

PUBLICATIONS REVISION
System 80
OS/3 Integrated Communications Access Method (ICAM) Technical Overview
UP-9744 Rev. 1

This Library Memo announces the release and availability of the *System 80 OS/3 Integrated Communications Access Method (ICAM) Technical Overview*, UP-9744 Rev. 1.

This overview is a standard library item (SLI). It is part of the standard library provided automatically with the purchase of the product.

The integrated communications access method (ICAM) is a symbiont that handles input and output between your program and input/output (I/O) devices tied directly to the central processor. ICAM isolates your program from the hardware, eliminating the problems that arise in physical I/O control. This manual is one of a series to guide you in programming and using ICAM with Operating System/3 (OS/3).

Changes to this document for Release 12.0 include the addition of System 80 model 15 and DCP channel support. All other changes in this document are corrections, deletions, or expanded descriptions applicable to items present in the software prior to this release.

Additional copies may be ordered through your Unisys representative.

**Destruction Notice:** This revision supersedes and replaces *Operating System/3 (OS/3) Integrated Communications Access Method (ICAM) Concepts and Facilities*, UP-9744, released on Library Memo dated February 1984. Please destroy all copies of UP-9744, its updates, and Library Memos.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists MBZ, MCZ, MMZ, M28U, and M29U	Mailing Lists MB00, MB01, and MBW (238 pages plus Memo)	Library Memo for UP-9744 Rev. 1
		RELEASE DATE: October 1988



**UNISYS**

**System 80  
OS/3**

**Integrated  
Communications  
Access Method (ICAM)**

**Technical Overview**

OS/3 Release 12.0

October 1988

Priced Item

Printed in U S America  
UP-9744 Rev. 1



**UNISYS**

**System 80  
OS/3**

**Intergated  
Communications  
Access Method (ICAM)**

**Technical Overview**

Copyright © 1988 Unisys Corporation  
All rights reserved.  
Unisys is a trademark of Unisys Corporation.

OS/3 Release 12.0

October 1988

Priced Item

Printed in U S America  
UP-9744 Rev. 1

The names, places and/or events used in this publication are not intended to correspond to any individual, group, or association existing, living, or otherwise. Any similarity or likeness of the names, places, and/or events with the names of any individual, living or otherwise, or that of any group or association is purely coincidental and unintentional.

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication should be forwarded, using the User Comments form at the back of this manual or remarks addressed directly to Unisys Corporation, to E/MSG Product Information, P.O. Box 500, M.S. E5-114, Blue Bell, PA 19424 U.S.A.

**PAGE STATUS SUMMARY**  
**ISSUE: UP-9744 Rev. 1**  
**RELEASE 12.0 Forward**

Part/Section	Page Number	Update Level	Part/Section	Page Number	Update Level	Part/Section	Page Number	Update Level
Cover								
Title Page/Disclaimer								
PSS	iii							
About This Overview	v thru ix							
Contents	xi thru xvii							
1	1 thru 3							
2	1 thru 43							
3	1 thru 35							
4	1 thru 33							
5	1 thru 46							
6	1 thru 7							
7	1 thru 13							
8	1 thru 7							
Appendix A	1 thru 7							
Glossary	1 thru 20							
Index	1 thru 6							
User Comments Form								
Back Cover								





# About This Overview

## Purpose

ICAM is flexible. It works with almost any kind of terminal, common carrier, or application. To be this flexible, ICAM has a large number of options. The main purpose of this overview is to describe these features so you can select the ones you need. This overview, however, is only one of the manuals you need to use ICAM. The communications documentation breaks down into three groups:

1. Documentation for selecting the communications hardware
2. Documentation for defining your network and creating the ICAM symbiont and operating ICAM
3. Documentation for writing or selecting the programs

Refer to "Related Product Information" for a list of these documents.

*Note: You must read this manual in order to understand the other ICAM manuals, which are how-to manuals and contain a minimum of theory.*

## Scope

This manual gives an overview of the facilities provided by ICAM, including the hardware supported, the types of programs supported (those written in basic assembly language (BAL), COBOL, and RPG II), and the services provided, such as polling, queuing, and buffering.

## Audience

The intended audience is the programmer with a basic knowledge of data communications and OS/3.

## How to Use This Overview

If you are going to use ICAM, you should read this overview.

If you are writing your own communications program using either the standard message control program (STDMCP), the direct data interface (DDI), or the communications physical interface (CPI), this overview is required in addition to the specified interface manual.

If you are using Unisys supplied software, like Information Management System (IMS), Distributed Data Processing (DDP), or Nine Thousand Remote (NTR), we recommend you read this overview for a background in communications.

## Organization

The information contained in this overview is presented in three parts. Part 1 describes ICAM and the software and hardware facilities that comprise the communications system. Part 2 describes the software modules that are common to two or more types of ICAM interfaces. Part 3 provides appendixes that present general reference data that can be of help in using this overview.

The structure of each part of the manual is explained as follows:

### **PART 1. BASIC DESCRIPTION**

#### **Section 1. Introduction**

This section describes what ICAM does and the documentation available for your use.

#### **Section 2. OS/3 Communications Software**

This section describes the structure of ICAM, the capabilities and limitations of each interface, and the functions of each of the several utility programs.

#### **Section 3. Communications Hardware**

This section describes the physical elements within the communications system, from the remote terminal to the interface with the system processor.

### **PART 2. ICAM SERVICES**

#### **Section 4. Line and Terminal Support**

This section describes the functions of the remote device handlers in servicing line and terminal protocol, controlling the flow of input and output, translating character codes, formatting messages, and reporting status and error codes.

#### **Section 5. Buffers and Queues**

This section describes the functions and use of line and network buffers, activity request packets, and input and output queues.

#### **Section 6. Message Processing Procedure Specification (MPPS)**

This section summarizes how to write a limited message processing routine within the ICAM network.

### **Section 7. DCA and DDP**

This section introduces distributed communications architecture (DCA) and how it provides a total system approach to communications. Also discusses support for DDP.

### **Section 8. Administrative Functions**

This section describes the facilities for developing reports about line and terminal usage, network buffer utilization, communications hardware facilities, and certain other communications-related system activities.

## **PART 3. APPENDIXES**

### **Appendix A. Coding Conventions**

This appendix describes the format and coding conventions for macroinstructions you use in your ICAM network definition and communications programs.

### **Glossary**

The glossary defines ICAM and communications terms.

## **Related Product Information**

As one of a series, this manual is designed to guide you in programming and using the OS/3 integrated communications access method. Depending on your need, you may wish to refer to one of the other ICAM manuals. Complete manual names, their ordering numbers, and a general description of their contents and use follow.

*Note:* Throughout this overview, when we refer you to another manual, use the version that applies to the software level at your site.

### ***Integrated Communications Access Method (ICAM) Utilities Programming Guide (UP-9748)***

This guide describes the following ICAM utilities: ICAM device emulation system, remote batch processing; journal utility, COBOL message control system, single-line communications adapter (SLCA) dump routine, ICAM trace facility, ICAM edit dump, UNIX<sup>®</sup> system access module (UNXSAM) and DCP/Telcon load facility.

### ***Integrated Communications Access Method (ICAM) Operations Guide (UP-9745)***

This guide describes how to define an ICAM network, submit it to system generation, and load and operate the resulting ICAM symbiont. Many sample network definitions are provided to make it easier to define your ICAM network.

---

UNIX is a registered trademark of AT&T Information Systems.

### ***Integrated Communications Access Method (ICAM) Standard MCP (STDMCP) Interface Programming Guide (UP-8550)***

The standard interface is a logical interface that provides a general communications capability with message queuing and a message processing capability. This guide provides all of the macroinstructions, programming requirements, and terminal information you need for the standard interface.

You will need this guide only if you are writing your own communications program. Programs that use the standard interface directly must be coded in basic assembly language, and your system must include the OS/3 assembler.

If you write your program in COBOL, you will require the COBOL message control system utility. You won't need this guide because the utility converts your COBOL statements to instructions that this interface recognizes.

### ***Integrated Communications Access Method (ICAM) Direct Data Interface (DDI) User Guide (UP-8549)***

The direct data interface commonly supports ICAM utility programs and programs written in the RPG II language. If you are using an ICAM utility only, or your program is written in RPG II, you won't need this guide because the utility programs and the RPG II compiler automatically convert any requests by your program to the proper instructions needed to work with this interface.

The direct data interface also enables you to write your own specialized communications program. If you do this, you must take care of your own message buffering and queuing. If you write a program to interface directly with the direct data interface, it must be written in basic assembly language, and your system must include the OS/3 assembler.

### ***Integrated Communications Access Method (ICAM) Communications Physical Interface (CPI) Programming Guide (UP-9746)***

The communications physical interface requires the least amount of main storage, but it also provides a minimum amount of support. If you use this interface, you must have considerable knowledge of data communications because your program must initialize the hardware, format all output messages using the appropriate protocol, perform any required translations, acknowledge and process all input messages, and perform all error detection and recovery procedures. In addition, your program must be written in BAL and your system must include the OS/3 assembler.

### ***Integrated Communications Access Method (ICAM) Programming Reference Manual (UP-9749)***

This reference summarizes the information found in the other ICAM manuals. It also describes the optional Message Processing Procedure Specifications (MPPS). No introductory information or examples are given; however, it is a useful document when you are familiar with ICAM and you need a quick reference to macroinstructions, formats, and tables.

***Integrated Communications Access Method (ICAM) Remote Terminal Processor (RTP) Programming Guide (UP-10047)***

The remote terminal processor is a data communications program that permits your Unisys System 80 processor to function as a remote job entry terminal to one or more IBM<sup>®</sup> host processors. Using the Unisys OS/3 integrated communications access method (ICAM) software, the remote terminal processor enables you to:

- Send jobs to an IBM host
- Transmit and receive files on tape, punched cards, or diskette
- Send messages to the central site
- Receive output data and console messages from the IBM host

Remote terminal processor operations are directed from the OS/3 system console.

***NTR Utility Programming Guide (UP-9502)***

The NTR system utility allows a System 80 processor to operate as a remote job entry/batch terminal to a Unisys Series 1100 system using ICAM. The utility permits operation of reader, printer, and punch device-dependent files. It also supports user-own-code tasks to process device-independent files (e.g., tape, disk, and paper tape).

***OS/3 - UNIX Operating System Connectivity Operating Guide (UP-14207)***

OS/3 - UNIX connectivity lets the hardware running your OS/3 operating system and the hardware running your UNIX operating system communicate with each other to perform a variety of functions. This guide presents an overview of the connectivity process and the program products that you can use to connect your OS/3 and UNIX operating systems.

---

IBM is a registered trademark of International Business Machines Corporation.



# Contents

About This Overview .....	v
---------------------------	---

## PART 1. BASIC DESCRIPTION

### Section 1. Introduction

### Section 2. OS/3 Communications Software

<b>2.1. ICAM - The Software Program .....</b>	<b>2-1</b>
2.1.1. ICAM Internals and Interfaces .....	2-1
2.1.2. Global versus Dedicated Networks .....	2-5
Dedicated Networks .....	2-6
Global Networks .....	2-7
2.1.3. Static and Dynamic Sessions .....	2-10
2.1.4. Generating Your ICAM Symbiont (Load Module) .....	2-13
2.1.5. Yielding Program Control .....	2-13
2.1.6. How to Activate a Communications Program and Pass It a Message .....	2-13
<b>2.2. Communications User Programs .....</b>	<b>2-14</b>
2.2.1. User-Written Program Interfaces .....	2-15
Standard Interface and Direct Data Interface Programs .....	2-15
Writing Your Program in Assembly Language .....	2-18
Sending and Receiving Messages with the Standard Interface ...	2-20
Sending and Receiving Messages with the Direct Data Interface .....	2-23
COBOL Programs .....	2-26
RPG II Programs .....	2-27
2.2.2. Programs and Products Supplied by Unisys .....	2-29
Information Management System (IMS) .....	2-30
Remote Batch Processing (RBP) Utility .....	2-34
ICAM Device Emulation System (IDES) Utility .....	2-36
NTR Utility .....	2-38
IBM 3270 Emulator .....	2-40
OS/3 Remote Terminal Processor .....	2-40
Journal Utility .....	2-41
Connecting OS/3 to a UNIX System .....	2-42
Connecting OS/3 to a MAPPER 5 System .....	2-42

**Section 3. Communications Hardware**

<b>3.1. Basic System 80 Communications System</b>	3-1
<b>3.2. The Terminals</b>	3-2
3.2.1. Hardware - What Is a Terminal?	3-3
3.2.2. How Are Terminals Used?	3-7
3.2.3. Interface Characteristics - How Terminals Communicate	3-9
Message Formatting	3-9
Communications Direction	3-13
Synchronizing Transmission	3-16
Line Control	3-18
Additional Terminal Interface Characteristics	3-20
<b>3.3. Communications Lines</b>	3-20
3.3.1. Dedicated Circuits - Dedicated Lines and VLINEs	3-22
3.3.2. Switched Lines	3-24
3.3.3. Public Data Networks	3-26
Circuit-Switched Public Data Networks	3-27
Packet-Switched Public Data Networks	3-29
<b>3.4. Single Line Communications Adapters</b>	3-33
<b>3.5. DCP Channel</b>	3-35

**PART 2. ICAM SERVICES**

**Section 4. Line and Terminal Support**

<b>4.1. Remote Device Handlers</b>	4-1
<b>4.2. Line Connections</b>	4-4
<b>4.3. Terminal Polling</b>	4-8
4.3.1. Polling Groups	4-9
4.3.2. Polling Interval	4-11
4.3.3. Polling Algorithms	4-13
Polling with Buffered Interactive Terminals	4-14
Polling with Unbuffered Interactive Terminals	4-20
<b>4.4. Input and Output Devices</b>	4-21
<b>4.5. Formatting Your Data</b>	4-23
4.5.1. Device Dependent Control Characters	4-24
4.5.2. Device Independent Control Expressions	4-24
Hexadecimal Notation	4-26
DICE Macroinstructions	4-27
4.5.3. DICE Summary	4-29
<b>4.6. Format Edit</b>	4-30
<b>4.7. Translate Tables</b>	4-31
<b>4.8. Status and Error Codes</b>	4-31
4.8.1. Input Error Notification	4-31
4.8.2. Output Error Notification	4-32
4.8.3. Terminal Statistics	4-33
<b>4.9. Output Delivery Notification Request</b>	4-33



**Section 5. Buffers and Queues**

<b>5.1. General</b> .....	5-1
<b>5.2. Line Buffers and VLINE Buffers</b> .....	5-3
5.2.1. Line Buffers .....	5-3
5.2.2. VLINE Line Buffers .....	5-5
<b>5.3. Activity Request Packets</b> .....	5-7
<b>5.4. Network Buffers</b> .....	5-9
<b>5.5. Queues</b> .....	5-22
5.5.1. Output Queues .....	5-33
5.5.2. Input Terminal Queues .....	5-34
5.5.3. Process Files .....	5-35
5.5.4. Locap Files .....	5-38
Queue Arrangement .....	5-40
5.5.5. Using Queues .....	5-41
Obtaining Messages from Terminals .....	5-41
Sending Messages from Programs .....	5-43
Distribution Lists .....	5-45
<b>5.6. Dynamic Buffer Pool Expansion</b> .....	5-46

**Section 6. Message Processing Procedure Specification (MPPS)**

<b>6.1. General</b> .....	6-1
<b>6.2. Message Header Examination and Manipulation</b> .....	6-2
<b>6.3. Error Recovery</b> .....	6-4
<b>6.4. Message Routing</b> .....	6-6
<b>6.5. Miscellaneous Functions</b> .....	6-7

**Section 7. DCA and DDP**

<b>7.1. Introduction TO DCA</b> .....	7-1
<b>7.2. DCA Concepts</b> .....	7-2
<b>7.3. Single-Node and Multinode Global Networks</b> .....	7-6
<b>7.4. ICAM Support For Distributed Data Processing</b> .....	7-10

**Section 8. Administrative Functions**

<b>8.1. Overview</b> .....	8-1
<b>8.2. The Journal Utility - Report Segment</b> .....	8-1
<b>8.3. The Journal Utility - Restart Segment</b> .....	8-1
<b>8.4. Online Diagnostic Facilities</b> .....	8-2
<b>8.5. System Activity Monitor</b> .....	8-2
8.5.1. Input Message Rate (CIMR) .....	8-3
8.5.2. Output Message Rate (COMR) .....	8-3
8.5.3. Number of Interrupts (CINT) .....	8-4
8.5.4. Number of Sense Commands (CSEN) .....	8-4
8.5.5. Number of Error Commands (CERR) .....	8-4

8.5.6.	Number of No-Traffic Responses (CNOT) .....	8-4
8.5.7.	Rate of Poll Interrupts (CPOL) .....	8-4
8.5.8.	Rate of Bytes Transmitted (CBYT) .....	8-5
8.6.	<b>ICAM Trace Facility</b> .....	8-5
8.7.	<b>ICAM Edit Dump</b> .....	8-6

**PART 3. APPENDIXES**

**Appendix A. Coding Conventions**

A.1.	<b>Types of Macroinstructions</b> .....	A-1
A.2.	<b>Declarative and Imperative Macroinstructions</b> .....	A-1
A.2.1.	Positional Operands .....	A-2
A.2.2.	Keyword Operands .....	A-3
A.3.	<b>Macroinstruction Coding Conventions</b> .....	A-3
A.4.	<b>S-Type Macroinstructions</b> .....	A-5
A.4.1.	L-Form S-Type Macroinstruction .....	A-5
A.4.2.	E-Form S-Type Macroinstruction .....	A-6
A.4.3.	SD-Type Macroinstruction .....	A-6

**Glossary**

**Index**

**User Comments Form**

# Figures

1-1.	Functions of Each Part of an OS/3 Communications System .....	1-3
2-1.	The ICAM Internals .....	2-3
2-2.	The ICAM Interfaces .....	2-4
2-3.	Standard Interface Communications System .....	2-16
2-4.	Direct Data Interface Communications System .....	2-17
2-5.	Typical COBOL-CMCS/ICAM Environment .....	2-27
2-6.	IMS in an ICAM Environment .....	2-32
2-7.	Remote Batch Processing System .....	2-35
2-8.	ICAM Device Emulation System .....	2-37
2-9.	NTR Utility System .....	2-39
2-10.	Connecting OS/3 to a MAPPER 5 System .....	2-43
3-1.	Basic System 80 Communications System .....	3-2
3-2.	UTS 4000 System Connected to System 80 .....	3-6
3-3.	Common Message Formats .....	3-12
3-4.	Public Data Network Basic Configuration .....	3-26
3-5.	Circuit-Switched Public Data Network .....	3-28
3-6.	Packet-Switched Public Data Network .....	3-30
3-7.	Input/Output Microprocessor Interface within System 80 .....	3-33
5-1.	Comparison of Standard and Direct Data Interfaces .....	5-2
5-2.	Activity Request Packet Statistics in a System Dump .....	5-8
5-3.	Buffers Statistics in a System Dump .....	5-19
5-4.	Statistics Area in a System Dump for Network Buffers .....	5-20
5-5.	Message Characteristics .....	5-32
7-1.	Example of Communications System and Session Path .....	7-5
7-2.	Typical Host Processor Using a Single-Node Global Network .....	7-6
7-3.	Multinode Global Network .....	7-8
7-4.	Multinode DCA Global Network Using Telcon .....	7-9



# Tables

3-1.	ICAM-Supported Terminals and Workstations .....	3-8
3-2.	ICAM Circuit-Switched Public Data Network Support .....	3-29
3-3.	ICAM Packet-Switched Public Data Network Support .....	3-32
3-4.	Single Line Communications Adapters .....	3-34
4-1.	DICE Codes and Functions .....	4-26
4-2.	DICE Macroinstructions and Their Functions .....	4-28
8-1.	System Activity Monitor Available Data .....	8-2



# Section 1

## Introduction

The integrated communications access method (ICAM) is a symbiont that becomes an extension of the supervisor to handle input and output between your program and terminals tied to the central processor by a communications system. It supports multiple user programs; however, throughout this manual, we use the singular (your program) even though you may have several programs operating in the same ICAM environment at the same time. In its function, ICAM is like data management, which handles input and output between your program and input/output (I/O) devices tied directly to the central processor. Like data management, ICAM isolates your program from the hardware, eliminating the problems arising in physical I/O control.

