICS/VS systems, an attach function management header may be required when the session is allocated. The ATTACHID operand, specifying the name of an attach header control block, causes an attach FMH to be transmitted with the data. The FMH must have been created previously using a BUILD ATTACH command.

If no session is available when an allocation request is made, the SYSBUSY condition is raised. The default action is to queue the request and suspend the task until a session becomes available. The application program can take other action only if a HANDLE CONDITION SYSBUSY statement has been executed previously.

A program transmits data by executing a SEND or CONVERSE statement specifying the name of the session to be used and the data area containing the data.

A program can receive data only when it is in receive state. It does so by executing a RECEIVE or CONVERSE command. All back-end transactions are in receive state when they are invoked. All front-end transactions are initially in send state. The states can be reversed only by a transaction that is in send state. It does this by executing a CONVERSE or SEND INVITE command, or, less efficiently, a RECEIVE command. The CONVERSE or SEND INVITE command causes a message to be sent before the states are reversed.

Both RECEIVE and CONVERSE allow the naming of a data area into which received data is to be placed by CICS/VS. The data area may contain an FMH prefixing the data. If so, the INBFMH condition will be raised and the field EIBFMH will be set in the Exec Information Block. It is recommended that, in programs that may receive FMHs, an IGNORE INBFMH statement is executed at the start of the program and EIBFMH is tested after each RECEIVE or CONVERSE.

A transaction in receive state can notify the one in SEND state that it needs the states reversed by executing an ISSUE SIGNAL command. This causes the SIGNAL condition to be raised in the send-state transaction when the next SEND or CONVERSE command is executed. The transaction may ignore the condition: CICS/VS takes no action if a HANDLE CONDITION SIGNAL command has not been executed previously. But the application programmer may include code to change states if he wishes.

A session is explicitly freed by the FREE command specifying the session name. Either the front-end or the back-end transaction can issue the command. The command can be issued only when the transaction

is in SEND state or the free indicator (EIBFREE) in the Exec Interface Block is on. A session is implicitly freed when one of the transactions ends.

SYNCHRONISATION POINTS

Either transaction can initiate a synchronization point by issuing a SYNCPOINT command. This causes the one-byte field EIBSYNC in the Exec Information Block in the remote transaction to be set to X'FF'. That transaction must issue its own SYNCPOINT command before sending any more data. Alternatively, it can terminate, since termination causes a synchronization point to be taken. A transaction must be in send state with all its sessions to issue a synchronization point.

EFFICIENT USE OF SESSION

To optimize the use of sessions, CICS/VS implements deferred output processing for SEND commands. This means that, in general, a message is not sent to the remote system until the command following the SEND that generated it is executed. This frequently allows SNA indicators, such as change direction and synchronization point indicators, to be sent with this data, when they would otherwise require a message each.

Output is not deferred if the SEND command has the WAIT operand; the message is transmitted immediately. Indicators generated by subsequent commands would require further messages. This reduces the scope for optimization by CICS/VS. The WAIT operand should not therefore be used without good reason.

In addition to CICS/VS optimization, the application programming interface offers scope for the user to optimize the flow of messages. The INVITE operand allows the number of messages to be reduced in some circumstances, by sending a change direction (CD) indicator with the output data.

For instance, the following three commands result in only one message being sent to the remote system.

```
EXEC CICS SEND FROM(data-area) INVITE
EXEC CICS SYNCPOINT
EXEC CICS RECEIVE INTO (data-area)
```

CICS/VS deferred I/O processing means that the output message is not sent until the SYNCPOINT command is executed, and the INVITE operand means that the change direction indicator is sent with this message. The one message therefore contains the change direction indicator, the synchronization point indicator and the output data. The RECEIVE command generates no messages to the remote system; it simply causes any data received from it to be transferred to the named data area.

The following is an example of how I/O and processing by the local application program may be overlapped using the WAIT operand.

```
EXEC CICS SEND FROM(data-area) INVITE WAIT
.
.
Non-CICS/VS programming statements
.
.
EXEC CICS RECEIVE INTO(data-area)
```

Because of the WAIT operand, execution of the non-CICS/VS programming statements will not begin until execution of the SEND is complete. However, transmission of the input data from the remote system can overlap the processing of these statements. Without the WAIT operand, execution of the SEND would not start until processing reached the RECEIVE statement, so no overlapping would be possible.

System Programming Considerations

A Terminal Control Table entry must be generated for each session that is to be used by DTP. The same considerations apply as to TCT generation for CICS/VS function request shipping (see "System Programming Considerations " in Chapter 7.2).

A special type of entry in the Program Control Table may be generated for DTP sessions, to specify terminal control processing options. A DFHPCT TYPE=PROFILE macro is used. The options control FMH handling, journaling, message handling and other factors affecting the session. Each set of options may be given a name by which it may be specified in the PROFILE operand of the ALLOCATE command.

Design Hints

TYPES OF APPLICATION

Batch Applications: These use only one session, namely the LU Type 6 session between the two systems. The front-end transaction is started by automatic transaction initiation (as a result of either a transient data queue trigger level being reached or a START command). This transaction then sends a message to the remote system to start the back-end transaction. Each transaction becomes the principal facility of the other - no terminal is involved.

This technique provides a convenient way of transmitting queues or files of information across the link. In a typical situation, a local terminal runs a local transaction which stores data on a queue. When the trigger level is reached, another local transaction, the front end transaction for the intersystem link, is initiated. This initiates in turn the back-end transaction in the remote system to receive the data, and transmits the queue to it.

Programming is simplified in batch designs, because the terminal control session is handled separately from the data transmission. Futhermore, delays in response to the terminal operator because of session allocation and use are eliminated.

If it is desired to send queues between two systems in both directions at the same time, the recommended method is to use two sets of transactions on two parallel sessions. This is the most efficient method and it provides the simplest application design.

Examples 6 and 7 at the end of this chapter show queue transfer operations that might be part of batch applications.

Interactive Applications: These use both a terminal session and at least one LU Type 6 session connected to a transaction simultaneously. They are more complex than batch applications, but may be required if enquiry and update on remote systems is to be performed. They also facilitate operator assistance and prompting.

Examples 1 to 5 at the end of this chapter show interactive transactions.

MASTER AND SLAVE DESIGN

The two transactions involved in an LU Type 6 intercommunication should, if possible, be designed as a master, or main routine, and a slave, or subroutine. In general, the front end transaction will be the master.

The logic should be contained as far as possible in the master. This transaction should pass requests to the slave only as necessary. The slave should return its response to the master, and then wait for another request. Attempts to distribute the logic between the two transactions, thus making them into peers, are likely to lead to complex design problems. If the roles of the two systems need to be reversed, it is generally preferable for an independent second pair of transactions to be invoked, rather than for the master and slave roles of a single pair of transactions to be reversed.

Protocols are conveyed at the application programming interface by means of fields in the the Exec Interface Block (EIB). These contain information about the status of the transaction, such as whether it is in send or receive state, whether it is required to take a synchronization point, and whether it is permitted to free the session. Full details are given in the Application Programmer's Reference Manual, including a recommended order for testing the fields. A transaction must perform the actions indicated in the EIB fields, otherwise it will be abended. As far as possible, the master transaction should dictate the contents of the slave transaction's EIB fields, rather than vice-versa.

Fields in the Exec Information block will be updated after execution of a command that transmits or receives messages, so it is advisable for an application program to copy the EIB fields after each such command.

SNA INDICATORS

An understanding of the SNA protocols and corresponding data flow control indicators used by CICS/VS, particularly BB (begin bracket), EB (end bracket), and CD (change direction), will aid the design of efficient and error free applications.

A conversation between two transactions is delineated by BB and EB indicators: a conversation is, in other words, an SNA bracket. It is started by the first SEND command following allocation of the session. This generates the BB indicator. It is ended by one of the following, all of which generate EB indicators.

a SEND command with the LAST operand
a SEND command followed by a FREE command
a SEND command followed by termination of the task

The CD indicator changes the issuing transaction from send state to receive state, and the other transaction from receive state to send state. It is generated by one of the following.

a SEND command with the INVITE operand
a CONVERSE command
a SEND command followed by a RECEIVE command

It should be noted that all the indicators are always issued by the transaction that is in send state. The front-end transaction is always in send state when it is initiated and this transaction issues the BB indicator; the front-end transaction must issue the first CD indicator and thereafter only the transaction in send state may issue it; and the EB indicator can be issued only after a SEND command.

The examples at the end of this chapter show how these and other SNA indicators are used by CICS/VS.

QUEUE TRANSFER

The following special considerations apply to transactions intended to transfer queues of data between systems.

- The sending transaction should be designed as the master and the receiving transaction as the slave.
- For simplicity of design, a separate pair of transactions should be used for each queue to be transferred.
- The master transaction should not send large numbers of records without first obtaining confirmation from the slave transaction that it is attached and is processing the records successfully.
- For large queues, take synchronizations points at regular intervals, unless performance is a crucial consideration.
- For large queues, use the pacing facilities of VTAM to avoid flooding the network. Alternatively, use frequent synchronization points, which have a similar effect to pacing.
- It will probably be advisable for the receiving transaction to store the incoming data on either a transient data or a temporary storage queue, rather than to update permanent storage as the records are received. The design will be simpler, because the updating will be a separate operation and because error recovery will be easier.
- Consider how transmission should be restarted if an error occurs when the queue has been partly transmitted. There is a basic choice to be made between continuing from the point at which the error occurred and retransmitting the whole queue.

Examples 6 and 7 at the end of this chapter show queue transfer between systems.

MULTIPLE LU TYPE 6 SESSIONS

A transaction may initiate several transactions in other systems. The design of applications using such transactions is likely to be very complex unless it is highly structured. The least complicated design will probably be a master/slave tree, in which each transaction acts as the master of all transactions for which it is the originating node, and the first transaction to be initiated is the master of the whole tree.

The master/slave concept is particularly important in relation to synchronization points in a tree of transactions. Unless these are originated by the transaction that is the master of the whole tree, they are unlikely to be successful. They are originated at the top of the tree, then propagated down the whole tree.

Any transaction issuing a synchronization request must be in send state with respect to all its LU Type 6 sessions, except those for which either a synchronization point has been requested (EIBSYNC set) or which have been freed by the transaction at the other end of the session (EIBFREE set). If an attempt is made to take a synchronization point when these conditions do not hold, the transaction making the attempt will be abended, which will lead to all transactions in the tree being abended.

ERROR HANDLING

If a transaction abends, its partner at the other end of the session will be notified if the abending transaction:

- is in send mode; or
- received a definite response request (for instance, by the use of a SEND command with the DEFRESP operand in the other transaction); or
- received a request for a synchronization point; or
- received a request to free the conversation.

A transaction may not, therefore, be notified of an abend by its partner. This is why the sending transaction in a queue transfer application should check periodically that the records it sends are being received and processed.

A transaction in receive mode can notify the sending transaction of an abnormal situation by executing an ISSUE SIGNAL command. The SIGNAL condition will be raised in the sending transaction at the next execution, after receipt if the signal request, of an output command on the session. This transaction should normally then stop sending data and issue one of the commands that generate an SNA CD (change direction) indicator (see "SNA Indicators" earlier in this chapter). This protocol is the responsibility of the user; the receiver of the signal request is not forced by CICS/VS to stop sending.

ISSUE SIGNAL and the SIGNAL condition can lead to complex programming, so they should be used only when definitely required.

CICS/VS TO NON-CICS/VS SYSTEMS

CICS/VS can communicate with transactions running in other types of system, provided they implement a suitable subset of the SNA LU Type 6 protocols. This includes IMS/VS. It is necessary, however, for the designer of such applications to understand in detail the SNA data flow control commands and protocols generated by the other system.

In some cases the CICS/VS transactions converses with the remote system, rather than with user written transactions running in that system. CICS/VS transactions converse with the DC component of IMS/VS, for example, so the protocols and data formats of that component must be understood and compiled with.

Other systems may allow direct communication with their transactions. It is then necessary to know the protocols generated by user-written code in the transactions. In particular, it is necessary to know how BB (begin bracket), EB (end bracket) and CD (change direction) indicators are generated and responded to.

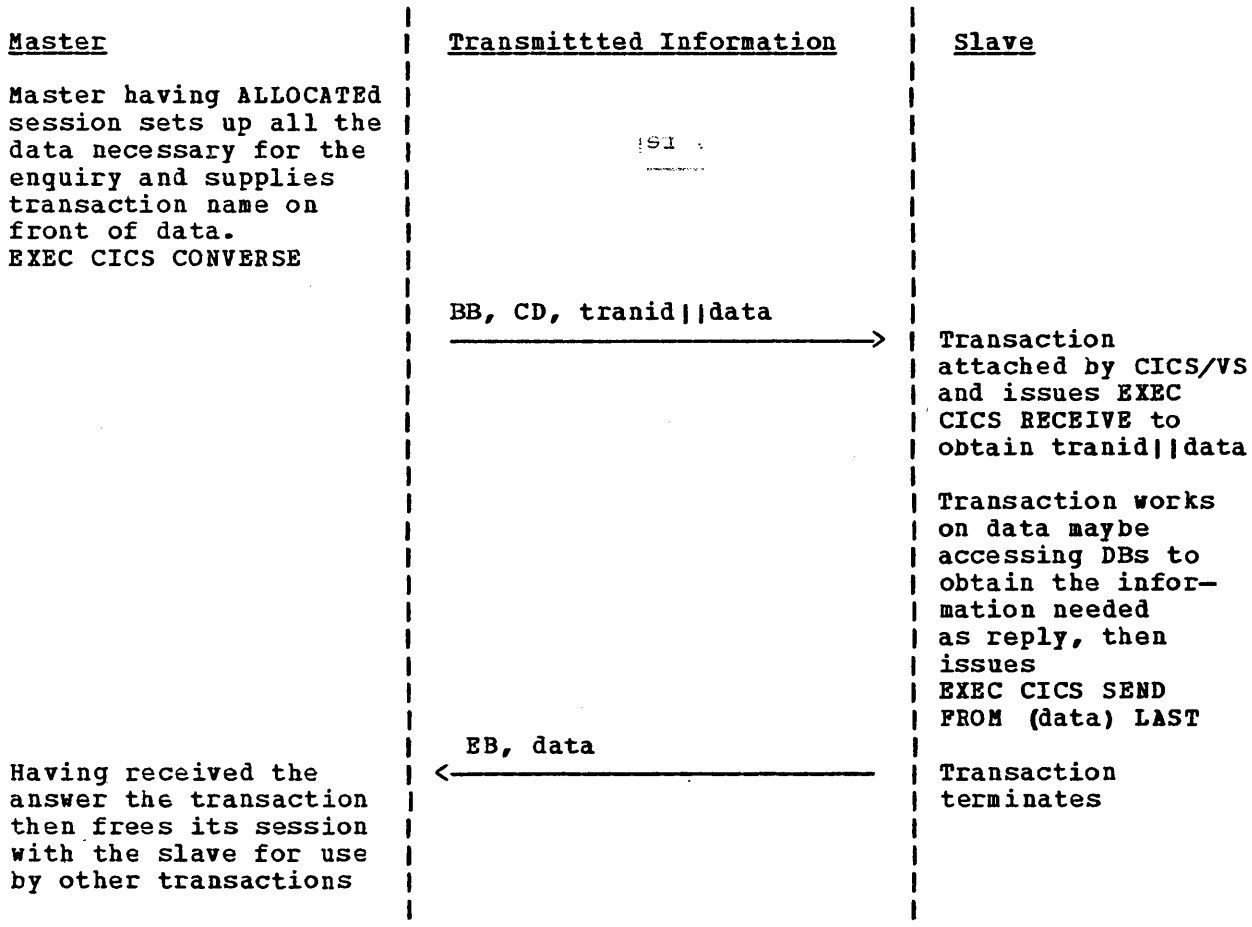
In any case, the following problems will need attention during the design of the application.