Consider what your program must do to write a record into a disk file without data management. It must write the record in the same format as the other records in the file, send an operation code to the correct disk unit (telling the disk drive what to do and exactly where to put the record), then examine the sense and status bytes returned by the disk unit to see whether errors occurred. If they did, your program must handle them. With data management, of course, all this is taken care of for you.

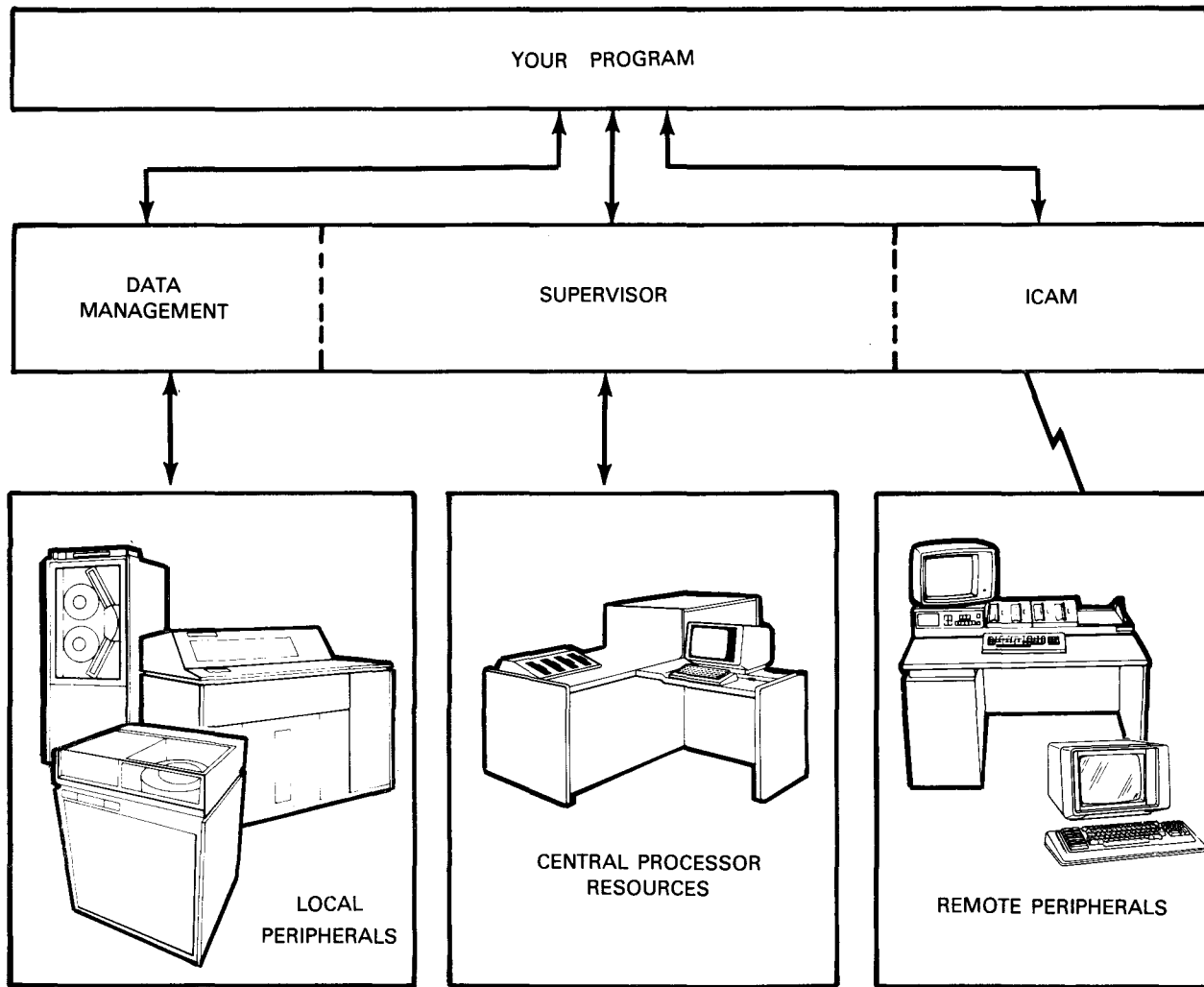
A program writing a message to a terminal without ICAM has all the problems of a program writing a record to a disk unit without data management, plus some others. For example, your program must honor the protocols for exchanging messages between the central processor and the terminal. Different terminals use different protocols (for example, UNISCOPE,<sup>®</sup> universal data link control (UDLC), and binary synchronous communication (BSC)) to send messages. With ICAM, all this is taken care of for you.

ICAM also does some things that data management doesn't do. For example, when your program works with data management and you want to read a record, data management reads it in. You process the record before getting the next one. Data never arrives faster than your program can handle it.

That's not true in a communications system. Your program controls output, but has no control over input (except when you use the ICAM direct data interface). Most of the time, messages come in about as fast as your program can process them. Sometimes, however, a long time goes by between messages. ICAM allows your program to suspend execution when no messages are arriving; otherwise, your program would have to loop, wasting processor time, while it waits for a message.

---

UNISCOPE is a registered trademark of Unisys Corporation.



The supervisor isolates your program from the hardware by performing the tasks that actually control the hardware. Your program "uses" system resources by requesting the supervisor to work with the hardware.

At other times, messages arrive faster than your program can process them. ICAM provides buffers to store the messages until your program can process them.

At its simplest, this is ICAM: a symbiont program that controls communications input and output, temporarily storing messages when needed. Figure 1-1 shows a complete OS/3 communications system and the functions of each element.



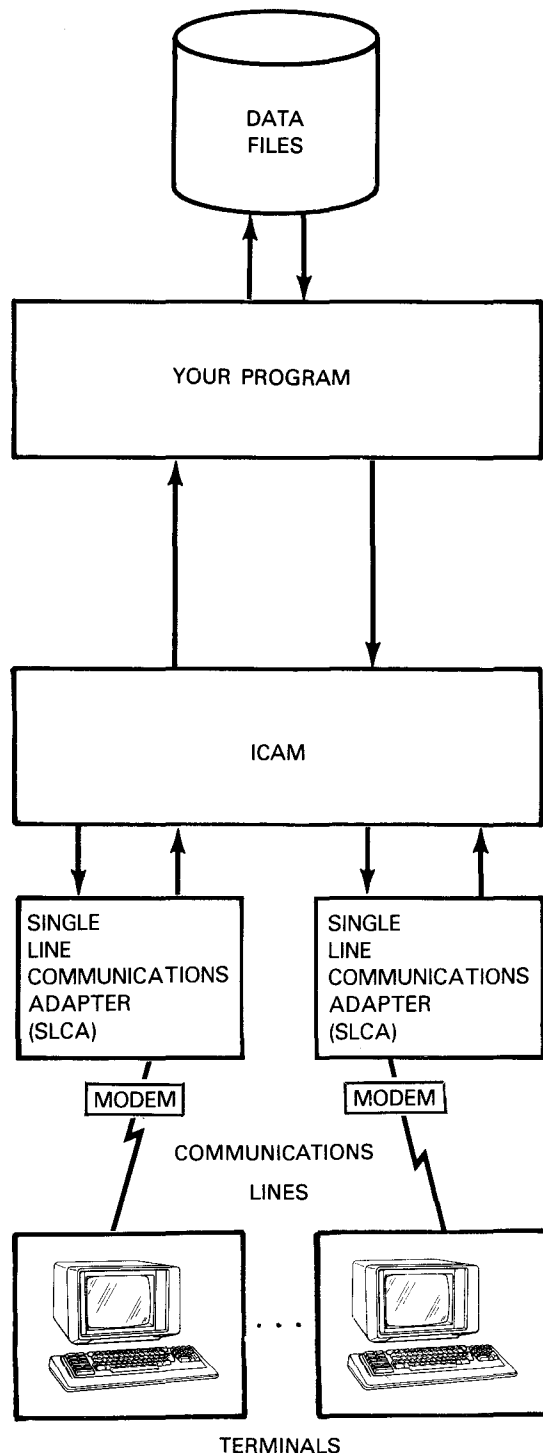


Figure 1-1. Functions of Each Part of an OS/3 Communications System

The communications user program - your program - does all the message processing. (Your program can be one you write or one supplied by Unisys, such as IMS, DDP, and interactive services.) It reads the messages, processes the data in them, updates files, and creates output messages. It is similar to other applications programs, with the additional capability of working with remote terminals. Because of its additional capability, it has the additional responsibilities of (1) working with ICAM, (2) handling communications-related errors, and (3) providing the security for your data files.

Thus, basically, ICAM is a message handler; it can't read messages, process them, or update files. Your program does that; but ICAM provides you with the macroinstructions to allow your program to request the I/O services it needs and interface the communications software with the communications hardware.

Single line communications adapters are hardware devices that interface the central processing unit with communications lines and also free ICAM of some of the communications-related tasks it would otherwise have to perform.

The communications lines are any communications medium - telephone lines, microwave relays, satellite, etc - that transfers messages from the communications adapter to your terminals and vice versa.

The terminals receive and transmit data. The data can be a count of items passing by on an assembly line, an airline reservation, or 1000-punch card images.



## Section 2

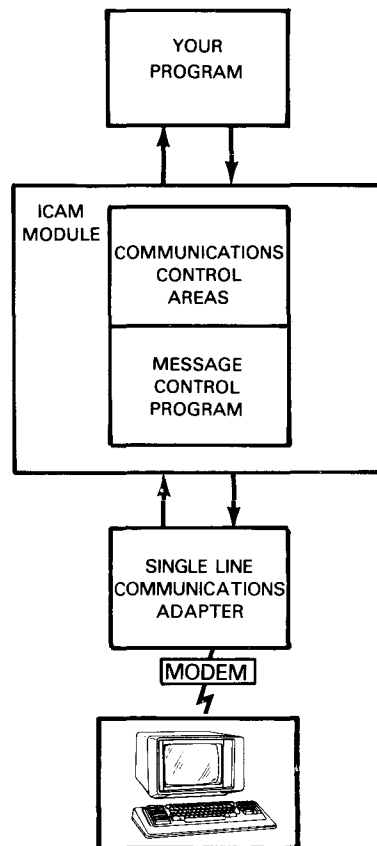
# OS/3 Communications Software

### 2.1. ICAM - The Software Program

The following paragraphs present an overview of ICAM, how it is organized, the two basic kinds of networks, dedicated and global, and why you would select each type. Also discussed is the concept of static and dynamic sessions used with global networks. Finally, we describe how to generate an ICAM symbiont, and how your program works with it.

#### 2.1.1. ICAM Internals and Interfaces

So far, we've treated ICAM as a mysterious entity handling input and output between your programs and terminals. Now let's look at what ICAM does and how it does it.



It shows the same basic elements represented in Figure 1-1 except that the ICAM module is split into two parts: the message control program and communications control areas. The message control program contains the code that does the processing. This code handles the communications input and output operations, the remote device handlers, and the message processing routines. The communications control area is a set of tables and work areas that support a single network. In this context, a network is a group of lines and their terminals that may be a subset of all your lines and terminals. For example, you might have a network of terminals in your retail stores and another network in your warehouses. Each network would have its own communications control area and the terminals in that network would be controlled by a separate program (Figures 2-1 and 2-2). (See 2.1.2 on global versus dedicated networks for more on the uses of communications control areas.)

Regardless of how many communications control areas are in ICAM, there is only one message control program. Each network is supported by an interface, like the one shown in Figure 2-1, that's a composite of its communications control area and the message control program. In Figure 2-1, the line buffers, queues, and main storage network buffers are in the communications control area; and the communications physical input/output control system, remote device handlers, and message processing routines are in the message control program.

ICAM has four kinds of interfaces. Figure 2-2 shows their elements, and the following discusses their use:

1. Standard message control program interface

Called the *standard interface* throughout the rest of this manual, this is the most sophisticated interface offered by ICAM. It completely isolates your programs from the physical aspects of communications. ICAM handles input and output; your program asks ICAM for input messages and passes its output messages. This is the easiest interface to write programs for because almost all of your program is devoted to message processing and very little is devoted to communications functions. The ICAM needed for a standard interface, however, is larger than that needed for other interfaces. You can write your program in either basic assembly language or COBOL. If you are programming in basic assembly language, see the *ICAM Standard MCP (STDMCP) Programming Guide* (UP-8550). If you are programming in COBOL, see the *ICAM Utilities Programming Guide* (UP-9748).

2. Transaction control interface

This interface is similar to the standard interface. The big difference between them is that the transaction control interface supports the information management system (IMS) supplied by Unisys, while the standard interface is intended to support programs you write. Since this interface operates like the standard interface and its primary function is to support IMS, there is little discussion of this interface in this manual.