- How the required transaction is to be attached in the remote system. It may be necessary to send an attach header, in which case the remote transaction could have a name up to 8 characters long.
- The structure of the messages passing between the local and remote transactions, and how any mapping component of the remote system is to be used.
- The possible replies to each type of request, together with the SNA indicators that may need to be present on the request and replies.
- Ensuring that the SNA indicators are followed precisely by the transactions at both ends of a session.
- Which transaction or transactions may end a conversation.
- Whether synchronization points are to be used, and if so, whether they are to be on single or multiple sessions. The remote system may support only single session synchronization points.

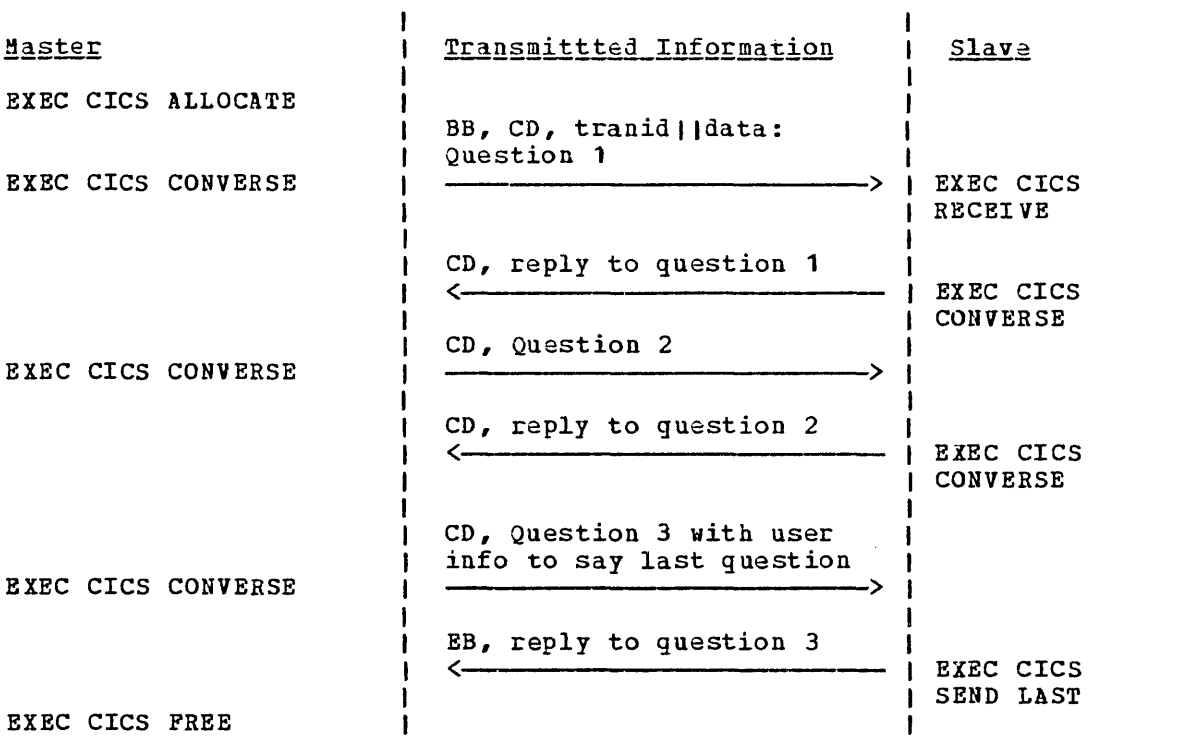
Distributed Transaction Processing Examples

This section gives some examples of transactions using distributed transaction processing to communicate between systems. The SNA flows between the systems for particular application program commands are shown. The simplest examples are given first, and successive examples introduce more function. The first five examples illustrate the types of flow that might arise from interactive applications and the last two those from batch applications.