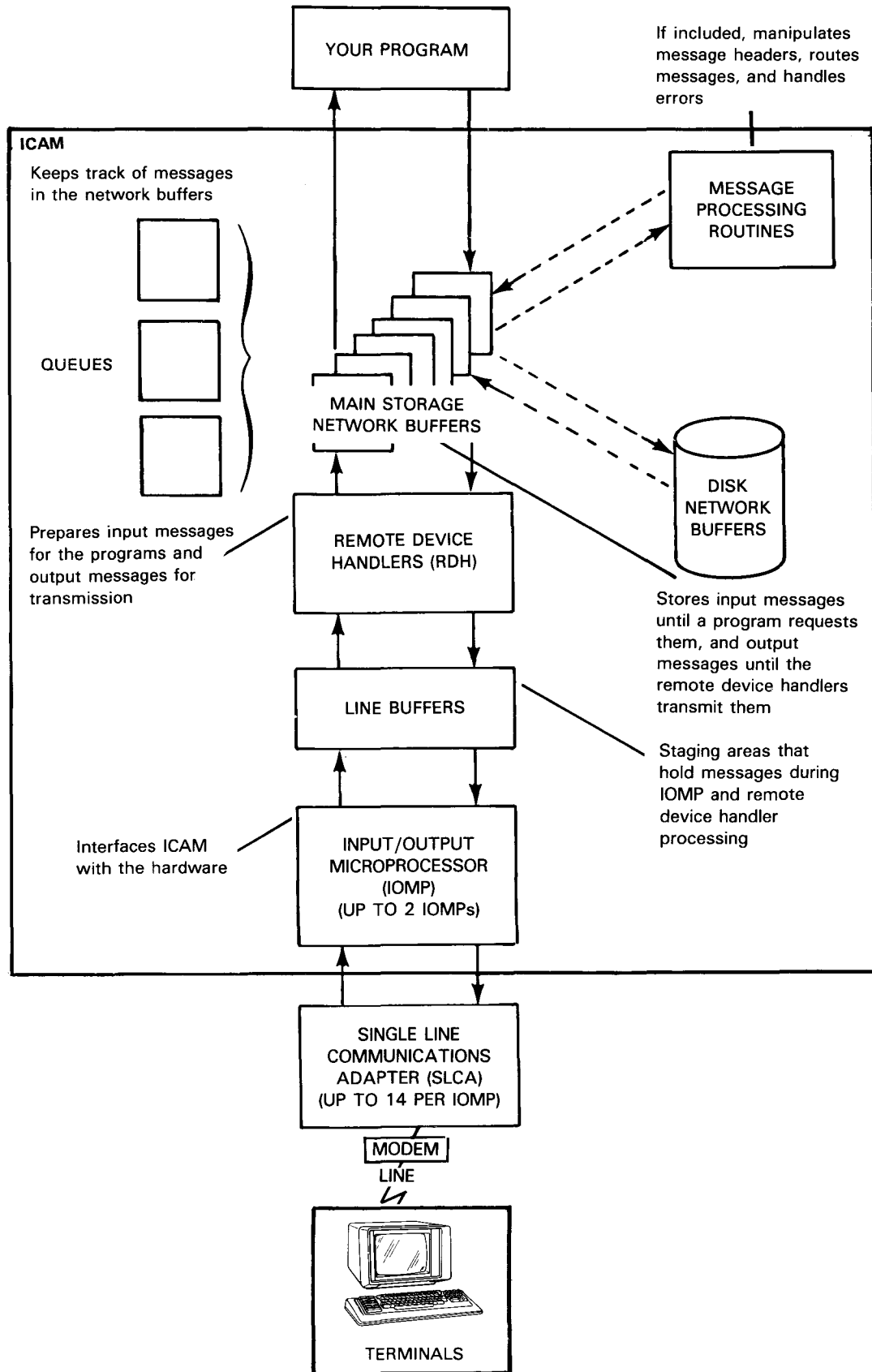


Figure 2-1. The ICAM Internals

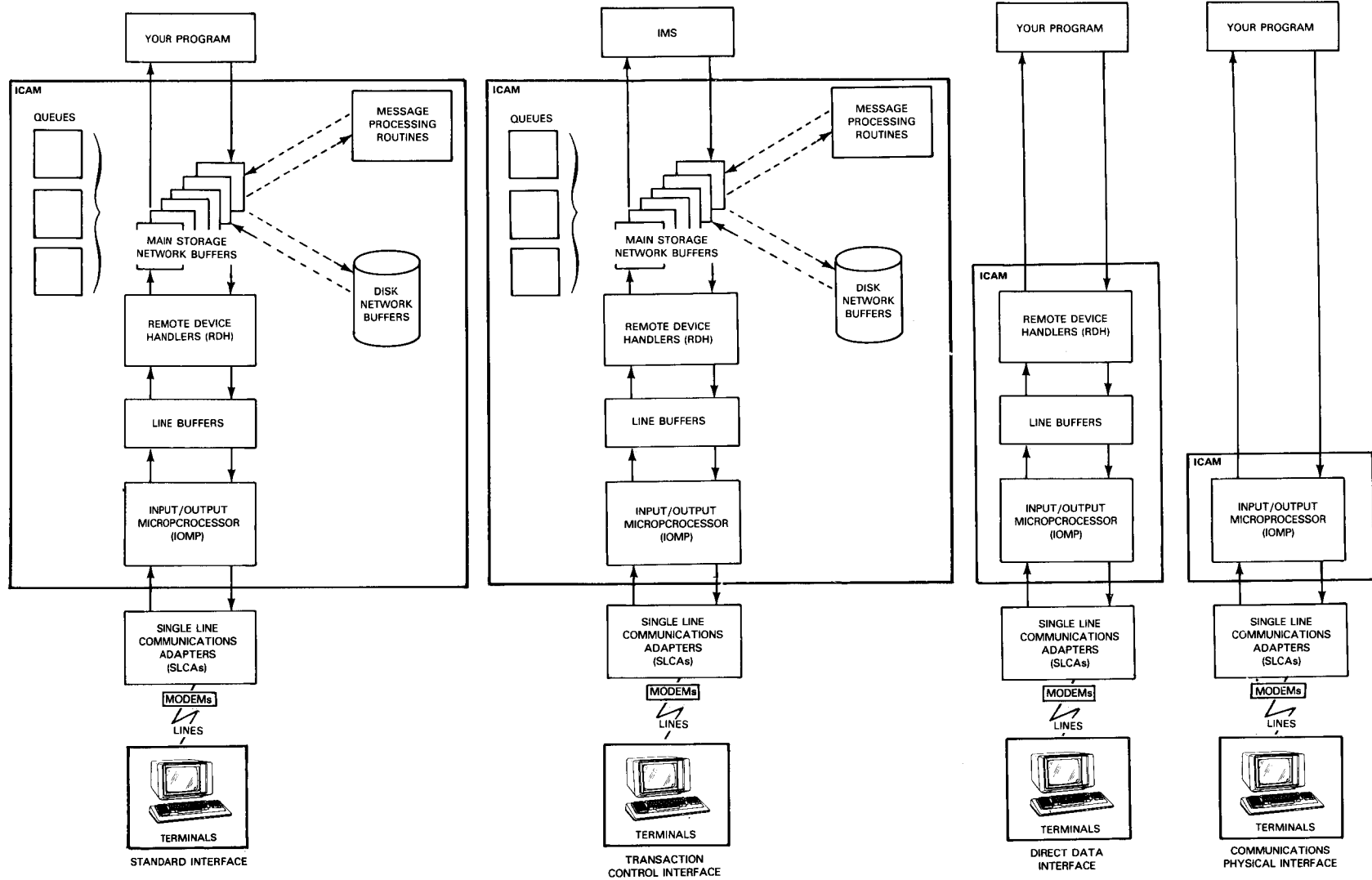


Figure 2-2. The ICAM Interfaces

### 3. Direct data interface

This interface is a minimum configuration interface that still handles the physical aspects of communications, but your program must direct ICAM activity. For example, your program must request ICAM to solicit input from the terminals, must request ICAM to send output to the terminals, and must direct ICAM recovery if an error occurs. Direct data programs are harder to write than standard interface programs because much of the program code directs ICAM operations. But the direct data program has more control over the operation of your communications system. Use the direct data interface when:

- You have special requirements that ICAM doesn't meet. With the direct data interface, you can usually write a program that provides the special processing that ICAM doesn't. The program is written in assembly language.
- You want to write a program in RPG II. (Note, that when you write your program in RPG II, you don't use the direct data interface macroinstructions - the RPG II compiler converts your instructions to the required direct data interface instructions.)
- You use a utility requiring the direct data interface.

Details on writing an assembly language program using this interface are in the *ICAM Direct Data Interface (DDI) User Guide* (UP-8549).

### 4. Communications physical interface

This interface provides you with the communications physical input/output control system that allows your program to work with the single line communications adapters. Your program has to provide all the functions described in this manual that ICAM provides with the other interfaces. Use the communications physical interface only if (1) you know a great deal about communications programming and (2) you have requirements that cannot be met by any of the other interfaces. This manual does not describe the communications physical interface because the knowledge needed to write a program for it is beyond its scope. Programs are written in basic assembly language at the physical machine level. If you decide to use this interface, see the *ICAM Communications Physical Interface (CPI) Programming Guide* (UP-9746).

## 2.1.2. Global versus Dedicated Networks

The basic difference between dedicated and global networks is that, in a dedicated network, the lines and terminals are dedicated (permanently assigned) to one program at a time. A global network permits multiple programs to share the resources of a network (lines, terminals, and files) and allows you to statically or dynamically alter resource assignments.

You should use a dedicated network when using only a single program or when using more than one program if:

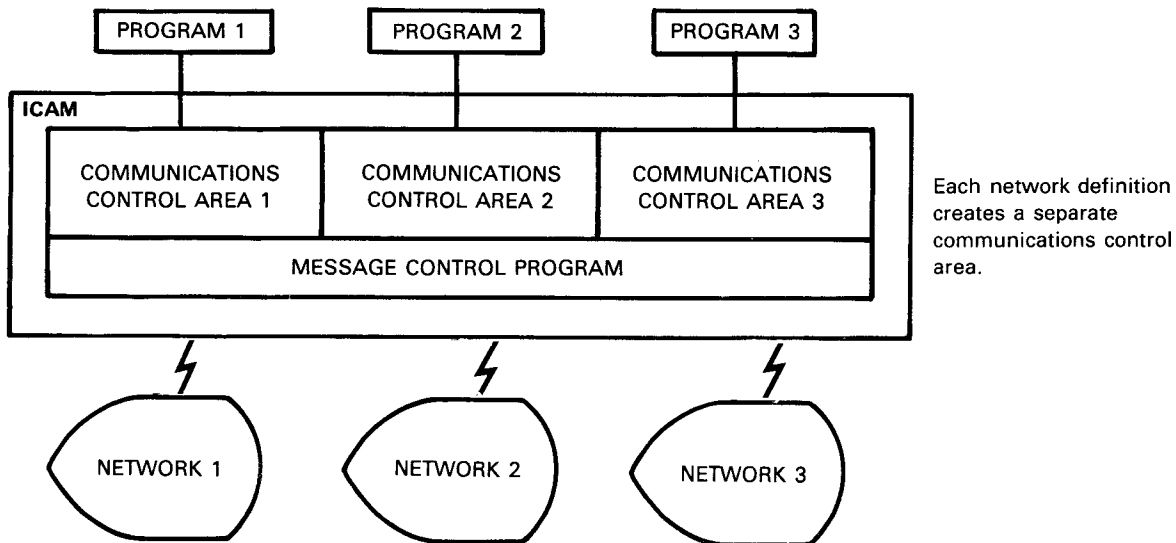
- your programs use specific networks of terminals and never need to share a network of terminals (at the same time); and
- your communications system has just one central processor.

Use a global network when:

- your programs need to share the same terminals; or
- your communications system has more than one central processor.

### Dedicated Networks

A dedicated network is one where lines and their terminals are dedicated to one program at a time. If you have more than one program executing at a given time, you define separate networks for each program:



Each network typically has terminals associated with a particular application. Network 1 could be terminals for inventory control; network 2 could be terminals for sales order entry; network 3 could be terminals for production control. Each network can be a distinct set of lines and their terminals, or the networks can overlap with lines and their terminals defined in two or more networks. However, only one program at a time can use a line. But one program can use a line and its terminals for a while and then release it for other programs to use. For example, if an inventory program receives its data before 8 o'clock each morning, the terminals it uses can be used by a production control program for the rest of the day.



Dedicated networks place these restrictions on your communications systems:

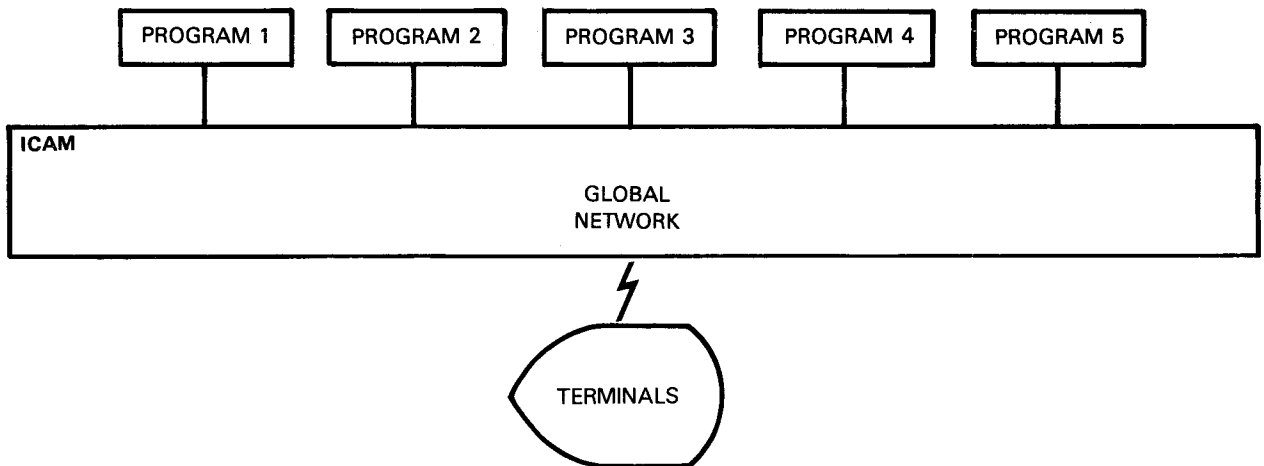
- Programs cannot share lines and their terminals at the same time.
- You cannot have two central processors in your communications system.

### Global Networks

ICAM's global network facility provides all of the services available to dedicated network user programs and, in addition, it provides the following features:

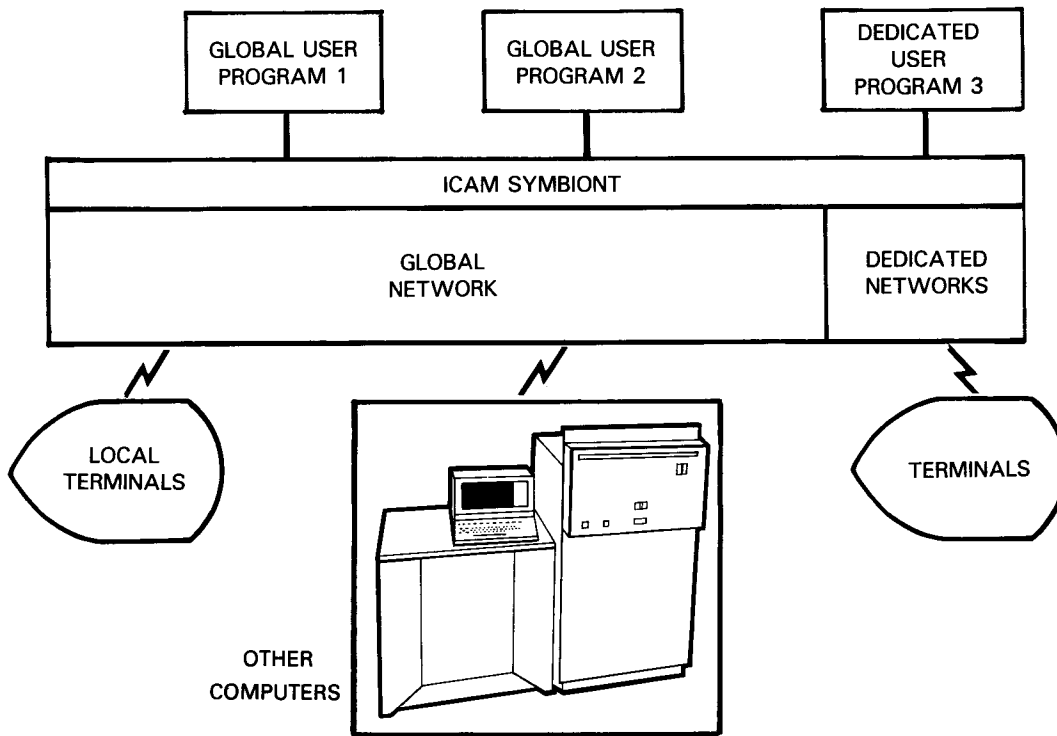
- Programs using the global network in the same computer can send messages to each other.
- Programs in the same computer can send and receive messages from any in the network.
- Programs in different host computers can send messages to each other.

The following global network allows multiple programs to concurrently use the same network and hence the same terminals.



Because programs use the same global network, they share its lines and terminals. Any program can receive or send messages to any terminal in the global network regardless of how many other programs are also using that terminal. Your program can also send messages to other programs using the same global network.

You may also concurrently operate one or more dedicated ICAM networks in the same ICAM symbiont with a global network. The global network may also support communications with other computers.

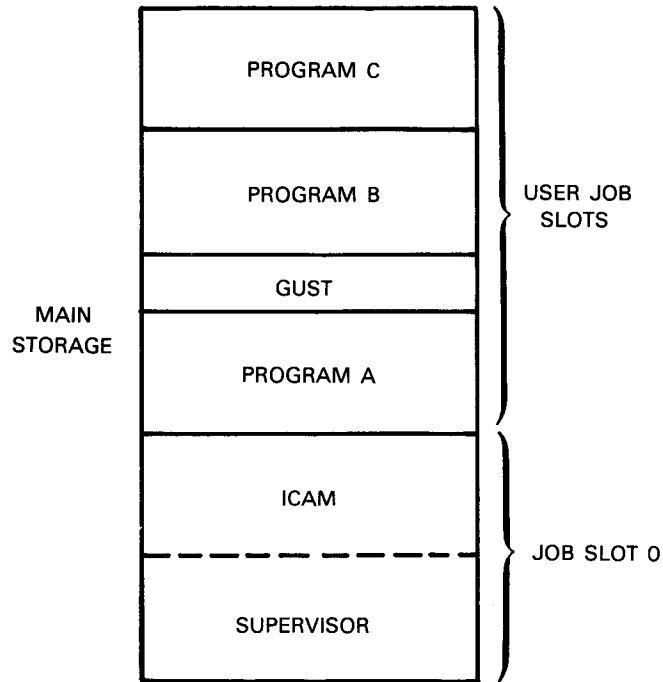


In networks with multiple central computers, programs in your computer are able to send and receive messages from programs in the other computers and from terminals connected to other computers.

If you can use dedicated networks, you should. Generally, they're simpler than global networks and use less main storage.

A major difference between global and dedicated networks is how the status of communications lines and terminals is handled, and how the ICAM network is loaded. In a dedicated network, this is done by the using program. In a global network, it is performed by a special program called the global user service task (GUST). GUST controls which ICAM network is to be loaded. It asks the operator for the name of the network to be loaded and activates the lines used by the global network.

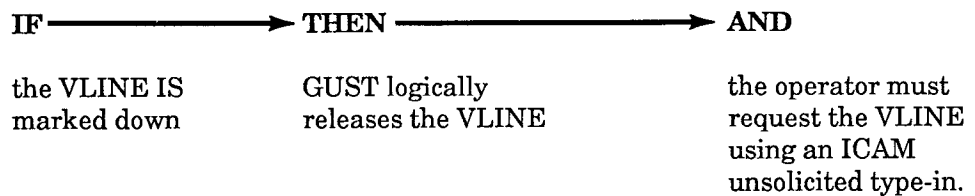
Unlike the rest of ICAM, which is part of the supervisor and runs in job slot zero, GUST is a separate program running as a user job in a user job slot:



When you bring up your global network, the ICAM symbiont is loaded first, and then GUST. After GUST is loaded, it sends a message to the system console asking which global network it is to request. After the console operator supplies the name, GUST requests the network and initializes ICAM.

After GUST requests the network name, it asks the names of the lines it is to activate. Any line not requested at this time must be brought up using an ICAM unsolicited type-in to mark up a line. At this point, your programs can attach themselves to ICAM to receive and send messages.

GUST also handles recoveries from VLINE down conditions:



A VLINE is marked down whenever a protocol error occurs, the other host computers do not respond, or a hardware error occurs.

Any time after GUST is brought up, the console operator activates local lines via ICAM operator unsolicited type-ins. Once a line is connected, any program attached to the global network can send and receive messages from terminals on that line. GUST handles recoveries from local line-down conditions as follows:

<b>IF</b> →	<b>THEN</b> →	<b>AND</b>
line is logically marked down	GUST does nothing	the line is marked up as soon as a terminal on the line responds to a poll.

<b>IF</b> →	<b>THEN</b> →	<b>AND</b>
the line is physically marked down	GUST logically releases the line and rerequests it	if the line request is unsuccessful, the operator is notified. He or she can rerequest the line at any time.

**OR**

if the line request is successful, the line is brought back up. The console operator is not notified that the line was down.

When GUST is shut down by the system operator, it releases the network and sends an ICAM cancel message to all programs attached to the network.

### 2.1.3. Static and Dynamic Sessions

A session is a communications path between two end users. End users are: a locap file (a local application file representing your program), a process file, or a terminal. (A process file is a 3-queue structure you define in your ICAM network that you can use to temporarily store messages.) Sessions apply only to global networks.

1. Static sessions

A static session is a permanent communications path between two end users, and you specify them in your network definition using a SESSION macro. Their advantages are that they save you some connect time over dynamic sessions, and you don't need to provide session establishment processing in your program. They are also permanent for the life of a network definition. However, you may not want to use them because, depending on the type of end user, that end user may become unavailable to any other end user. To decide, consider the following:

- Only one session at a time can be established with a terminal. Therefore, if you define a static session to a terminal, you cannot also establish a dynamic session for that terminal to communicate with another end user.

- Two sessions can be established to a process file. Therefore, if you define one static session to a process file, you can still establish a dynamic session with that process file. Any combination of sessions can be specified - one static and one dynamic, both static, or both dynamic.
- Any number of sessions can be specified to a locap file - in other words, to a program.
- If you use interactive services, you cannot use static sessions. You must use dynamic sessions.