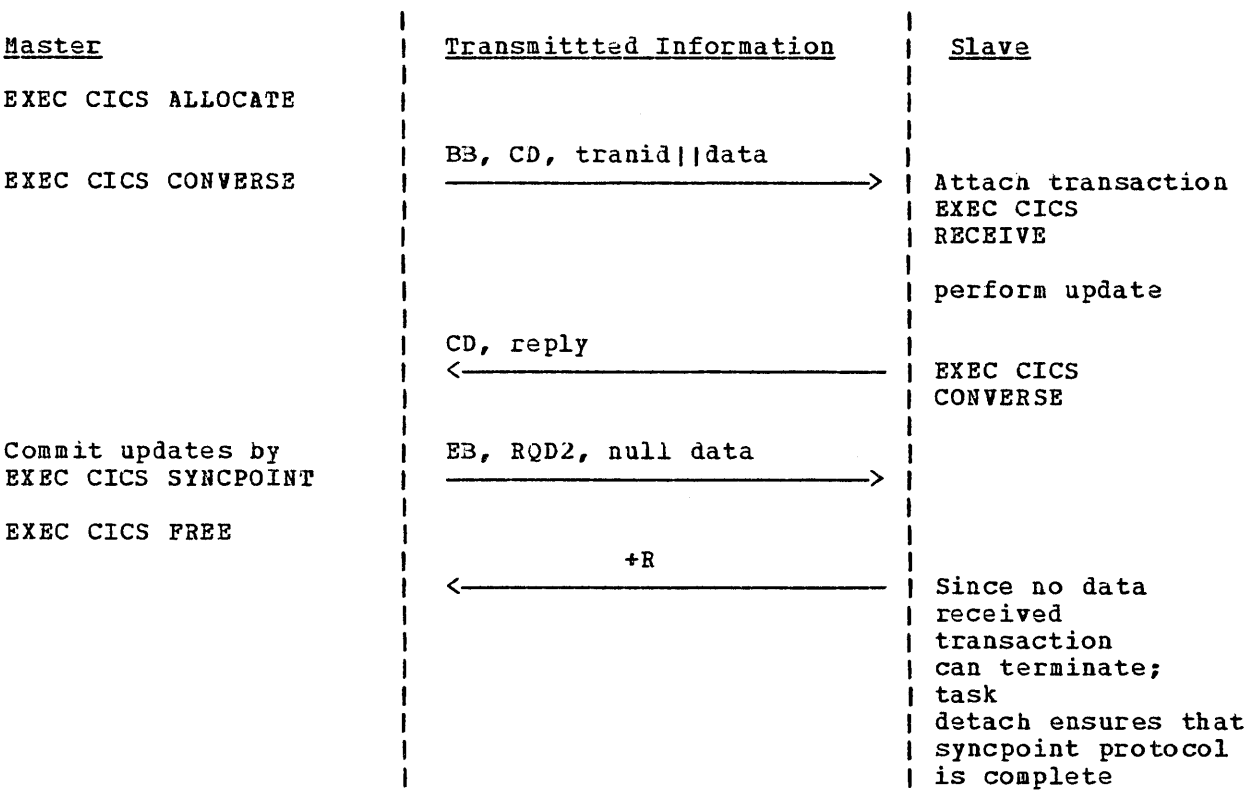
1. Simple Single Enquiry



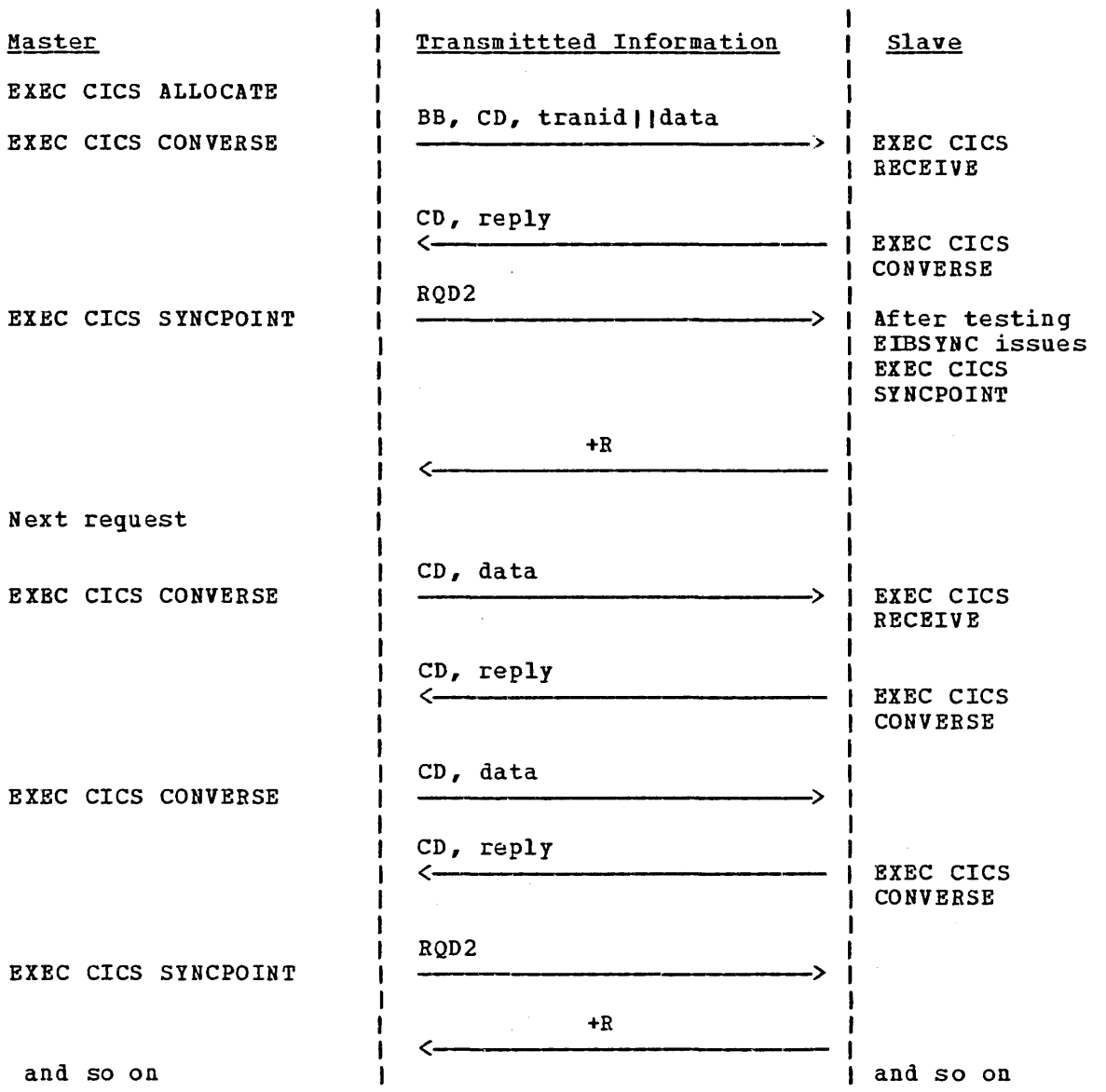
2. Simple Multiple Enquiry



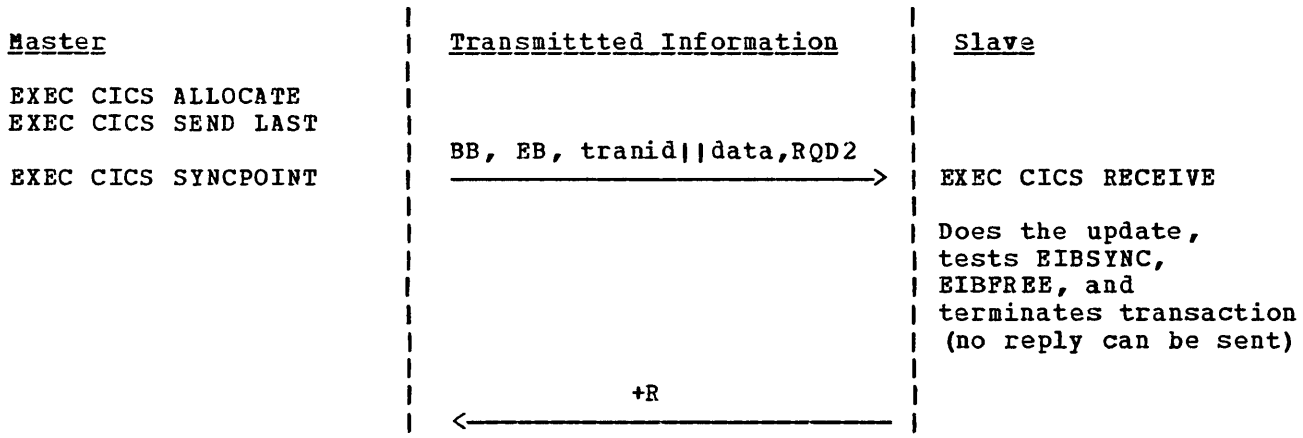
3. Simple Update Request



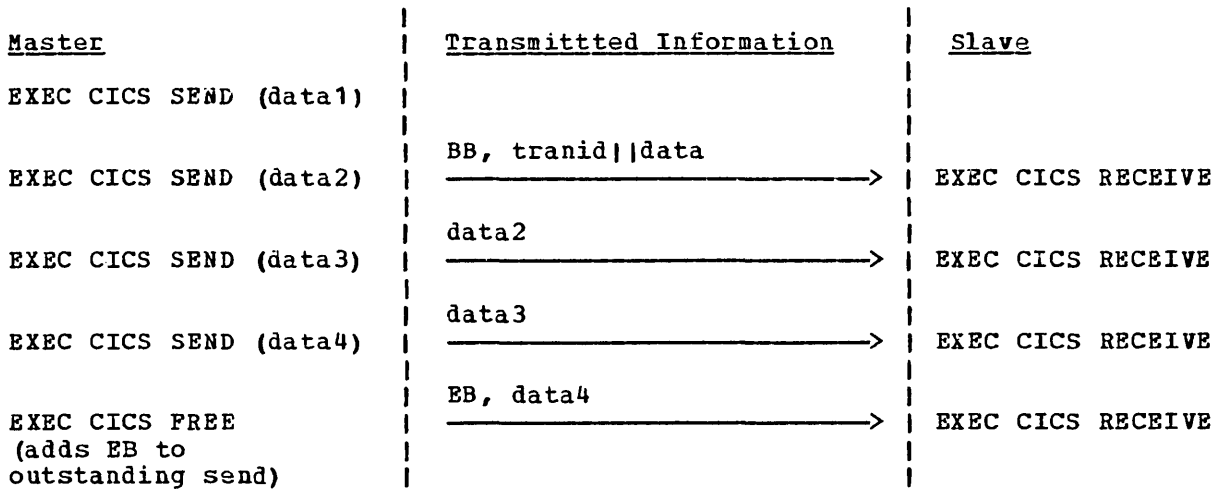
4. Simple Update with user synchpoint



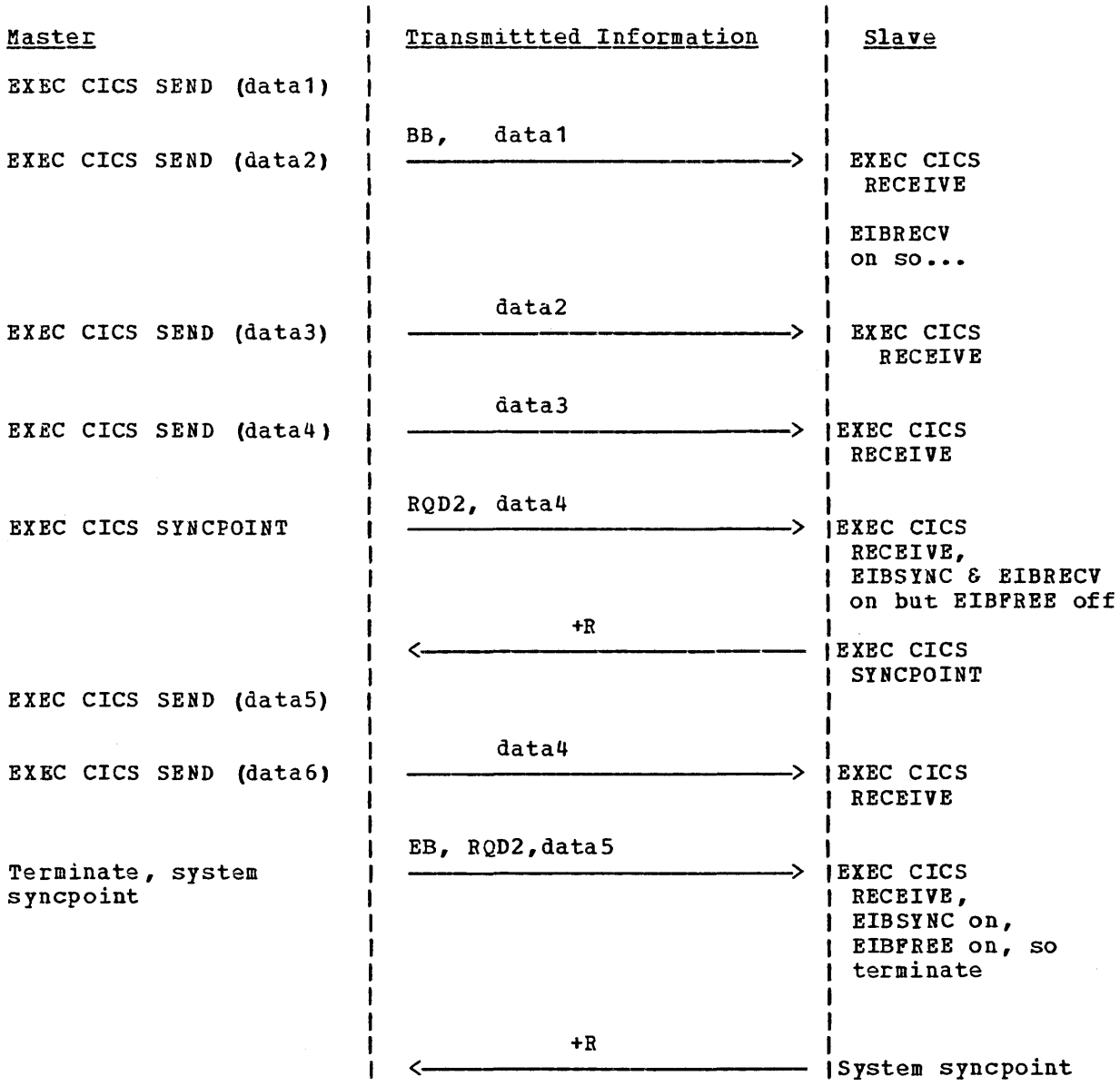
5. Optimized Single Update Request



6. Optimized Queue Transfer Without Synchpoint



7. Optimized Queue Transfer With Synchpoint and/or Termination



Chapter 7.4. Sessions Between Domains

This Chapter discusses the means of communication between two domain-remote CICS/VS systems, that is between two CICS/VS systems in different processor systems or between two CICS/VS systems in the same processor system that use the VTAM application-to-application facility.

CICS/VS intercommunication uses Systems Network Architecture (SNA) communications protocols. CICS/VS establishes one or more logical communications paths (or sessions) with each of the other CICS/VS systems with which it may be required to exchange messages. The sessions and protocols are managed entirely by CICS/VS and are transparent to CICS/VS application programs.

CICS/VS supports intersystem communication sessions only through ACF/VTAM (or an equivalent SNA access method). As Figure 7.4-1 shows, the physical links between the processors may be through communications facilities using SDLC line protocols, or through the multiple channel attachment capabilities of the IBM 3705 communication controller. In addition, two or more CICS/VS systems in the same processor may communicate through the ACF/VTAM application-to-application facilities (not illustrated) or through the interregion communication facility of MRO (discussed in Chapter 7.2). In the network shown in Figure 7.4-1, any of the three transactions may communicate with either or both of the others, and any of them may be minor transactions. Although CICS/VS DTP and domain-remote function request shipping are supported only through ACF/VTAM (or equivalent), the application program making the request may be invoked from any CICS/VS-supported terminal attached by means of BTAM, VTAM or TCAM.

CICS/VS intersystem communication facilities can run with either CICS/OS/VS or CICS/DOS/VS, using the command level interface. For communication between a number of processors, the ACF/VTAM Program Product with the Multisystem Networking Facility feature is required, together with the ACF/NCP/VS program product and the System Support Programs for ACF/NCP/VS.

Applications using intersystem communication can be tested by using two CICS/VS systems in the same processor communicating via VTAM. For this configuration the ACF/VTAM Program Product only is required. CICS/VS intersystem communication is independent of both the location of its remote partner and the means of physical communication employed by ACF/VTAM.

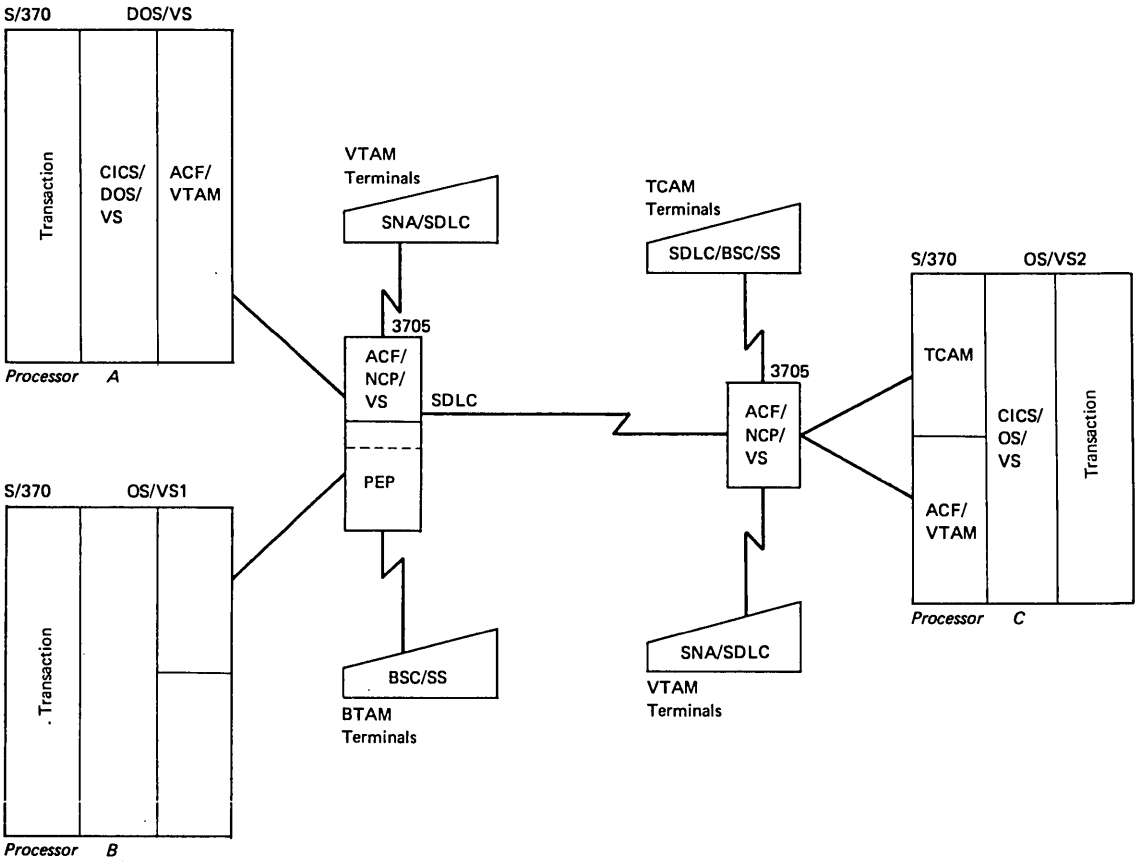


Figure 7.4-1. A Possible Configuration Connecting Three CICS/VS Systems

The Session

CICS/VS uses ACF/VTAM to establish logical connections with other remote systems. ACF/VTAM uses SNA protocols to establish the physical connections between connected systems necessary to complete the logical connection. This logical connection is called a session and is independent of the physical routes selected by ACF/VTAM and ACF/NCP/VS. There can be multiple sessions over a single physical route. Within the session, the CICS/VS terminal control program uses specific SNA protocols to exchange requests and replies between the two systems.

CICS/VS requests ACF/VTAM to establish a session with another specific system either at CICS/VS initialization, or when requested by means of a CICS/VS master terminal or VTAM system operator command, or at the time that a command level request is issued requiring communication with a remote system for which a session is not available. Refer to Part 3 of this manual and to the CICS/VS System Programmer's Reference Manual for further details on establishing CICS/VS sessions via VTAM. For intersystem communication, each CICS/VS system is defined as a logical unit to VTAM and the network. The session is established between the two CICS/VS system logical units. If an application program makes a request that requires a remote resource or transaction, but the remote system is out of service, the request is terminated with a code that indicates that the system is unavailable. If the remote system is in service, but no session exists, CICS/VS will request VTAM to establish a session (provided INTLOG has been specified for the session by the system programmer or terminal operator). If no session can be established, the request will be terminated. No requests will be queued awaiting later establishment of a session (except when the local queueing facility is operational, as described under "Local Queueing" in Chapter 7.2).

When the intersystem communication component sends a request to a transaction in another system, the session selected is allocated to the requesting transaction program and the corresponding remote transaction until one or other transaction terminates. While the session is allocated to a pair of transactions in this way, the session cannot be used by any other transactions. It does not become available again until both transactions have freed it, either explicilty or by terminating.

When there are one or more sessions with the required system but none is available, the intersystem communication component will delay any request to allocate a session until one becomes available, thus putting the requesting application program into a CICS/VS wait condition. If contention is high, the wait may become apparent to the associated terminal operator.

A large queue may degrade the performance of the system overall by causing heavy paging of main storage. In this respect, remember that inquiry applications, or transactions that deal only with unprotected resources (for example, initiation of transactions with unprotected request identifiers) require allocation of a session for a shorter period of time than those that manipulate protected resources. The use of the CMXT operand of the DFHSIT macro (which limits the size of queues) or the DTIMOUT operand of the DFHPCT macro (which limits this time a transaction waits for resources) should be considered as means of limiting contention for sessions.

When sessions are established, the protocols used specify which of two systems shall win contention when both systems concurrently attempt to allocate the session on behalf of an application program. The designer should determine which system has the higher probability of, or higher priority for, allocation of the session between two systems, and

reflect this in the SEND, RECEIVE or SESTYPE operand of the DFHTCT TYPE=SYSTEM macro, which is described in the CICS/VS System Programmer's Reference Manual.

CICS/VS could be controlling several sessions on behalf of an application program using CICS/VS intersystem communication extensions: one with each remote transaction with which the application interacts; and one (or its non-SNA equivalent) with the terminal associated with the local transaction. The transaction-to-transaction sessions may be with several different remote systems. Furthermore, they may include several parallel sessions with a single remote system. Just as when a session with a terminal fails, so, when a session with a connected system fails, the transactions at both ends of the session are abended.

Operating Considerations

Only the CICS/VS master terminal operator is involved with CICS/VS intersystem communication, because the normal terminal operator is unaware of the location of programs and data, which all appear to be in the system to which the terminal is connected.

The master terminal operator may be involved in starting and stopping sessions between CICS/VS systems, and making inquiries about the status of the session. The only commands related to intersystem communication are: ACQ (acquire); INQ (inquire); REL (release); and INSERV/OUTSERV (change service status). These commands are listed in the CICS/VS Operator's Guide. For consistency with existing commands the individual sessions between remote CICS/VS systems are treated as VTAM-supported terminals and are referenced by means of the name in the TRMIDNT operand of the DFHTCT TYPE=SYSTEM macro defining a particular connected system. For convenience the inquire and change service status commands may also address the system by means of the SYSIDNT operand of the DFHTCT TYPE=SYSTEM macro.

Before two domain-remote CICS/VS systems can be connected successfully they must both be active and must both be connected to VTAM. This operation is described in the CICS/VS Operator's Guide. In addition, the entry for the common session must be specified as "in service" in both systems before issuing the acquire command from either system. Session initiation will include, where necessary, resynchronization of units of work. This will ensure that any system inconsistencies resulting from previous use of the session are diagnosed. No additional restart procedures beyond those discussed in Part 5 are required when CICS/VS intersystem communication is involved.

Chapter 7.5. Recovery and Restart

Introduction to Intercommunication Recovery

Recovery is concerned with ensuring that, following a failure, the recoverable resources being modified by the interconnected system, both locally and remotely, are restored to a self-consistent and well-defined state. Part 5 of this manual describes how this is done in a single system by backing out the changes made by a logical unit of work in progress at the time of the failure. The backout is performed either during emergency restart following system failure, or by Dynamic Transaction Backout for individual transaction failures.

Provided the intercommunication sessions remain intact, and in most instances when they do not, the concepts of recovery, logical units of work, and sync points are applicable to connected CICS/VS systems unchanged: should a transaction or one of its connected transactions fail to complete, all changes made to recoverable resources on all of the CICS/VS systems will be backed out to the state they were in at the last sync point, (or beginning of the transaction in the case of the first or only logical unit of work). This applies whether the facility being used is function request shipping, transaction routing or DTP, whether the the connection is via MRO or VTAM and whether the transactions involved are user-written, mirror or relay transactions.

Note that each system maintains its own system and dynamic logs and that changes to recoverable resources are recorded on the logs in the system owning the resource, as opposed to the system that initiated the change.

Designing for Recovery

FAILURES IN CONNECTED SYSTEMS

The failures that may occur in connected systems are:

1. Session failures, either between the CICS/VS systems or between CICS/VS and the terminal associated with a transaction. These failures cause transaction abends of the transaction or transactions connected to the session and resource recovery is performed as for non-connected transaction abends.

2. Total CICS/VS system failures. The failing system is recovered using emergency restart as for a non-connected system though there are extra features, described following, added for intersystem communication. Any remote system connected at the time of the system failure, sees the failure as a session failure and treats it as such. Thus, a remote system failure causes a local transaction abend.

3. Transaction abends. These are recovered using dynamic transaction backout as for non-connected systems. The mirror and relay transactions are not special in this respect and to make full use of CICS/VS recovery, dynamic transaction backout should be specified for both. The transaction restart facility may also be used. This may not be specified for the mirror or relay transaction, but, if specified for the associated user-written transaction, the mirror or relay transaction will be restarted.

DATA BASE SYNCHRONIZATION

To achieve proper synchronization of resource changes following any of the above failures, it is necessary for CICS/VS to know in one system whether or not the other system has completed the current logical unit of work. When the intersystem session itself fails, it is not always possible to find this out. Of course, applications that do not change resources in the remote system present no problem - it makes no difference whether they complete or not. In other cases, CICS/VS limits the period over which it is "in-doubt" as to the disposition of the LUW in the other system to a small period during the course of synchronization point processing (initiated by the SYNCPOINT or RETURN commands). During this period, the system local to the application is awaiting acknowledgement that the remote systems are also ready to commit the changes.