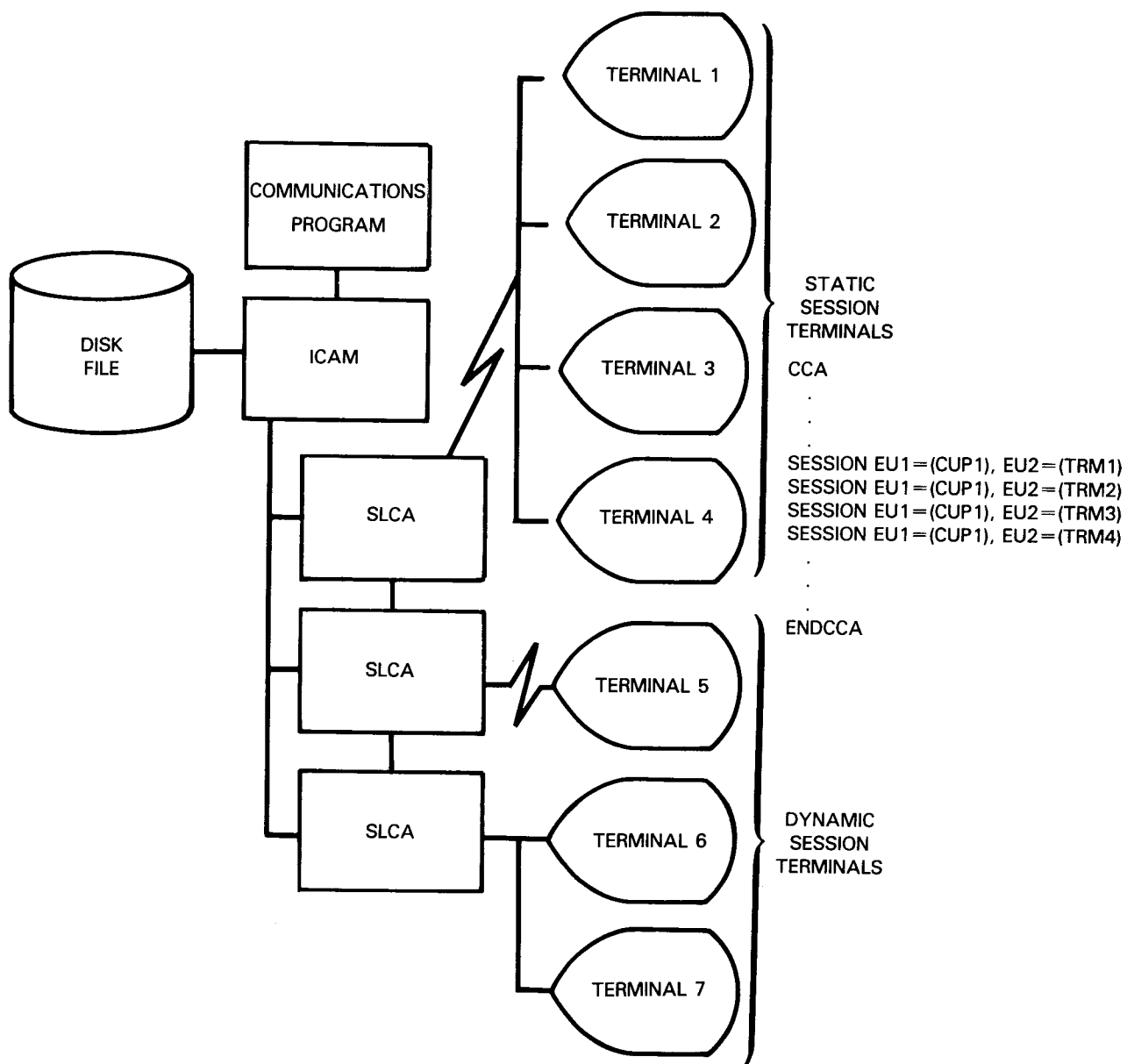
## 2. Dynamic sessions

Dynamic sessions are temporary communication paths established by ICAM due to a request by your program or a terminal user. Dynamic sessions can be established from a terminal to another end user (i.e., a locap file, process file, or terminal), or from your program to any end user. Dynamic sessions are always required with distributed communications architecture (DCA) networks.

Your program initiates and terminates dynamic sessions by issuing a session establishment (SESCON) macro to ICAM. Terminal users request dynamic sessions by means of session establishment (\$\$SON) commands. These commands name the end user with which communications need to be established, as well as the identifier of the requesting terminal; ICAM establishes the session path if possible. If the request for session establishment is directed to your program, ICAM interrogates your program by means of messages called control datagrams to see whether your program is willing to accept the session. If so, the session path is established. If your program rejects the session establishment request, the session is not established.

You incorporate the capability for dynamic session establishment into ICAM by specifying the GAWAKE operand in the ICAM CCA network definition macro. Your program must then issue a GAWAKE macro (TYPE=INPUT) to permit it to be activated (awakened), with control datagrams passed to it; this enables your program to accept or reject sessions.

Depending on the application, sometimes it is advantageous to have a global network supporting both dynamic and static session terminals. Look at the following example:



Here, terminals 1 through 4 are assigned (using SESSION statements) as static session terminals and are permanently tied up in our network. On the other hand, terminals 5 through 7 are dynamic session terminals that are used online with the network or offline outside the network.

### 2.1.4. Generating Your ICAM Symbiont (Load Module)

You generate your ICAM symbiont (load module) by a combination of network definition macros and system generation message control program (MCP) parameters. The macros define the network, that is, its lines, terminals, and other resources such as network buffers. A complete description of these macros is presented in the *ICAM Operations Guide* (UP-9745). Later in this manual is a discussion of the lines, terminals, and queues that will be helpful to you in understanding the operands associated with these macros.

The message control program parameters name the ICAM symbiont and identify the disk volume containing the symbiont and each single line communications adapter associated with your network. The MCP parameters are listed in the *ICAM Operations Guide* (UP-9745). A complete description of the message control program parameters is found in the *Installation Guide* (UP-8839).

### 2.1.5. Yielding Program Control

ICAM allows you to suspend execution of your program when there are no messages available and, when one becomes available, to automatically return control to your program to resume processing. You can use this feature by accessing a message from an input file, processing the message, and setting up another *get request* to receive control when the next message arrives. In writing a basic assembly language program, this feature is known as a deferred get. When writing a COBOL program, you use the RECEIVE statement, along with a NO DATA phrase, to do other processing while awaiting another message.

You may also use this feature when you wish your program to become dormant until an event occurs, such as another program sending you a datagram, or a dynamic session from a terminal or another program end user.

RPG II does not support this feature. If no message is available, your program must wait.

### 2.1.6. How to Activate a Communications Program and Pass It a Message

ICAM provides a GAWAKE feature that permits any program anywhere in the system to activate (awake) a communications program, optionally pass it a message called a datagram, and give control to the activated program at a predetermined entry point. This feature is available for global networks only, and the program that initiates the action need not be a communication program. This feature allows you to coordinate the activities of both a program with communications and another program without communications. Thus, it's possible to run the noncommunications program continuously and to activate the communications program only when needed - such as when output is ready for transmission or at certain times of day.

To use this feature, the receiving communications program registers itself with ICAM by issuing an input-type GAWAKE macro; the entry point where control is returned when the program is activated is specified; and the address of a receiving work area for datagrams is supplied if the program is to receive them.

The program initiating the activation does so by issuing an output-type GAWAKE macro. If a datagram is to be sent, the address of the work area containing the datagram (in the sending program) is also specified.

In addition to these requirements in your program, you must also specify the GAWAKE operand in the CCA macro. By doing this, you automatically include the GAWAKE software at system generation when you prepare your ICAM network. Without distributed communications architecture (DCA), this feature applies only to programs in the same computer node; that is, you cannot activate programs in another computer with this feature.

## 2.2. Communications User Programs

Communications programs are split into two groups: user-written application programs and programs supplied by Unisys. When you write your application program, you can use the basic assembly language, COBOL, or RPG II. Selecting the language depends on the ICAM interface you wish to use and your familiarity with a particular language. The basic assembly language gives you the most control over communications operations, whereas both COBOL and RPG II have some limitations.

In addition to the programs you write yourself, Unisys provides you with several utility programs that perform the following major functions:

- COBOL message control system (CMCS) - Lets you write your program in COBOL. CMCS is a module you create, using macros supplied by Unisys, and link to your program to enable operation with the ICAM standard interface.
- Remote batch processing (RBP) - Is the method of submitting batch jobs from a remote site.
- ICAM device emulation system (IDES) - Enables your system to emulate a 1004 card processor, a DCT 2000 data communications terminal, an IBM 2780 data communication terminal, or an IBM 3780 terminal.
- Nine thousand remote (NTR) - Allows your system to operate as a remote job entry batch terminal to a Series 1100 processor.
- Remote terminal processor (RTP) - Allows you to use System 80 as a remote job entry terminal to an IBM host processor.



## 2.2.1. User-Written Program Interfaces

### Standard Interface and Direct Data Interface Programs

Four major differences distinguish programs written for the standard interface from those written for the direct data interface:

1. The standard interface is easier to use because your program doesn't have to control the communications network.
2. With the standard interface, your program deals with the remote device handlers indirectly through network buffers and queues supplied by ICAM - not directly as with the direct data interface.
3. Standard interface programs can use both dedicated and global networks while direct data interface programs can only interface to dedicated networks.
4. You write standard interface programs in assembly language or COBOL; direct data interface programs must be written in assembly language or RPG II.

If you write a program that uses the direct data interface, it must initiate almost every action ICAM takes. It must direct ICAM to receive messages, send messages, and your program must also supply an error recovery routine. In addition, your program receives its messages directly from the ICAM remote device handlers, line buffer by line buffer. Hence, it must do its own message buffering.

If you write a program that uses the standard interface, ICAM automatically receives and queues all messages as terminals send them; it also delivers them to your program one at a time when your program requests them. It also queues outgoing messages given to it by your program. These are sent when the destination terminal is able to receive them (i.e., not down or busy). The standard interface also provides built-in error recovery routines.

When you are deciding whether to use the standard or the direct data interface, consider these points:

- It is easier to write a program for the standard interface because there's less you have to do. With little training, any programmer familiar with basic assembly language (BAL) noncommunications programs can write a standard interface program. To write a direct data interface program, however, a programmer must be well versed in communications and BAL.
- An ICAM module that supports a standard interface program requires considerably more main storage than one defined to support a direct data interface program. Figure 2-3 shows the logical components supplied by ICAM with a standard interface, and Figure 2-4 shows a direct data interface. The standard interface has more routines, tables, and storage areas than the direct data interface.

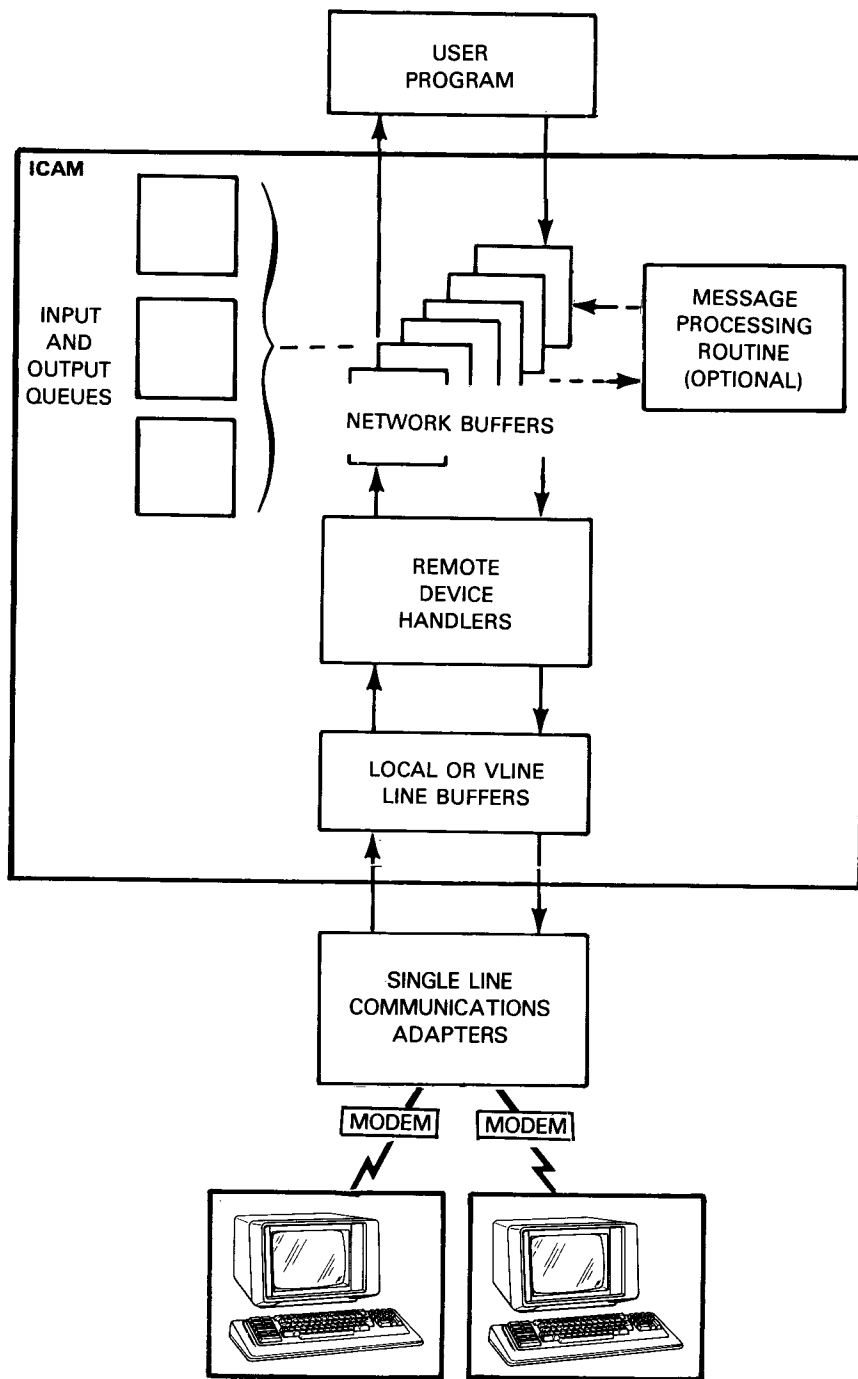


Figure 2-3. Standard Interface Communications System

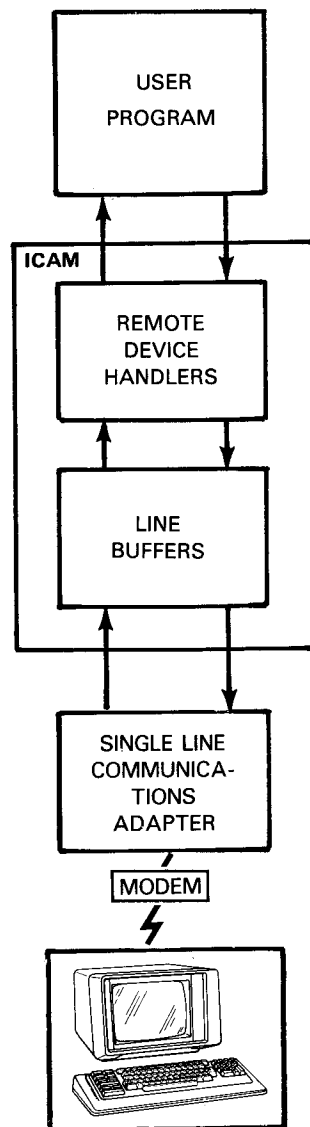


Figure 2-4. Direct Data Interface Communications Systems

- If you wish to use global networks, you must use the standard interface.
- If you want to write your programs in COBOL, you must use the standard interface. If you want to write your programs in RPG II, you must use the direct data interface. If you want to write your programs in basic assembly language, use either interface.
- If you are writing your own remote device handler, it is simpler to use the direct data interface because you can put many of the remote device handler functions in your program, making the remote device handler easier to interface with ICAM.

- The direct data interface is better for batch applications. It returns terminal status data to your program that may be used in error recovery procedures. It is also faster than the standard interface, allowing your program to receive messages as fast as the terminal sends them.

### Writing Your Program in Assembly Language

You can use assembly language to write a program that uses the standard interface or the direct data interface. Both of these interfaces involve the use of the following three basic functions:

- Request/ release ICAM facilities
- Send or receive messages
- Update certain ICAM tables

Before your program can do anything with ICAM, it must be linked to a network. How it does this depends on whether you are using a dedicated or global network. If your program uses the standard interface, you can program it to link to a dedicated or a global network. If it uses the direct data interface, it can only link to a dedicated network.

A dedicated network is linked to your program when your program issues a network request - a NETREQ macro. If the request is valid and the network is not already assigned, ICAM assigns the network and all lines and terminals defined in it to your program. Your program retains control of the network until it releases it. The interface is automatically initialized, and all lines are activated and ready to use unless you requested them not to be via the network request. In this case, your program must issue a line request to activate each line.

At any time, your program can activate or deactivate lines or release the network. If you deactivate a line, no traffic can flow on it. Output messages from a standard interface program are held on queue and terminals attempting to send input are ignored.

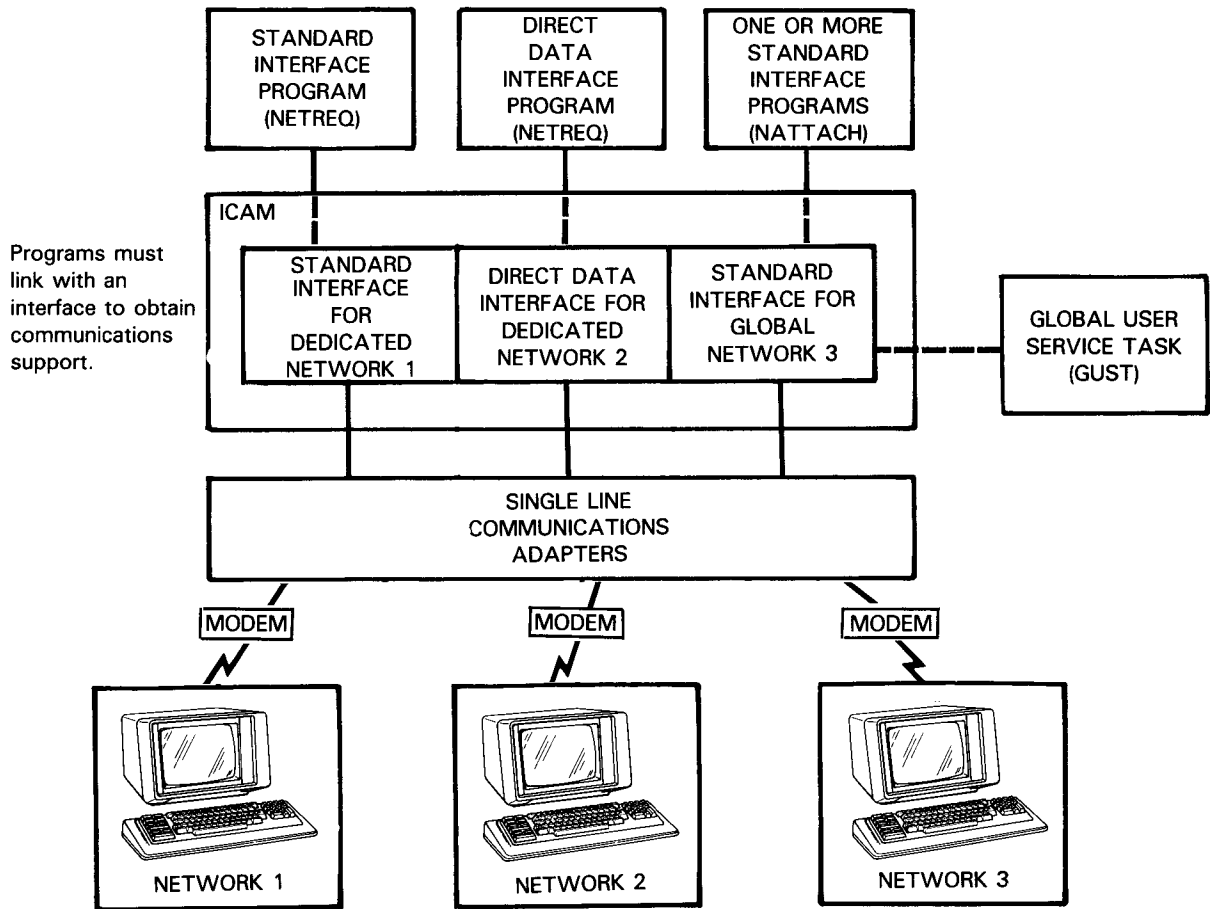
Your program can release a dedicated network at any time, thereby making it available to another program. If it is the last user of the ICAM symbiont, the symbiont will terminate. Networks are often activated and released frequently - this releases substantial amounts of main storage for other uses, and is often used in systems where communications activity is carried on only at certain times of the day. If you don't want the ICAM symbiont to terminate automatically, have the operator specify the KEEP operand when he loads the ICAM symbiont.