Should the intersystem session fail before reaching the in-doubt period, both sides will back out. After this period, both sides will complete (commit) their changes. However, should an intersystem session fail during this short but critical period, the systems will take unilateral decisions on whether to back out or complete. (This allows them to release the enqueues held on the recoverable resources and continue processing independently.) These unilateral decisions may result in data base changes being out of synchronization. However, in the great majority of cases, it is possible to design applications such that the impact of the lack of synchronization is not too severe:

Suppose we have two linked updates, A and B, to perform. If it is imperative that update B occurs only if update A occurs, it is usually not so serious if update A occurs but update B is delayed. This is because one out-of-synchronization state usually corresponds to an item being overcommitted (for example, double booking), whereas the other only means the item is (temporarily) undercommitted or treated as unavailable (for example, the delayed cancelation of a reservation). CICS/VS intersystem communication is flexible enough to take advantage of these characteristics.

To achieve the best design, the rules obeyed by the system and the features available must be clearly understood:

1. If ever an intersystem session failure results in the possibility of data base changes being out-of-synchronization, message DFH2101 is issued to the CSMT transient data destination in the application's local system at the time of the failure. The message contains the following parameters, which can be used to cross-reference to the corresponding message at session recovery (see below): time stamp; transaction identifier and task number; operator identifier and operator's terminal identifier; remote system identifier and terminal control table entry identifier for the remote system.

2. If the local transaction issues a synchronization point request, but the remote transaction fails to complete, the local transaction will be abended, and backed out if dynamic transaction backout is specified. This is ensured by backing out the remote transaction only when it is known that the local transaction will see the failure and itself abend. When a session failure occurs during the in-doubt period, the remote transaction will commit its changes. This avoids the possibility of the local transaction completing before the session failure is detected at its side, so leaving the possibility of a data base synchronization error being undiagnosed by the local system.

3. If in doubt as to whether the system completed or backed out, the local CICS/VS system's action following a synchronization point issued by one of its transactions depends on the dynamic transaction backout specification in the Program Control Table entry for the transaction, as follows:

   a. DTB=NO (or the operand omitted). The local transaction is abended.

   b. DTB=YES The local transaction is abended and its changes backed out (because the remote transaction might also back out).

   c. DTB=(YES,NO). If the failure occurs before the intersystem sync point exchanges begin, the local transaction is abended and backed out as normal, knowing that the remote transaction will also be abended. When in doubt because of a session failure during the sync point exchange, the local transaction is abended but Dynamic Transaction Backout is not invoked, even though the remote transaction might back out. This option is available for the cases where the DTB=YES option may not be appropriate for the application; for example, where it may result in lost data, as opposed to duplicate data. Note that if dynamic transaction backout is not attempted, transaction restart will not be invoked.

   d. DTB=(YES,WAIT). This specification applies to communication via VTAM but not to communication via MRO. If the failure occurs before sync point exchanges begin, the local transaction is abended and backed out as normal. If the failure occurs during the sync point exchanges, the local transaction is abended but Dynamic Transaction backout is not invoked, as for DTB=(YES,NO). The difference is that enqueues resulting from WRITEQ TS commands or START commands with the PROTECT option issued by the local transaction to write to local temporary storage queues, or start other local transactions, are not released. In other words, these actions will be neither committed nor backed out on session failure.

      During session recovery, the local CICS/VS system will commit the enqueues and STARTs or back them out, in accordance with the action taken by the remote system prior to the session failure. If the session recovery is unsuccessful, the START commands will be cancelled but the temporary storage queue changes will be committed. DTB=(YES,WAIT) is effective only if there is a single session between the local and remote transactions. It offers a method of ensuring complete recovery from a session failure. Provided the only way that the local transaction modifies recoverable resources is by START PROTECT and WRITEQ TS commands, CICS/VS can be relied upon to resynchronize the two systems during session recovery. It has the disadvantages that the temporary storage queue remains locked from the time of session failure and cannot be used until recovery is completed.

4. On session recovery, message sequence numbers will be exchanged and compared to see whether the unilateral actions taken by the systems matched. The appropriate message (DFH2102 or DFH2103, respectively) is written to CSMT again in the system that issued the synchronization point request. The message includes the same parameters as in the corresponding message (DFH2101) issued at the session failure (see 1. above). In the DTB=(YES,WAIT) case, message DFH2106 or DFH2107 will be issued at session recovery to indicate whether both systems finally committed or backed out. If a mismatch occurred, the appropriate reconciliation action can be taken by user. (See discussion below.)

   The detail of exchanges between systems at sync point time, and their relationship to the in-doubt period will be found in the CICS/VS Diagnosis Reference Manual.

CONNECTED SYSTEM RECOVERY - AN EXAMPLE

As an illustration of how the above may be used, consider the following simple example: A transaction is given a part number; it checks the entry in a local file to see whether the part is in stock; decrements the quantity in stock; and updates the stock file and sends a record to a remote Transient Data queue to initiate the dispatch of the part. It is assumed that function request shipping is used, which means that a mirror transaction runs in the remote system, but the same principles would apply if DTP were being used and the remote transaction were user-written.

Ideally, the update to the local file should take place only if the addition is made to the remote Transient Data queue, and vice versa. The first step towards achieving this is to specify both the file and the TD queue as recoverable resources and to specify DTB=YES on the PCT entries for the local transaction and the mirror transaction in the remote system. Synchronization of the changes to the resources is thus ensured for all cases except an intersystem session failure at the in-doubt period during the execution of the RETURN at the end of the transaction. The actions in this case are determined by the rules given above.

Firstly, message DFH2101, warning that the resources might be out of synchronization, might be sent to CSMT. If not, the resources are synchronized - either both sides backed out as indicated by the normal Dynamic Transaction Backout messages or both completed.

If message DFH2101 was issued, the mirror may, or may not, have backed out. Since DTB=YES was specified in the example, the local transaction will be backed out. Thus, there is a danger that the entry dispatching the part was added to the remote TD queue but the stock file was not updated - the part might then be dispatched a second time elsewhere. If, however, DTB=(YES,NO) were specified for the transaction, the application would complete if the session failed during the critical period (even though the mirror might back out). The out-of-synchronization state, if it occurs, now corresponds to the stock record having been decremented but the dispatch request not yet having been sent. This is probably an acceptable situation, especially since the delayed dispatch can readily be reinitiated on session recovery. The DTB=(YES,NO) option should therefore be used in this case.

When the session is eventually recovered, CICS/VS will check whether the resources are in fact out of synchronization. If they are not, message DFH2102 is issued and all is well. Otherwise, DFH2103 is issued and a transaction to reconcile the mismatch should be run. In this

case, the reconciliation process is simply to retransmit the dispatch record to the remote transient data queue. This could be implemented by the same application with special logic to inhibit local changes.

Note that in general the reconciliation process will be a rerun of the original transaction with local changes inhibited for the DTB=(YES,NO) cases, and remote changes inhibited for the DTB=YES cases.

An alternative approach to the problem would be to split the transaction into two logical units of work by taking a user sync point between the file update and the TD WRITE. This would ensure that the out-of-synchronization state was the safe way round. The main disadvantage would be increased exposure to the problem. If the second logical unit of work could not be run for any reason, including an intersystem session being unavailable at the Transient Data request time as well as during sync point processing, the out-of-synchronization state would arise.

Another disadvantage of the two logical units of work solution is the lack of automatic diagnostics, should the out-of-synchronization state occur.