A global network is owned by a special program called the global user service task (GUST). The system operator loads and executes the GUST program, which loads the ICAM global network. Your program gains access to the global network by issuing an attachment request, and NATTACH macro, which is a variation of the network request you use to connect to a dedicated network.

However, because all lines and terminals belong to GUST, your program cannot activate/deactivate lines or terminals directly - it must send a message to the operator or you must directly request that the operator activate/deactivate certain lines.

The advantages of a global network are many, including the ability to have communications between your programs and the ability for all your programs to communicate with the same terminal (or other end users) concurrently. In addition, many facilities are not available with dedicated networks, like public data network support and distributed communications architecture.

The following figure shows three programs linked to an ICAM symbiont that includes two dedicated networks and a global network. One program is linked to network 1, a dedicated network using the standard interface. The second program is linked to network 2, a dedicated network using the direct data interface. The third links to the global network. Many programs can link to the global network at the same time.



After your program successfully requests a network, it can examine and, in some cases, change certain tables internal to ICAM. Both standard and direct data interface programs can do two functions:

1. Change the phone number for a line

This gives you the option of defining a logical line and terminal that actually support a number of terminals with different phone numbers. By defining one line and terminal in the network definition instead of several, you reduce the size of the ICAM module by hundreds or thousands of bytes. You could use this feature if you had a number of identical terminals that periodically need to communicate with the host processor. Your program would change the phone number for the logical line each time it wanted to send data to or receive data from a different terminal.

2. Have ICAM give the program information on the lines and terminals in the network

A program written for use with several networks would use this to get the names and characteristics of the lines and terminals in a specific network.

In addition to these two functions, a standard interface program can:

- Find out how many messages are in the queues of a process file, locap file, input terminal queue, or output queue
- Clear the messages from an output queue, process file, or locap file. The messages are cancelled without delivering them to their destination.
- Put a hold on an output queue, process file, or remote locap file, preventing messages from being delivered to their destinations. This function is useful if your program is building up a file for transmission only when the file is complete. The program then releases the messages in the output queue, process file, or locap file for delivery.
- Transfer messages from one output queue, process file, or locap file to another output queue, process file, or locap file

### **Sending and Receiving Messages with the Standard Interface**

The process for sending and receiving messages in the standard interface is purposely similar to the process of reading and writing messages from peripheral devices. Your program must create a file with a DTFCP macro and then issue a GETTCP or PUTTCP macro to receive or send a message. Compare the macros needed to read a card image with data management and to read a message with ICAM:

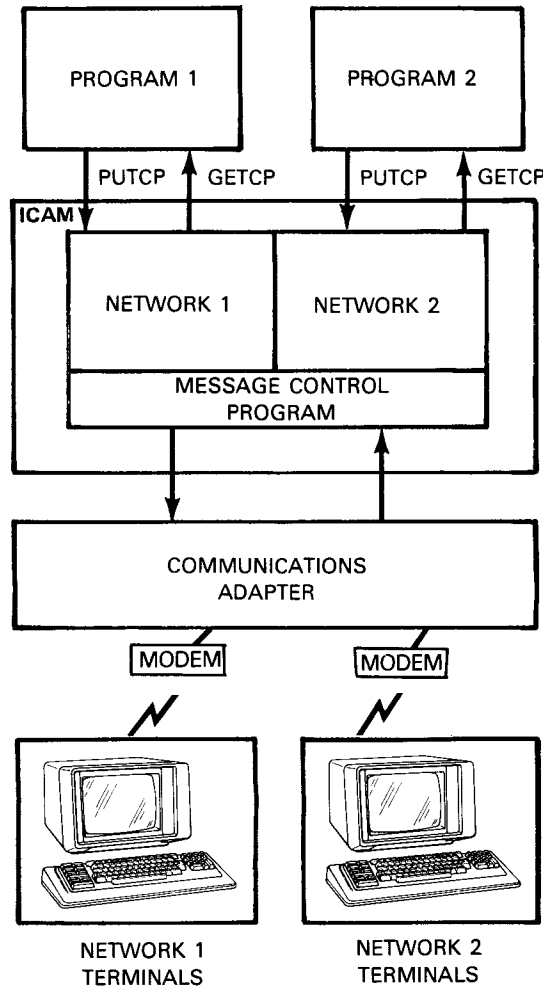
CONSOLIDATED DATA MANAGEMENT		ICAM DTFCP		
(CDM) CARD FILE		MESSAGE FILE		
CARD	CDIB	TRM1	DTFCP	TYPE=GT, LEVEL=LOW, ERRET=GETERR, NOMAV=NOMESS
	.			.
	.			.
	.			.
	.			.
	.			.
	DMINP	CARD	GETCP	TRM1,INAREA

Notice, only one macro is specified to create the CDM card file. Consolidated data management supplies the appropriate default values for the card file. For more information on System 80 data management, see the *Consolidated Data Management Macroinstructions Programming Guide* (UP-9979).

In the standard interface, reading and writing a message has nothing to do with transmitting it between the terminal and the host processor. Reading a message takes the first message in a terminal input message queue, process file, or locap file and places it in a work area in your program. This has nothing to do with receiving a message from a terminal and placing it on a queue. Writing a message places the message at the bottom of an output queue, process file, or locap file. Again, this has nothing to do with delivering the message to its final destination.

Just as the consolidated data management macros create file tables containing information necessary to GET and PUT messages, the standard interface DTFCP macro creates file tables. They are the major means of passing information between your program and ICAM. For example, the tables give the source or destination of a message, error and status codes, time and date stamps, error routine addresses, and special function codes.

The following illustrates two programs sending and receiving messages to their respective dedicated networks by using PUTCP and GETTCP macros. A PUTCP transfers a message from your program to ICAM (output); a GETTCP transfers a message from ICAM to your program (input).





## **Sending and Receiving Messages with the Direct Data Interface**

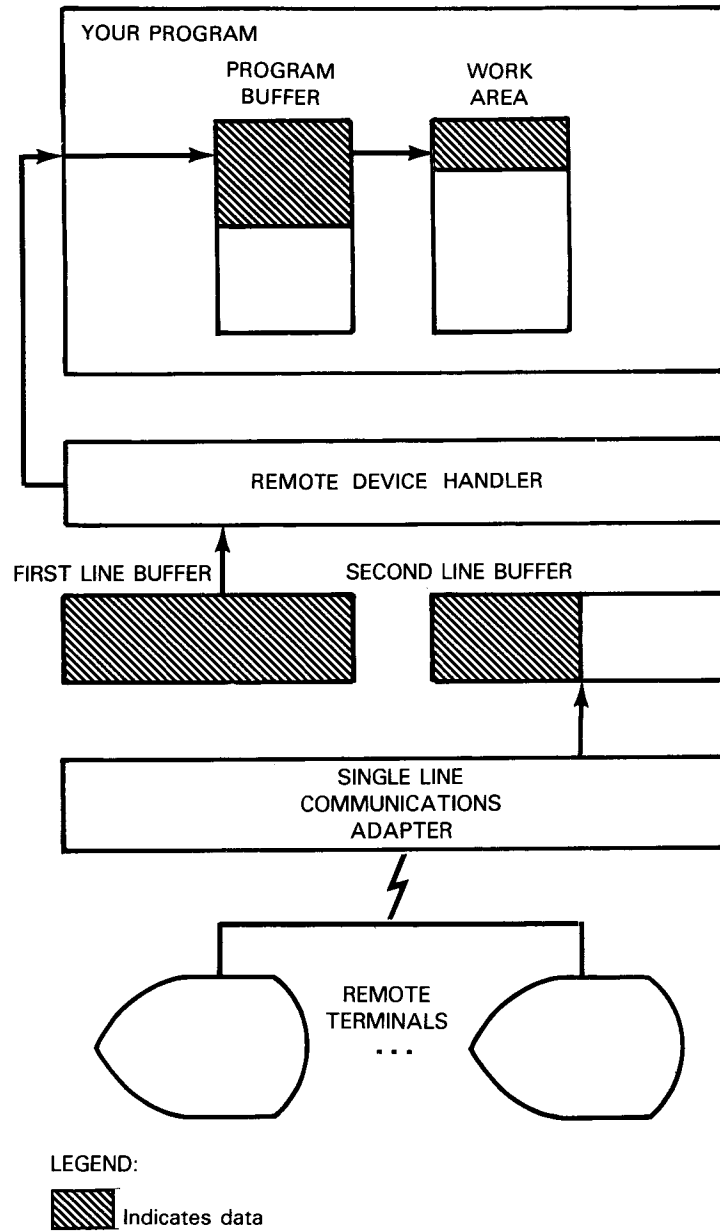
Receiving and sending messages in the direct data interface is totally unlike receiving and sending messages in the standard interface. In the standard interface, your program is isolated from the physical aspects of input and output by the network buffering function. In the direct data interface, your program directs the operation of the remote device handlers and performs the network buffering function itself.

The key to the use of the direct data interface is the message control table (MCT). ICAM uses them to direct the operation of the remote device handlers, and message control tables are used in all interfaces except the communications physical interface. Standard interface programs are not directly involved with their use because of the network buffering function performed by ICAM. Unlike a standard interface program, which uses the DTFCP file tables to pass information to and from ICAM, a direct data interface program controls many of the functions of ICAM by using the message control table.

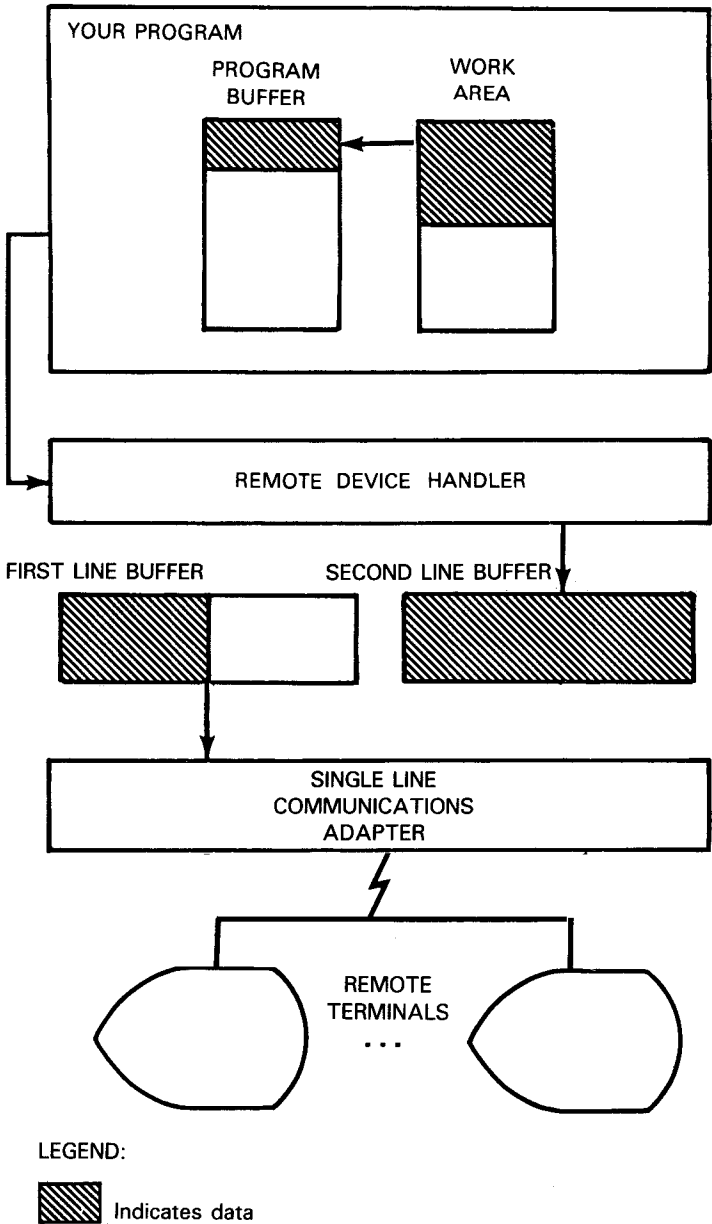
The remote device handlers cannot receive or transmit messages without a message control table. When your program creates a message control table, it defines the line, the terminal, and the auxiliary device the message is to come from or go to; it defines the function (send or receive in batch or interactive mode) to be performed, and it defines the program buffer the message comes from or goes to. After the remote device handler performs the function defined, it updates the message control table with error and status code and returns control to your program. If an error occurred, your program must tell the remote device handlers, through the message control table, how to recover.

Section 4.1 describes how the remote device handlers place messages in the network buffers as the line buffers fill. Because the direct data interface doesn't have network buffers, your program must provide program buffers with which the remote device handlers can work. Distinct from the program buffers, you may want to provide separate work areas to place the message in during processing.

The following illustrates input data flow:



For output, the process reverses:



### COBOL Programs

The COBOL message control system (CMCS) is a module that you generate and link to your COBOL program so you can communicate with remote terminals through ICAM. The COBOL message control system module acts like a bridge between your COBOL program and ICAM. It passes input messages from ICAM to your COBOL program, and it passes output messages from your COBOL program to ICAM. Figure 2-5 is a simplified diagram of a typical COBOL-CMCS/ICAM environment. It shows how your COBOL program requests functions, like activating a communications line, sending an output message, requesting an input message, or deactivating a communications line. You request these functions in COBOL terminology and the CMCS module converts your requests into the appropriate ICAM commands. ICAM takes care of queueing your output to the proper output queues and handles all input traffic, placing each input message on the correct input queue according to what you specified in your network definition. When your COBOL program requests a message from an input queue, ICAM accesses the appropriate queue (or hierarchical queue structure) and delivers the message to CMCS - and thereby to your COBOL program.

The information for writing COBOL communications programs is found in the *1974 American Standard COBOL Programming Reference Manual* (UP-8613), the *ICAM Utilities Programming Guide* (UP-9748), and the *ICAM Operations Guide* (UP-9745).

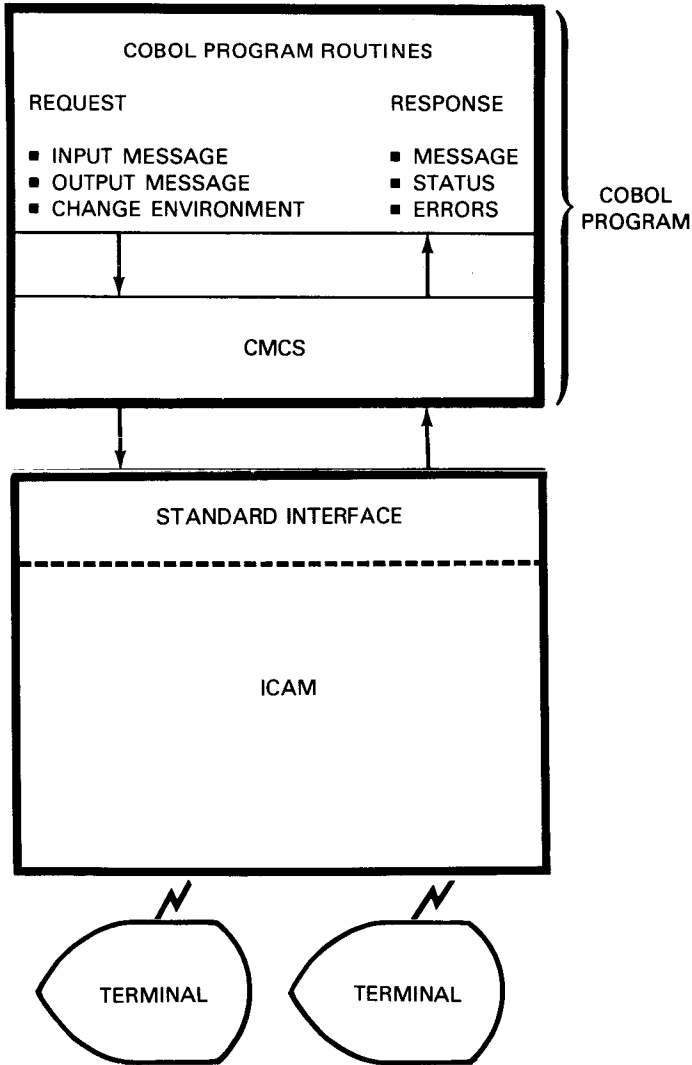


Figure 2-5. Typical COBOL-CMCS/ICAM Environment

**RPG II Programs**

RPG II has two communications-related commands, one to receive a message and one to send a message. All other functions needed by a program using the direct data interface are generated by the RPG II compiler when you compile your source code. RPG II cannot be used for sophisticated communications applications such as message switching. It is provided so your program can use terminals as input and output devices the same way it uses peripheral devices such as card readers and printers.

RPG II distinguishes between batch and interactive terminals. For batch terminals, you create input and output files the same way you would for card readers and printers. For interactive terminals, you create combined input/output files as you would for combined card reader/punches. For additional details, refer to the *RPG II Programming Guide* (UP-8067).

RPG II offers four processing modes for batch terminals:

1. *Receive only* - In this mode, a data file is received from a remote terminal. The file must be specified as an input primary file, input secondary file, or input demand file on the file description specifications, and it must be specified as a receive file on the telecommunications specifications. When this mode is used with a batch terminal, the input records are usually 80-column cards.
2. *Transmit only* - In this mode, a data file is transmitted to a remote terminal. The file must be specified as an output file on the file description specifications, and it must be specified as a transmit file on the telecommunications specifications. When this mode is used with a batch terminal, the output records are usually 80-column cards or lines of print.
3. *Transmit a file, then receive another file* - In this mode, two data files are specified for a remote terminal. The file to be transmitted must be specified as an output file on the file description specifications and as a transmit file on the telecommunications specifications. The file to be received must be specified as an input file on the file description specifications and as a receive file on the telecommunications specifications. When you use this mode, your program first processes the transmit file and then, when all data is transmitted to the remote terminal, it processes the receive file.
4. *Receive a file, then transmit a file* - In this mode, two data files are specified for a remote terminal. The file to be received must be specified as an input file on the file description specifications and as a receive file on the telecommunications specifications. The file to be transmitted must be specified as an output file on the file description specifications and as a transmit file on the telecommunications specifications. When you use this mode, your program first processes the receive file and then, when all data is received from the remote terminal, it processes the transmit file.