INTERSYSTEM COMMUNICATION AND EMERGENCY RESTART

As stated above, a total system failure looks to other connected systems like a session failure. The failed system is restored by Emergency Restart (as for a non-connected system), and the local resources are recovered in the usual way. Emergency Restart then effectively restores the intersystem components to the state they would have been in had the original failure been intersystem session failure alone.

Message sequence numbers are recovered from each system log as well as sufficient information to take action as specified in the rules above for intersystem session failures. Thus, if the system was running an application in the in-doubt state at the time of the failure, then on emergency restart this state will be restored. Actions will then be taken as for a failed session. The changes to recoverable resources will be committed, backed out, or held waiting as though the session had failed, and message DFH2101 will be issued warning that resource changes might be out-of-synchronization. When the session is restored, DFH2102, DFH2103, DFH2106, or DFH2107 is issued in the usual way.

Thus, Emergency Restart is made transparent to intercommunicating systems design.

RECOVERY AND MULTIPLE CONNECTIONS

For the straightforward case of one transaction connected directly to a number of others, there are no special considerations: for those intersystem sessions which remain intact through the critical periods, all transactions will abend if one of them does. For those sessions which do not, the rules described above for a single connection will apply.

Recovery and Chained Transactions

For chained transactions, where a request is routed through a second transaction to a third one in a third CICS/VS system, the same rules apply, except that if the session between the second and third transactions fails at the critical time then the second system is considered to be the application local system (that is, the system which initiated the synchronization point). Thus the diagnostic messages DFH2101, DFH2102 and DFH2103 would not appear in the first system. For this reason, remote updates to recoverable resources (as opposed to enquiries) via chained transactions are not recommended.

Error Handling Programs for Intercommunication

CICS/VS intercommunication uses CICS/VS terminal control facilities to exchange messages with connected systems. When an unrecoverable situation is detected in either CICS/VS system the exchange of messages will be terminated by means of a negative response. The negative response will be sent to the CSMT destination by the receiving system. It is followed by a detailed error recovery message. The sense code in the error message will lead to abnormal termination of both the transactions, so that CICS/VS dynamic transaction backout processing can be invoked to guard against inconsistent resource updates.

In case of domain-remote intercommunication, the negative response received by CICS/VS is handled by the Node Abnormal Condition Program (DFHZNAC) and passed to the user-supplied Node Error Program (DFHZNEP) if present. The default actions set by CICS/VS ensure that CICS/VS reads in the following error message. The sense code in this message is made available to DFHZNAC and DFHZNEP in the same way as system sense codes carried by the LUSTATUS commands or negative responses. CICS/VS default actions based on this system sense code are set by DFHZNAC, before making the code available to DFHZNEP. Thus, error conditions occurring on intersystem communication sessions are handled exactly like errors on other SNA sessions through VTAM. Details of VTAM error-handling functions will be found in the CICS/VS System Programmer's Reference Manual, which also contains the detailed action codes for system sense codes.

It is not necessary to write a Node Error Program to handle intersystem communication sessions, since the default actions set by DFHZNAC have been selected to enforce correct recovery based on the error condition detected. When the system sense code indicates that the original request to VTAM may be retried, CICS/VS will do so transparently to the application program attempting to send a message.

Apart from abnormal termination of a transaction, the other error conditions may only occur when the intercommunication component is attempting an invocation of a transaction. The error action codes specify abnormal task termination for those errors in this category which are not recoverable; for example, when the PCT entry for the remote transaction has been disabled by the Master Terminal Operator of the remote system.

Data Base Interlock

As a part of data base and application design in a single CICS/VS system, one must be careful not to design programs in such a way that two programs running concurrently can request the same records in such a way as to interlock on each others requests. (See "File Control — multiple file operations" in the appropriate command level CICS/VS Application Programmer's Reference Manual).

This problem continues to exist in interconnected systems where application programs in two different systems can cause transactions in a third system to interlock in a similar manner. Such an interlock will be detected by means of a timeout value specified for the PCT, which will expire when a program has waited the specified period without a reply from the deadlocked transaction. CICS/VS will abend the task that has been waiting the longest, so breaking the interlock and allowing the contending task (or tasks) to continue.

Use of transaction chaining may lead to such a situation. Chaining also opens the possibility for a designer employing function request shipping or transaction routing (though not DTP) to define a specific resource (including a transaction or terminal) as being in a remote CICS/VS system, and further define that resource in the remote system to be in yet another system. If the definition in the third system inadvertently specifies the resource to be in the first, any request for that resource will be routed to all three systems and will then deadlock until the specified timeout value expires, abending all the transactions. For these reasons great care should be taken during system definition to guard against unintended use or misuse of chained transactions.

Problem Determination

Application programs that make use of CICS/VS intercommunication facilities are liable to be subject to error conditions not experienced in single CICS/VS systems. The new conditions result from the intercommunication component not being able to establish a session with the requested system (for example, it is not defined to CICS/VS, or is not available).

In addition, some types of request may cause a transaction abend because invalid data is being passed to the CICS/VS function manager (for instance the file control program). Where the resource is remote, the function manager is also remote. Thus the transaction abend is suffered by the remote transaction. This in turn causes the local transaction to be abended with a transaction abend code of ATNI (for communication via VTAM) or AZI6 (for communication via MRO) rather than the particular code used in abending the remote transaction. However, the remote system sends the local CICS/VS system an error message identifying the reason for the remote failure. This message is sent to the local CSMT destination. Thus, if an application program uses SETXIT and user task abend exits to continue processing when abends occur while accessing resources, it will be unable to do so in the same way when those resources are remote.

Application programs not using the command level or DL/I CALL interfaces may inadvertently attempt to access, via function request shipping, resources defined as remote. In this case, the request will fail with a condition indicating that the resource is not defined to CICS/VS.

Trace and dump facilities will exist in both local and remote CICS/VS systems. When the remote transaction is abended, its CICS/VS transaction dump is available at the remote site to assist in locating the reason for an abend condition.

Applications to be used in conjunction with remote systems should be well tested to minimize the probability of failing when accessing remote resources. It should be remembered that a "remote test system" can actually reside in the same processor as the local system and so be tested in a single location where the transaction dumps from both systems, and the corresponding trace data, are readily available. The two transactions may be connected via MRO (for function request shipping and transaction routing) or the VTAM application-to-application facility (for function request shipping or DTP).

Detailed sequences and request formats for CICS/VS intercommunication may be found in the CICS/VS Diagnosis Reference Manual and CICS/VS Problem Determination Guide.

Recovery and Restart with Non-CICS/VS Systems

The cross-link exchanges used by CICS/VS to establish the state of the other system during recovery are defined by SNA. They are therefore independent of the nature of the remote system. CICS/VS follows the same recovery procedures whether the other system is CICS/VS or not.

Index

Each page number in this index refers to the start of the paragraph containing the indexed item.

CICS/VS System/Application Design Guide Printed in USA SC33-0068-2

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

Customer Information Control System/Virtual Storage (CICS/VS)
System/Application Design Guide

SC33-0068-2

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Number of your latest Technical Newsletter for this publication. . . . . .

Note: *Staples can cause problems with automated mail sorting equipment. Please use pressure-sensitive or other gummed tape to seal this form.*

Cut Along Dotted Line

If you want an acknowledgement, give your name and address below.

Name . . . . . .

Job Title . . . . . Company . . . . . .

Address . . . . . .

. . . . . Zip . . . . . .

Thank you for your cooperation. No postage stamp is necessary if mailed in the U.S.A. (Elsewhere, your IBM representative or IBM branch office will be happy to forward your comments.)

SC33-0068-2

Reader's Comment Form

Fold and tape Please do not staple Fold and tape

No postage necessary if mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 40 ARMONK, NEW YORK

Postage will be paid by addressee:

International Business Machines Corporation
Department 812HP
1133 Westchester Avenue
White Plains, New York 10604

Fold and tape Please do not staple Fold and tape

CICS/VS System/Application Design Guide Printed in USA SC33-0068-2

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601