Note that, in this mode and the transmit-a-file, then-receive-a-file mode, it is your responsibility to design your program so that a complete file is processed for a particular remote terminal before another file is processed. This can be done by processing one file during the normal RPG II processing and the other file when the last record indicator (LR) is set on. Another method would be to use demand input and/or exception on the calculation specifications so that all the records of one file are processed first, after which all the records of the second file are processed.

Note also that a particular remote terminal is not limited to one input (receive) file and one output (transmit) file when you use this mode or the transmit-a-file, then-receive-a-file mode. You can process more than two files on a particular remote terminal if your program is arranged so that the processing does not overlap.

RPG II offers two processing modes for interactive terminals:

1. Transmit with reception of conversational reply

In this mode, a record is transmitted to a remote terminal and a reply is received back from the remote terminal. The file must be specified as a combined file on the file description specifications and as a transmit file on the telecommunications specification. The file may be specified as the primary file.

2. Receive with transmittal of conversational reply

In this mode, a record is received from a remote terminal and a reply is transmitted back to the remote terminal. The file must be specified as a combined file on the file description specifications and as a receive file on the telecommunications specifications. The file may be specified as the primary file. This mode is used only with a BSC remote terminal (computer to computer).

## 2.2.2. Programs and Products Supplied by Unisys

The following is a list of programs supplied by Unisys that work with ICAM. Those that work with ICAM in a specialized manner are called utilities. All of them except IMS and NTR are described in the *ICAM Utilities Programming Guide* (UP-9748). See the *IMS Technical Overview* (UP-9205) for details on IMS, and the *NTR Utility Programming Guide* (UP-9502) for details on NTR.

- Information management system (IMS) - Provides a transaction processing system (described later in this section).
- COBOL message control system utility
- Remote batch processing utility - Allows you to submit OS/3 jobs from a remote batch terminal (described later in this section).
- ICAM device emulation system utility - Allows your host processor to appear as a batch terminal to another host processor (described later in this section).
- Nine thousand remote (NTR) system utility - Allows you to use your processor as a terminal to submit jobs and data files to a remote Unisys Series 1100 processor (described later in this section).

- **Journal utility** - Produces printed reports that list the text of incoming and outgoing messages, the number of messages sent or received, and the extent to which your network buffer and activity request packet pools were used. The journal utility also has a feature that allows you to recover input or output messages queued on disk when ICAM fails due to an unrecoverable disk queuing problem.
- **Single line communications adapter utility** - Dumps and prints the contents of the random access memory in a single line communications adapter. The operator idles the line serviced by the SLCA and issues a SLCADUMP command to initiate dumping. The command includes the channel and line identifier and optionally an SLCA type. Dump listing is automatic.
- **ICAM trace facility (ITF)** - Helps locate the cause of ICAM operational problems. It is not intended to aid users in their own troubleshooting, but as a method of accumulating records of ICAM functions for later analysis by Unisys personnel. ITF is loaded only when needed and has no system generation requirements other than the system must include dynamic buffering.
- **ICAM edit dump** - A symbiont that dumps selected groups of ICAM tables for diagnostic purposes.

Other jobs, including other communications programs, can execute concurrently with any of these utilities.

In addition to the utilities, Unisys provides the following interconnect products:

- **OS/3 to UNIX operating system connectivity** - These features let you connect OS/3 to a UNIX operating system. You can initiate both interactive and batch procedures on OS/3 from the UNIX operating system, execute UNIX applications and shell commands from OS/3, and transfer files between OS/3 and the UNIX operating system from either operating system. The UNIX system access module (UNXSAM), which allows you to execute UNIX applications and shell commands from OS/3, is described in the *ICAM Utilities Programming Guide (UP-9748)*.
- **OS/3 to MAPPER<sup>®</sup> 5 Connectivity** - This feature, which is described later in this section, lets you connect OS/3 to a MAPPER 5 system. You can access its data base or transfer files from a terminal or local workstation connected to OS/3.

### Information Management System (IMS)

The information management system (IMS) is a transaction-oriented file processing system that works with ICAM in an interactive environment. You enter an inquiry at a display or hard copy terminal, and IMS accesses your data files and responds with one or more output messages.

---

MAPPER is a registered trademark and service mark of Unisys Corporation.



IMS is suited to any application requiring instant access to information. Within moments of an inquiry, the information you request is displayed. That information is always current because IMS updates your files at your request from the terminal.

Here's a typical IMS inquiry/response sequence. The shaded messages are the ones your terminal operator enters; the remaining messages are the IMS responses.

```
CREDIT
PLEASE GIVE CUSTOMER NAME AND ACCOUNT NUMBER;
PAULVILLE CANDY 6743
CREDIT RECORD FOR PAULVILLE CANDY IS;
CREDIT LIMIT; 10000    CREDIT BALANCE;    534
A/R TERMS; 30 NET 45  A/R DAYS 30  A/R DISCOUNT; .080
DO YOU WISH TO CHANGE ANY OF THIS INFORMATION?
YES
PLEASE ENTER FIELD NAME FOLLOWED BY NEW INFORMATION;
CREDIT LIMIT 12500
NEW CREDIT LIMIT IS 12500
```

You create a tailored IMS system in a configuration process you use to describe your communications network, your files, and your applications, and to select optional features. The result of the process is an IMS load module that handles:

- Interfacing with ICAM
- Verifying, editing, and scheduling of communications messages
- Accessing of your data files

Routines called *action programs*, which you write or IMS supplies, process your input messages. When IMS receives a message, it schedules the appropriate action programs. They examine the contents of the message, request data from your files, prepare responses, and, if necessary, schedule additional action programs.

Figure 2-6 shows the ICAM communication path for IMS.

You can write action programs in COBOL, RPG II, or basic assembly language. The terminal display shown earlier in this section is an example of the kind of file processing and message formatting you can do when you write your own action programs.

The set of action programs supplied by IMS is called the uniform inquiry update element (UNIQUE). UNIQUE lets you retrieve, add, delete, and change records in your files by entering simple commands from terminals.

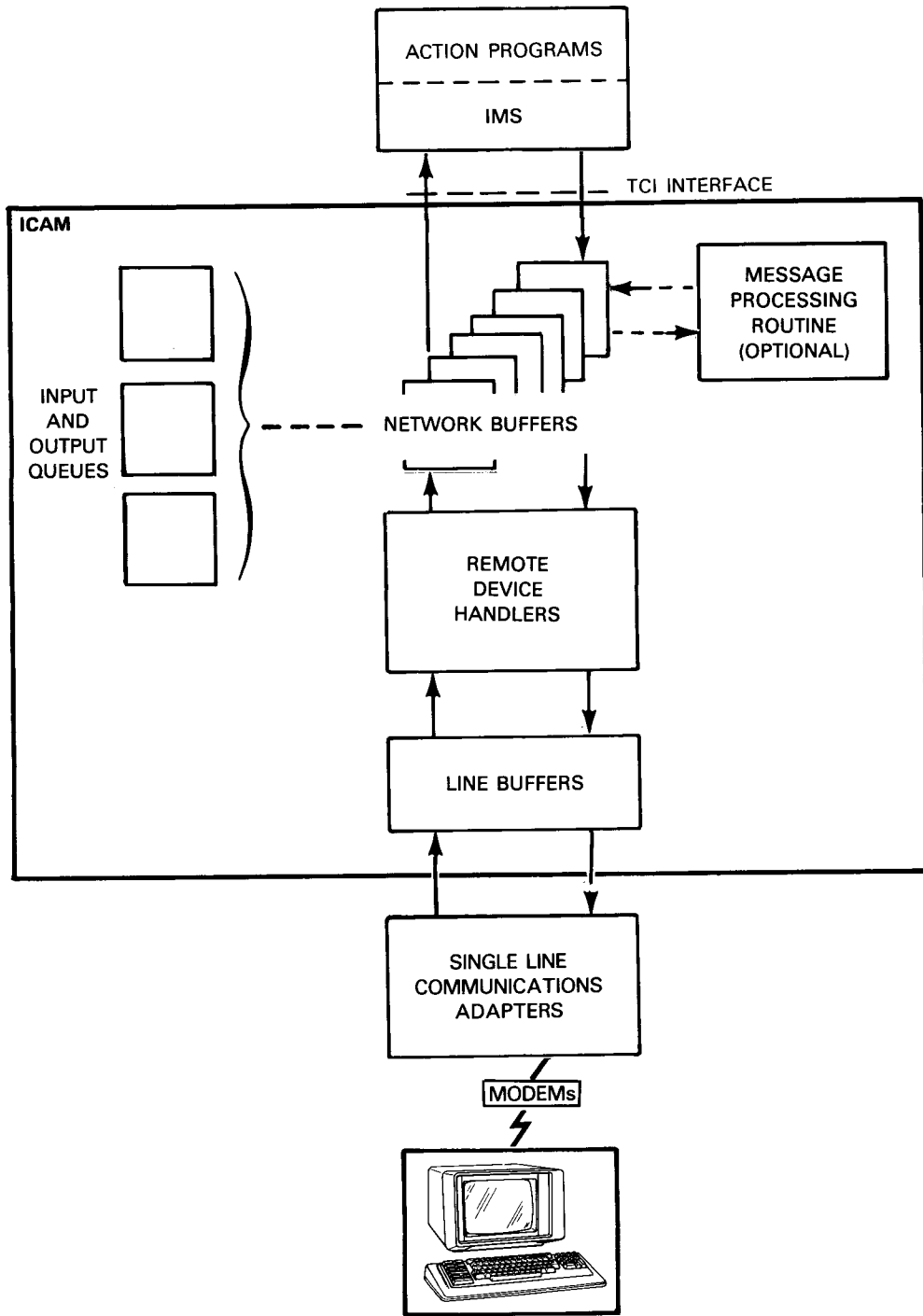


Figure 2-6. IMS in an ICAM Environment

Here's a typical UNIQUE dialog corresponding to the earlier inquiry/response example:

```

OPEN CREDIT
CHANGE PAULVILLE CANDY
NAME          ACCTNO CR LIM  CR BAL  A/R TERMS  A/R DAYS  DSCNT
PAULVILLE CANDY  6743  10000  534  30 NET 45  30      .080
                ***** ***** ***** ***      ***

```

To change the credit limit, you overwrite the asterisks in the CR LIM column; the display now looks like this:

```

OPEN CREDIT
CHANGE PAULVILLE CANDY
NAME          ACCTNO CR LIM  CR BAL  A/R TERMS  A/R DAYS  DSCNT
PAULVILLE CANDY  6743  10000  534  30 NET 45  30      .080
                12500 ***** ***** ***      ***

CHANGE COMPLETE

```

IMS supports most of the terminals supported by ICAM. Because the list of new terminals supported with IMS is constantly growing, you should refer to the *IMS System Support Functions User Guide* (UP-11907) for terminals currently supported.

You can also have IMS operate in a global network, which permits:

- Any mix of IMS and standard interface users
- IMS, standard interface, and interactive services to communicate with different terminals on the same communications line
- Both dedicated network and global network IMS users running concurrently
- IMS, standard interface, and interactive services users to establish sessions so that they all can use the same terminal right after each other (serially)

Details on generating IMS and on writing a network definition for an ICAM that supports IMS are in the *IMS System Support Functions User Guide* (UP-11907). Preparation and processing of IMS applications, including user action programs and UNIQUE, are described in the *IMS Action Programming in RPG II Programming Guide* (UP-9206), the *IMS COBOL/Assembler Action Programs Programming Guide* (UP-9207), and the *IMS Data Definition and UNIQUE User Guide* (UP-9209).

## Remote Batch Processing (RBP) Utility

The remote batch processor allows you to use a batch terminal at a remote site to submit jobs to the OS/3 supervisor and to receive output from those jobs. Users at the remote site use the batch terminal to send a job stream to the remote batch processor. The batch terminal can be another computer system such as a UNIX system. See the *OS/3 - UNIX Operating System Connectivity Operating Guide* (UP-14207).

Here is a sample job stream:

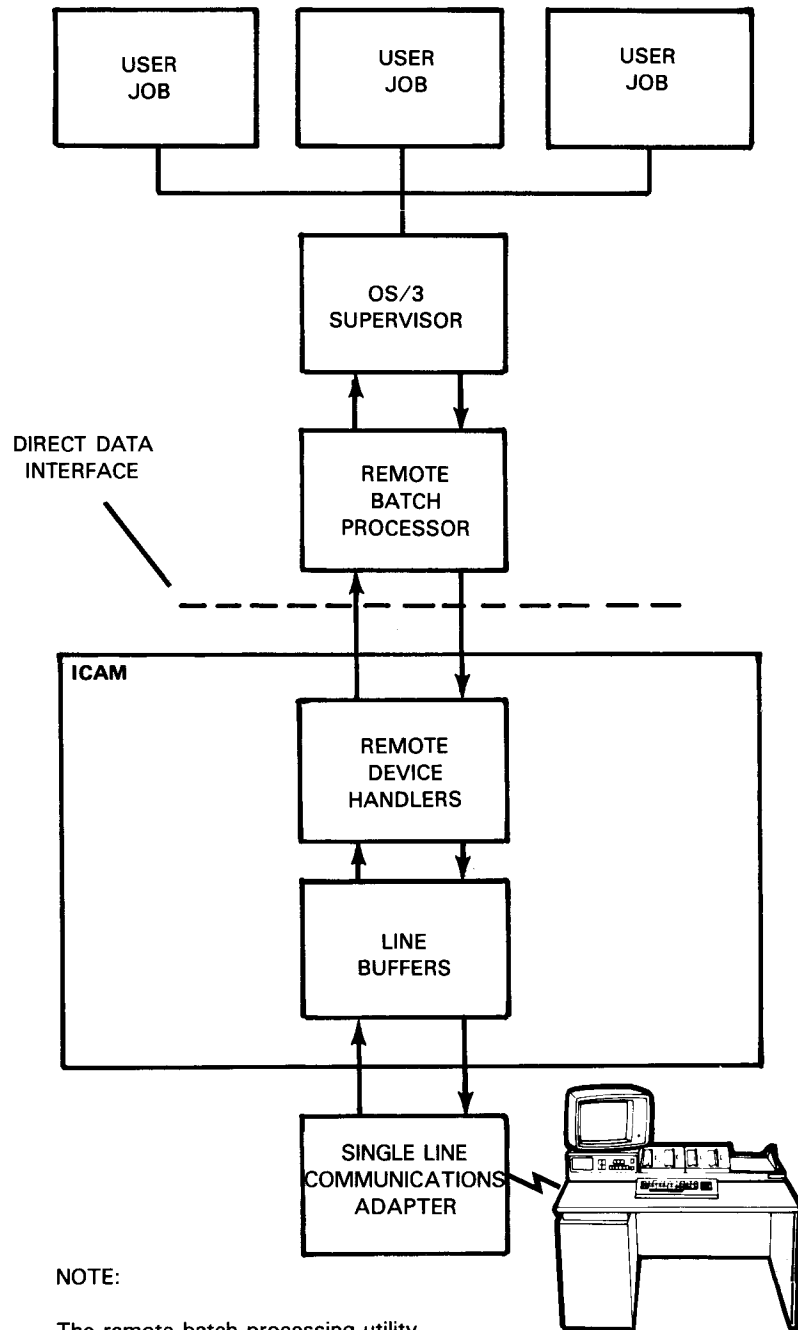
```
/RSTART   RBP1 _____ Logically attaches this terminal to the
/RLOGON   USERA,7476 _____ remote batch processing system.
// JOB A
// DVC 20
// DST USERA,USERB _____ Logs user A into the remote batch
// PRNTR                                     processing system.
// ASMLG _____ Lists users allowed to receive the output.
/$
PROGRAM   START 0
          .
          .
          .
/*
// FIN
/RLOGOFF _____ Logs user off.
```

In this example, the program to be run is embedded in the job stream. It could have been stored in a disk file.

Except for the few commands unique to the remote batch processor, there are only a few differences between submitting a job from a batch terminal via the remote batch processor and submitting a job from a card reader 10 feet from the host processor. The most important of these differences are:

- User identifications are defined at network definition. Anyone using the system must log on with a proper identification.
- When submitting a job, you must specify the user allowed to receive copies of the output. You can send copies of the output to as many as 23 destinations, including the central processor.
- You can check on the status of a job submitted through the remote batch processor from the remote terminal.
- You can have the output from a job go to another batch terminal or to a printer/punch attached to the host processor.

Figure 2-7 shows a remote batch processing system.



**Figure 2-7. Remote Batch Processing System**

The *ICAM Utilities Programming Guide* (UP-9748) has a complete description of the remote batch processing utility.

### ICAM Device Emulation System (IDES) Utility

IDES allows System 80 to operate as a batch terminal to another computer. Use it when you have one of these computers at a site where you need a computer acting as a batch terminal to transmit data files. Do not confuse IDES with distributed data processing. The processor acting as a batch terminal cannot process the messages, switch messages to other destinations, or interrogate the data base of the other computer.

The IDES system has four major parts:

1. A direct data interface

A direct data interface is used with a special remote device handler to emulate a 1004 card processing system, a DCT 2000 data communications terminal, an IBM 2780 data communications terminal, or an IBM 3780 terminal.

The remote device handler uses the data code and protocol of the terminal it's emulating. To the other computer, the computer using IDES is the terminal it emulates. A local line, and not a VLINE, connects the two computers.

2. An IDES driver program that runs as a normal OS/3 user job

The IDES driver program controls input and output to the peripheral devices. It does not process the messages sent between the peripheral devices and the other host computer.

3. Peripheral devices that may be card readers, card punches, and printers

These are the input and output devices for this processor when it acts as a batch terminal. As long as the IDES driver program is running, these peripherals are dedicated to IDES and cannot be used by other jobs.

4. Another host computer

This is any computer that has a remote device handler (or its equivalent) to work with a 1004 card processing system, a DCT 2000 data communications terminal, an IBM 2780 data communications terminal, or an IBM 3780 terminal.

Figure 2-8 shows an IDES.

The *ICAM Utilities Programming Guide* (UP-9748) has a complete description of IDES.

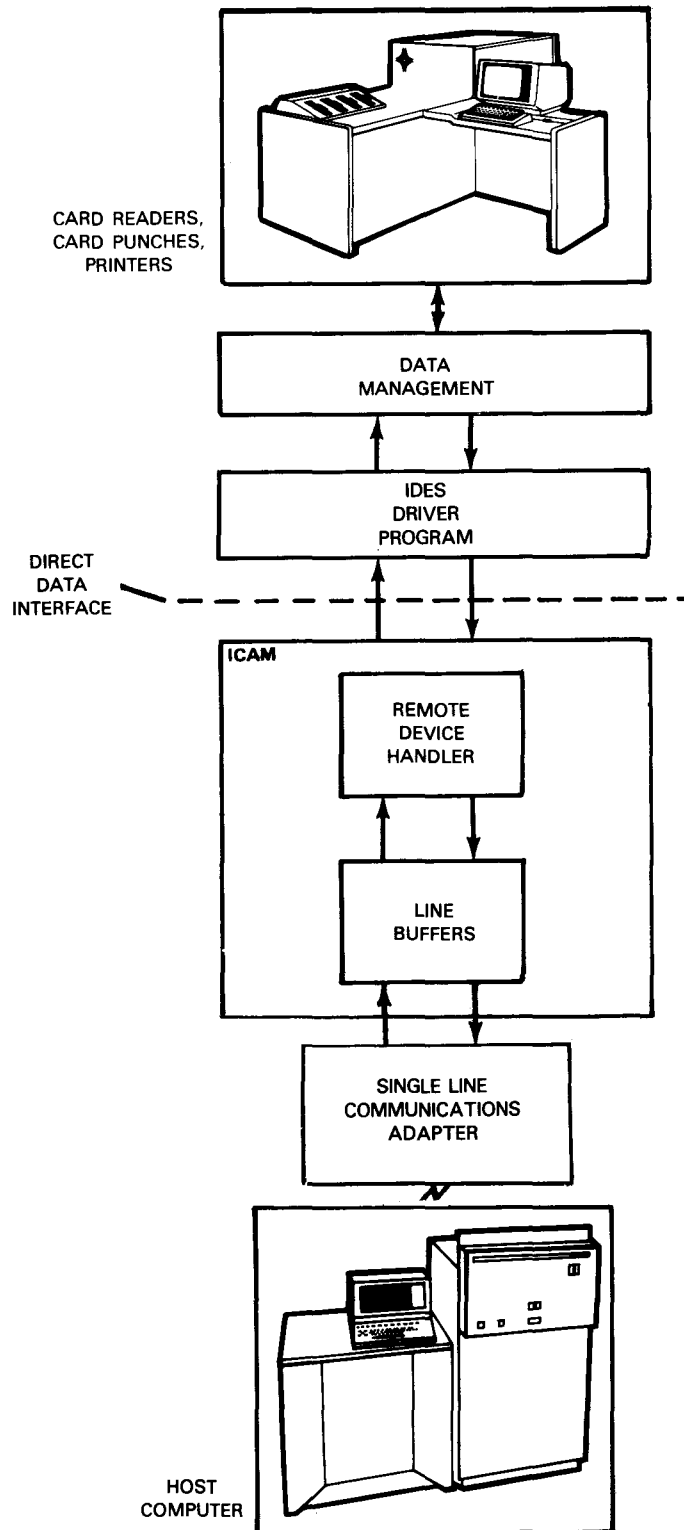


Figure 2-8. ICAM Device Emulation System

### NTR Utility

The NTR (nine thousand remote) utility, illustrated in Figure 2-9, allows a System 80 computer to act as a remote job entry terminal to a Unisys Series 1100 computer. You use this utility to submit jobs or complete data files to the Series 1100 computer and receive data files from it. You can also use the NTR utility to send messages between the operator console in the System 80 computer, and the operator console in the Series 1100 computer.

The NTR utility provides the capability to write user subtasks to process data files before they are sent to the Series 1100 computer or after they have been received from the Series 1100 computer.

The utility runs as a normal OS/3 job and provides an IDES driver program and task manager that enables your computer to act as a batch terminal to a Series 1100 host processor using the ICAM direct data interface. A 2-way simultaneous communications line (not a virtual communications line (VLINE)) is required to connect to the Series 1100 computer. Required single line communications adapters (SLCAs) are listed in Table 3-4.

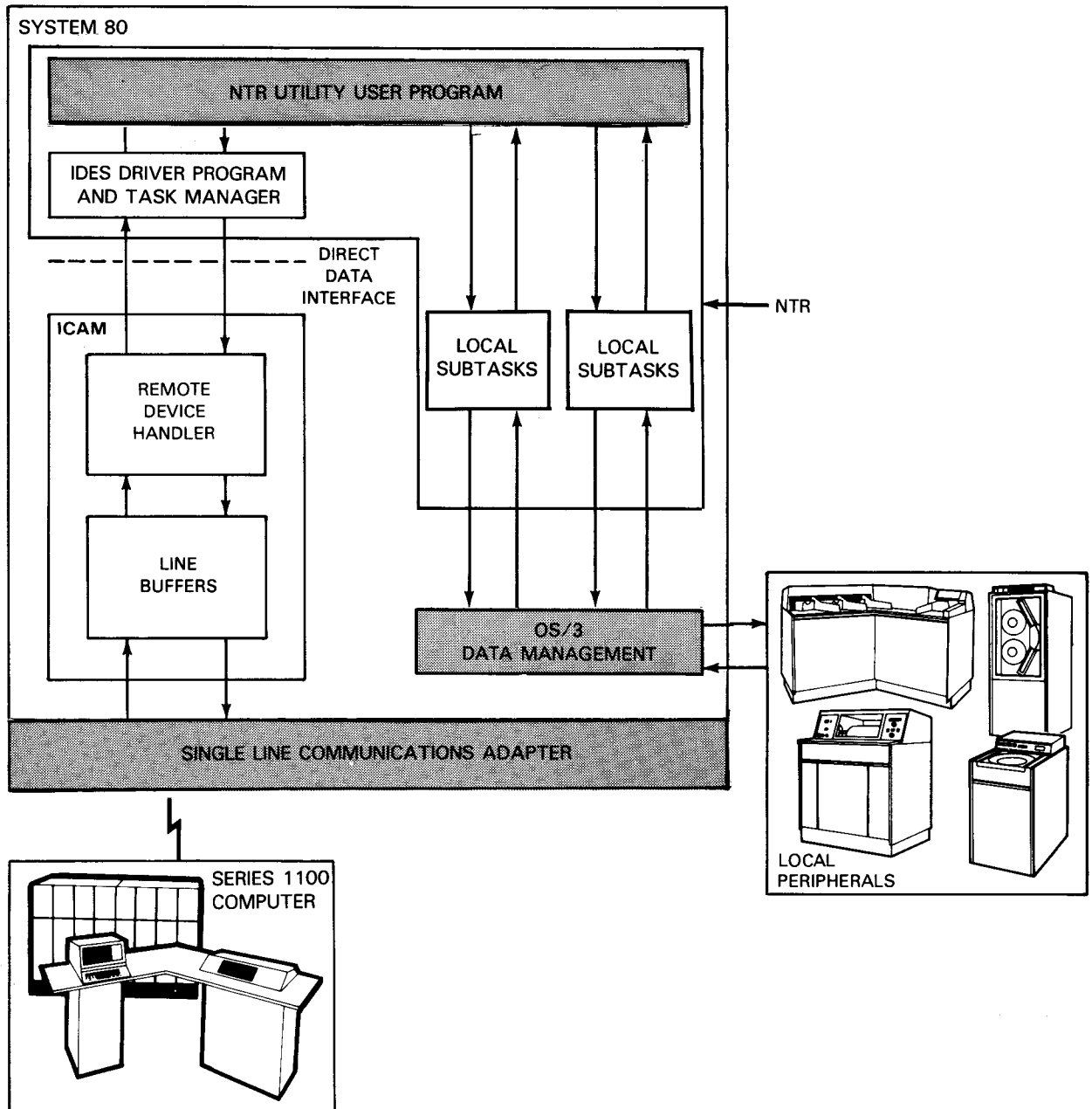
The NTR utility's IDES driver program and task manager provide local and user-written subtask capabilities. Local subtasks are subtasks written by Unisys that control input and output to local card readers, card punches, and the OS/3 system console. These local tasks perform no processing of data between the local peripheral devices and the Series 1100 host processor.

User subtasks are optional routines you write to process data files before or after transmission or to support any peripheral device not supported by a local subtask. For example, you might write a user subtask to control a disk drive, magnetic tape drive, or an optical document reader.

The peripheral devices supported by local and user-written subtasks are the input and output devices for this computer when it acts as a batch terminal to the Series 1100 computer. If you use spooling with the NTR utility, other jobs can also use these peripherals; otherwise, the peripherals defined are dedicated to the nine thousand remote utility.

The NTR utility is described in the *NTR Utility Programming Guide* (UP-9502).





NOTE:  
Shaded areas are not part of ICAM.

Figure 2-9. NTR Utility System

### IBM 3270 Emulator

The 3270 emulator provides a way to connect System 80 to an IBM host system. It allows System 80 workstation users to access application and IBM program products running on an IBM host. The 3270 emulator:

- Acts as an IBM 3271 control unit, providing communication between System 80 and the IBM host system
- Supports standard interface user programs and System 80 local workstations emulating IBM 3277 Model 2 display terminals

For details on the 3270 emulator, see the *ICAM Standard MCP Interface (STDMCP) Programming Guide* (UP-8550).

### OS/3 Remote Terminal Processor

The remote terminal processor (RTP) is a data communications program that permits your System 80 processor to function as a remote job entry terminal to one or more IBM host processors using one of the following software systems:

- Houston automatic spooling program (HASP)
- Job entry system 2 (JES2)
- Job entry system 3 (JES3)

RTP enables you to:

- Send jobs to an IBM host
- Transmit and receive files on tape
- Transmit files from diskette
- Send messages to the central site
- Receive printer or punch output and provide printer forms control
- Communicate with the IBM console from the OS/3 console

RTP is controlled through the OS/3 system console. Simple keyins direct the operation of RTP. For instructions for generating and using RTP, see the *ICAM Remote Terminal Processor (RTP) User Guide* (UP-8990).

## Journal Utility

Journaling is the process of having ICAM create records and write them to disk or tape. These records are the raw material from which the journal utility (a separate program) uses to produce printed reports about your network and, if necessary, initiate a restart process. Restart is the process of reconstruction of ICAM disk queues if hardware problems develop.

Journaling involves:

- Adding to your ICAM network definition the macros necessary to incorporate the journaling features
- Creating a message processing routine (MPPS) in your network definition to specify which journal records (messages) you want to record, the name of the file to receive the journal records, and point in time at which the records are to be made
- Executing the journal utility to produce printed reports, or effect a restart using restart records

The journal utility (JUST) produces printed reports that list the text of incoming and outgoing messages, the number of messages sent or received, and the extent to which your network buffers and activity request packet pools were used.

Should ICAM fail due to a disk problem, the journal utility allows you to recover messages queued on disk. When you request a restart, the journal utility rebuilds the disk message queues. Then, you perform a warm restart to resume message flow.

Note that the journal utility processes records gathered by ICAM, but the utility operates independently of ICAM, i.e., it does not require an ICAM environment.

To obtain printed reports, three journal utility statements are at your command. Each statement is associated with a report. For example, one statement (BSTAT) gives you a printed report on the extent of network buffer and activity request packet usage. These statements are placed in your job control stream, as is the statement for a restart. Details of these statements and examples of their use are found in the *ICAM Utilities Programming Guide* (UP-9748).

There are five different types of journal records you can create. The record types and the JRNINIT suboperands of the CCA macro that create them are:

1. Journal records - Each of these records contains the text of a message sent or received, the time and date the journal record was created, and other related information. Messages are written to a journal file at the request of a message processing routine (MPPS) you incorporate into the ICAM network.
2. ODNR records - ODNR (output delivery notice request) records are message dequeuing notices. ICAM writes these to a journal file when a message is dequeued for delivery to a terminal or other user program.

3. Line and terminal performance records - These records contain the total number of messages sent or received by each terminal on a line and the number of times each terminal was polled.
4. Buffer statistics records - These records contain the number of network buffers used and the frequency they were used. Also listed is the activity request packet size and frequency the packets were used.
5. Restart records - These records contain a copy of each message as it was queued to disk. If ICAM fails due to a disk message queueing problem, the journal utility can be used to reconstruct the ICAM disk message queue. You then perform a warm restart to resume message flow.

### Connecting OS/3 to a UNIX System

OS/3 to UNIX operating system connectivity enables you to connect a UNIX system to an OS/3 system over a communications line. This allows UNIX users access to both batch and interactive procedures from the UNIX system. You can also execute UNIX applications and shell commands from OS/3 and transfer files between the two operating systems.

Refer to the *OS/3 - UNIX Operating System Connectivity Operating Guide (UP-14207)* for details.

### Connecting OS/3 to a MAPPER 5 System

The OS/3 to MAPPER 5 connectivity program product allows you to sign on to OS/3 from an OS/3 terminal or workstation and access a MAPPER 5 system. Thus, when a communications line is installed between the MAPPER 5 system and the OS/3 system, you can use the data base and transfer files from the MAPPER 5 system from an OS/3 terminal or workstation just as though you were directly connected to the MAPPER 5 system.

When you configure your ICAM global network for MAPPER 5, you must specify the MAPPER 5 inverted remote device line handler; i.e., specify `DEVICE=(MAP5RDH)` on the ICAM LINE macro for the communications line that connects the MAPPER 5 system and OS/3.

Define the MAPPER 5 system as a U200 terminal; that is, specify:

```
LINE    DEVICE=(MAP5RDH)
TERM    FEATURES=(U200,screen-size)
```

You can configure your ICAM global network to support multiple lines with multiple terminals on the same line. This permits concurrent access of the MAPPER 5 system by OS/3 workstations or terminals.

Figure 2-10 shows how an OS/3 workstation or terminal operator signs on to an OS/3 ICAM global network that uses the MAPPER 5 remote device handler. The MAPPER 5 inverted remote device handler acts as a traffic manager and provides a passthrough function between the terminal and the MAPPER 5 system.

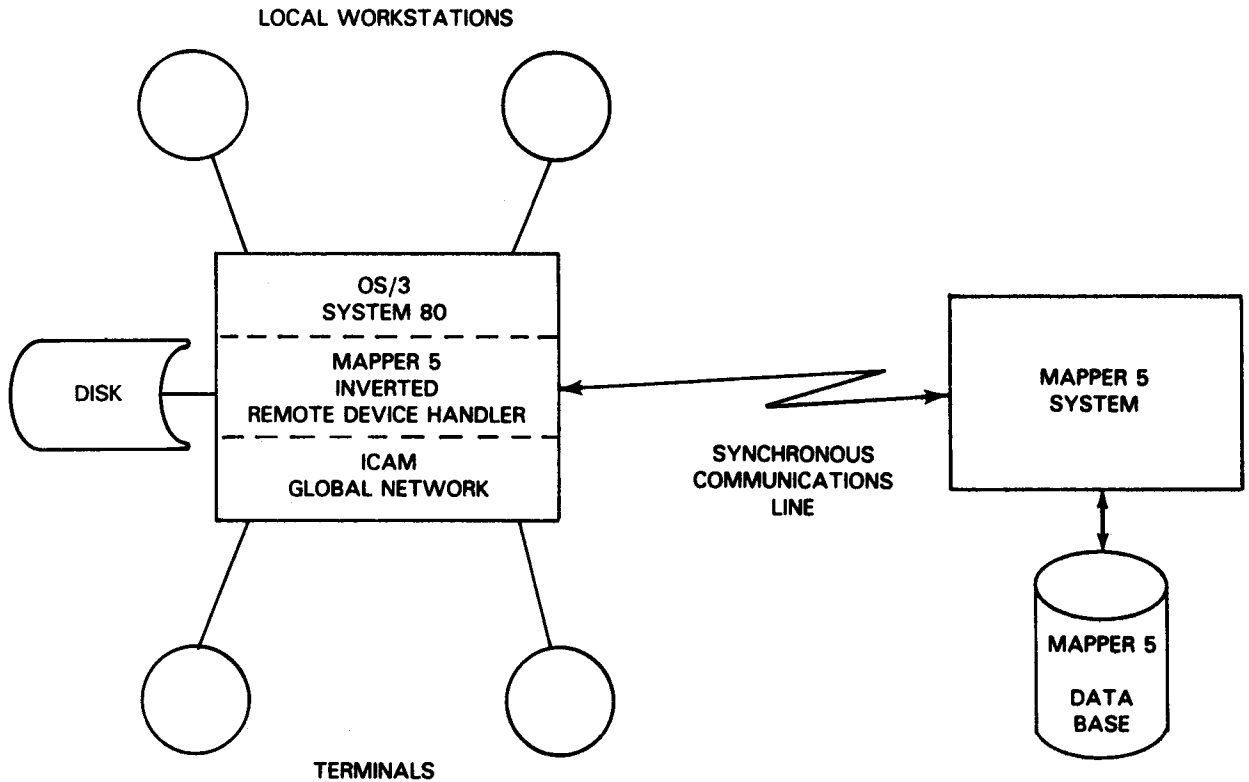


Figure 2-10. Connecting OS/3 to a MAPPER 5 System



# Section 3

## Communications Hardware

### 3.1. Basic System 80 Communications System

Up to this point, we have talked about communications software. Now, we will briefly discuss the communications hardware and its relationship with the software. Specifically, we will cover the terminals, lines, and the communications software.

Let's start with a discussion of the essential communications components of the smallest System 80. In this basic communications system (Figure 3-1) we have:

- Terminals - at least one on each communications line
- Four modems to support two communications lines
- Two single line communications adapters

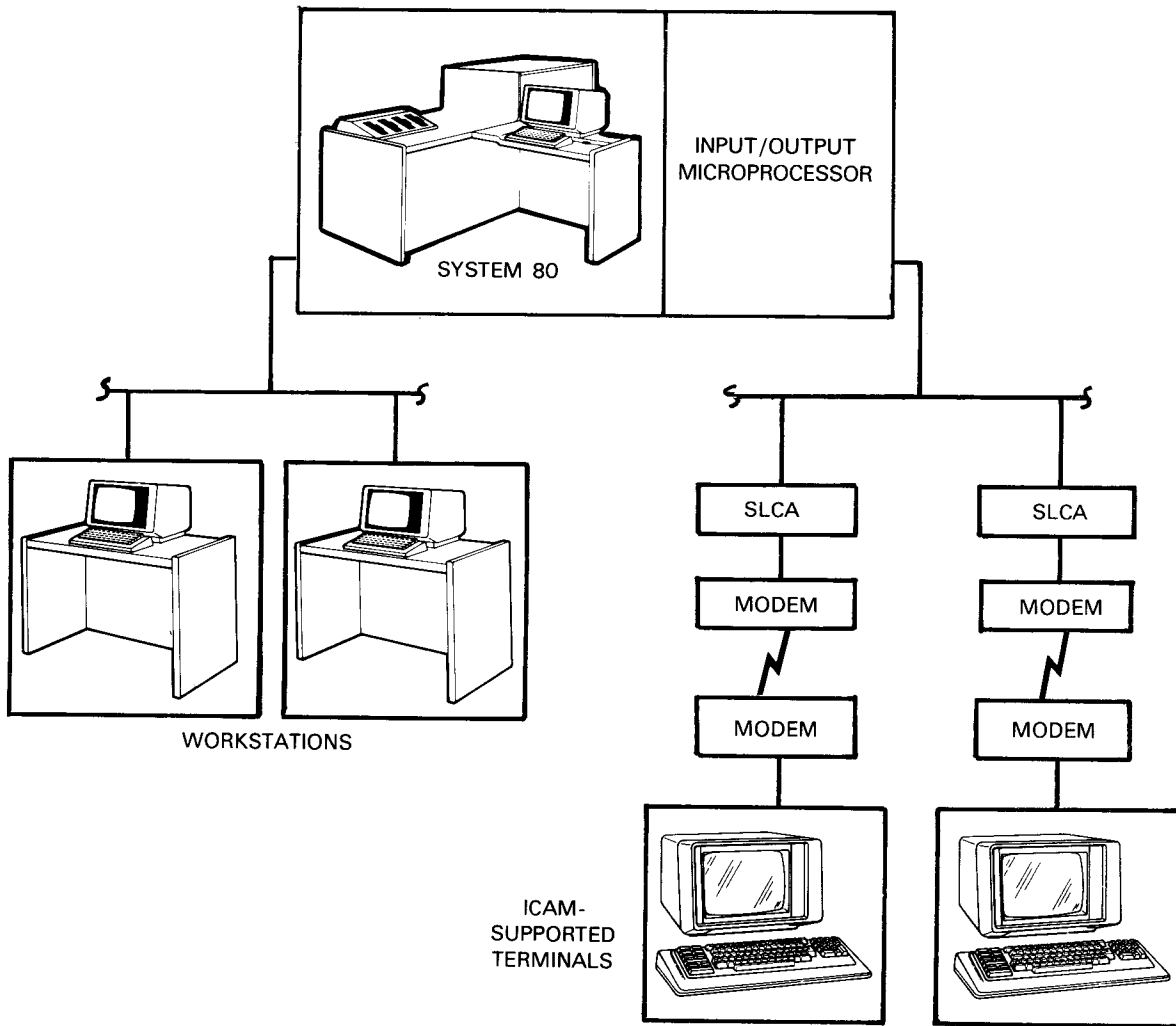
Terminals are devices that allow you to talk to your computer. They can be as large as an entire computer system or as small as a typewriter device.

Modems are devices used to interface a computer or terminal to a telephone line. Modems convert binary signals used by your computer and terminals to a form of linear signal that can be carried on a telephone line - a modulated carrier wave. (Modem is an abbreviation for modulator/demodulator.)

A single line communications adapter (SLCA) is required for each communications line connected to a System 80. During input the SLCA converts the serial data (bits) coming from a modem and assembles it into character form before transferring it to the computer. During output, the SLCA converts parallel data (bytes) to serial data and passes it to the modem for output to the communications line.

Within System 80, the input/output microprocessor coordinates all the input/output operations between each SLCA and the processor and handles the peripheral devices that make up your computer configuration, such as tape units, disk units, and card readers.

We have just described a minimum System 80 supporting only two communications lines. This system can be enhanced to support up to six additional communications lines, that is, up to eight lines. The largest System 80 provides up to 2 input/output microprocessor channels and each channel can support up to 14 SLCA's, that is, up to 28 communications lines.



NOTE:

SLCA means single line communications adapter.

Figure 3-1. Basic System 80 Communications System

### 3.2. The Terminals

A terminal is any input or output device that works with a computer through a communications line. The differences between a terminal and a local computer peripheral (like a printer or disk drive) aren't very great. A terminal has a few extra circuits, allowing it to work in a communications system, but the differences between it and a local peripheral end there. A peripheral is made into a terminal by adding communications interface circuits.



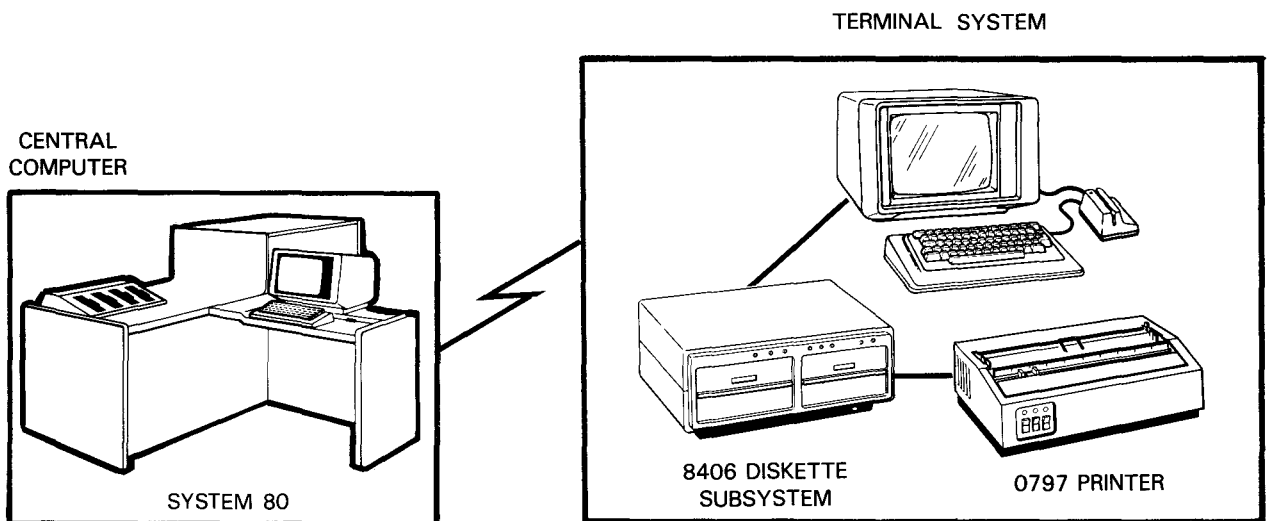
From the ICAM point of view, the communications interface is the important feature of a terminal because it's the part that works with ICAM. To your programs, however, the communications interface is invisible. To them, a terminal is an input/output device put to a particular use. We'll look at terminals from these two viewpoints: first, your program; then, ICAM.

Unfortunately, we can't use any one terminal as a typical example. Too many different terminals exist with different interface characteristics, different hardware, and different uses. Unless we describe the particular terminal you're using and the use you're putting it to, a description of a particular terminal isn't much help. What we give here describes idealized characteristics of a terminal. Before you can write a program or create ICAM, you must read the manual that comes with your terminal to see how your terminal uses these characteristics.

### 3.2.1. Hardware - What is a Terminal?

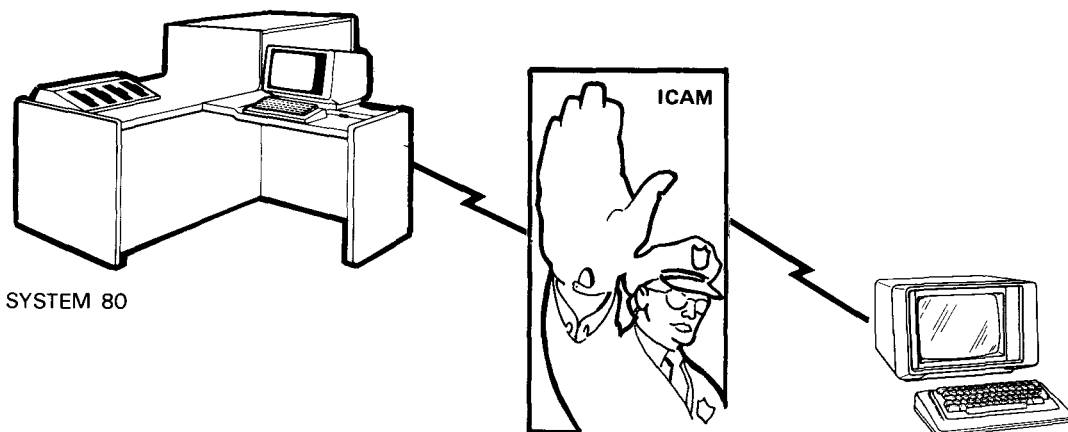
A terminal is any device capable of receiving or sending data over a communications line. Many terminals are the same as the peripheral devices that have been used for years: printers, disk drives, card readers, magnetic tape drives, diskette drives, keyboards, and cathode-ray tubes. Then there is the equipment normally used only in terminals, like paper tape punches and readers, cassette tape systems, graphics plotters, and cash registers. Whatever input and output equipment you have in your terminals, they must use digital signals.

Many remote locations combine two or more input and output devices to create a terminal system. The Universal Terminal System 40 Single Station pictured here combines a diskette subsystem, a printer, and a magnetic stripe reader to create a single terminal system.



A terminal system has another attribute - intelligence. Many terminals are now small computers, capable of running programs either in support of their communications role or to support offline batch jobs. For example, the UTS 40 single station system could run a program that edits messages or sorts data before they're sent to the central computer. Or it could run a payroll job that has nothing to do with communications. Larger computers, such as your system, can run communications jobs and noncommunications jobs concurrently.

The preceding paragraph brings up two points. First, your computer can also be considered a terminal. Like the UTS 40 single station, it has a processor, it has several input and output devices, and it can send and receive messages. ICAM sets your computer off from the rest of the terminals in your network. As the following subsections show, ICAM acts like a traffic cop. It controls when a terminal can send a message and when a message can go to a terminal.



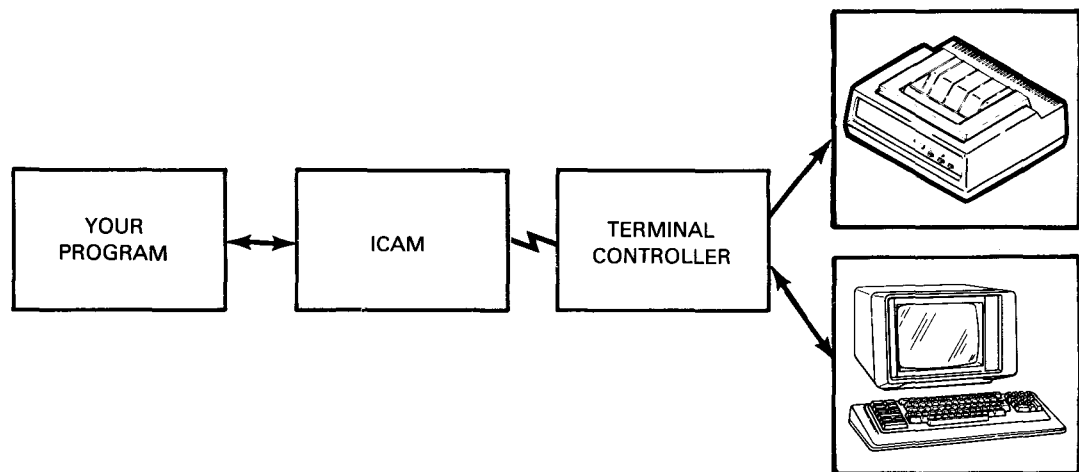
The second point is the idea of distributed processing - distributing the processing of your data. Instead of having one or two large central computers doing most of your processing, we can add a series of intelligent terminals. Not only do these terminals cost less, but they can process your work more efficiently since they can be set up at key locations to meet your local needs.

For example, let's say you're using your communications system for sales order entry. Your clerks fill out order forms at a UTS 40 single station, and the UTS 40 handles the formatting of the screen and the editing of the data. If you don't do it at the terminal, then you must do it at the host computer. If the host computer does the formatting and editing, then there's less time for the processing only it can do.

One big advantage of distributed processing is letting the host computer concentrate on message processing without getting tied down in the details of communications. Another advantage is that you spread your data base among several computers, often making data access faster and more responsive. As we said earlier about setting up the terminals at key locations - consider a warehouse having a computer dedicated to

keep inventory records up-to-date. When someone at the warehouse needs inventory information, it's right there. People in other parts of the company still get inventory information from the warehouse computer through a communications system.

Instead of a master-slave relationship among terminals as shown for the UTS 40 single station, the communications line can be connected to a terminal system consisting of a control unit with attached terminals. Input and output devices and any intelligence of a terminal are transparent to ICAM. ICAM works with the terminal control unit, not directly with the device:



The concept of the remote control unit with attached terminals is implemented with the Universal Terminal System 4020 (UTS 4020) and 4040 (UTS 4040) Cluster Controllers. You can connect many UTS 20W or UTS 40W terminal workstations and peripheral devices to these cluster controllers shown in Figure 3-2.

The cluster controller is a device that can control the input/output operations of numerous interactive terminals and peripherals. It is microprocessor-based and programmable. It controls both interactive communications and peripheral operations between the host processor and the display terminal workstations.

Writing to a terminal is like mailing a letter. You, through your program, write the message and put an address on it. Then you mail it by giving it to ICAM, and ICAM delivers the message to the right location. But just as the post office delivers mail to a mail room and not to individual departments within a company, ICAM delivers messages to a terminal controller and not to individual output devices or a processor.

When you write your programs, you must keep the hardware of the terminal in mind. In some cases, you must supply the control characters needed by the device, like screen format characters or forms control characters. (ICAM provides device independent control expressions (DICE) to ease this problem.) Also, whatever processing your program does must complement the processing done by the terminal.

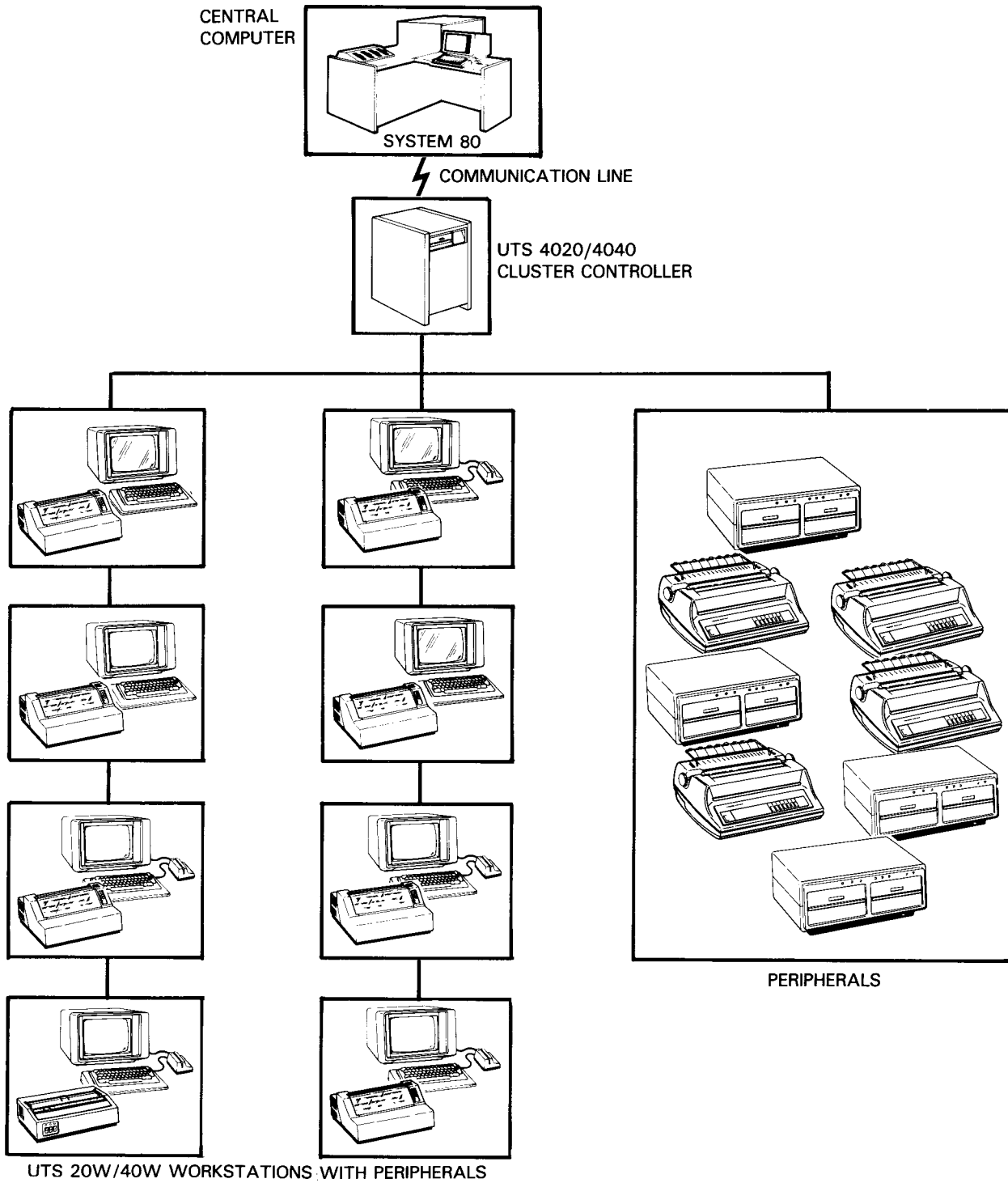


Figure 3-2. UTS 4000 System Connected to System 80

### 3.2.2. How Are Terminals Used?

Terminals are often separated into several categories based on use. Unfortunately, these categories are ill defined; a few terminals don't fit in any category, and many terminals fit in two or more. Keeping this in mind, here are the categories:

- *Real-time terminals* - Usually these are monitoring or process control instruments. A terminal that monitors air pollution would be real time, as would one that controls a heating plant. ICAM works with real-time terminals as long as they produce digital signals.
- *Interactive terminals* - These terminals allow you to carry on a conversation with a program in the host computer. Normally, you enter a message and the program responds with one. For example:

```

HOST:      WHAT INFORMATION DO YOU WANT TO WORK WITH?
USER:      CREDIT
HOST:      PLEASE ENTER CUSTOMER NAME AND ACCOUNT NUMBER;
USER:      PAULVILLE CANDY _6743
HOST:      CREDIT RECORD FOR PAULVILLE CANDY IS;

HOST:      CREDIT LIMIT; 10,000 CREDIT BALANCE; 534
           A/R TERMS; 30 NET 45 A/R DAYS 30 A/R DISCOUNT; .080

HOST:      DO YOU WISH TO CHANGE ANY OF THIS INFORMATION?
USER:      YES
HOST:      PLEASE ENTER FIELD NAME FOLLOWED BY NEW INFORMATION;
USER:      CREDIT LIMIT; 12,500

```

The basic interactive terminal almost always has a keyboard for entering data or messages and a printer or a cathode-ray tube for displaying them. Onto this, any number of auxiliary devices can be added. Many of these may be batch devices, such as tape cassette systems, printers, or paper tape readers/punches.

- *Batch terminals* - These terminals send and receive large amounts of data at a time. The data could be the code for a program or the data for a program. Unlike interactive terminals, batch terminals are rarely used conversationally.

The type of terminal tends to have some effect on ICAM. Real-time and interactive terminals typically send short messages with a few seconds to minutes between them. Batch terminals typically send long messages, often consecutively. When you create the ICAM line buffers (see 5.4), you must take the length and frequency of the messages into account.

Interactive terminals attached to a distributed communications processor (DCP) in a Unisys DCA network are supported by the ICAM standard (STDMCP) interface and the transaction control interface (TCI) for IMS interface. Batch terminals are only supported by the ICAM standard interface.

Table 3-1 summarizes the interactive and batch terminal system supported by ICAM.

**Table 3-1. ICAM-Supported Terminals and Workstations**

<b>Terminals</b>
<b>Interactive Environment</b>
UNISCOPE 100, 200
UTS 400
UTS 4000
UTS 20 Single Station (not programmable)
UTS 40 Single Station (programmable)
SVT 1120
UTS 4020 Cluster Controller with UTS 20W or 40W Workstations
UTS 4040 Cluster Controller with UTS 20W or UTS 40W Workstations
UTS 10 (character/teletypewriter mode)
IBM 3270 Terminal System (3271 controller)
TELETYPE <sup>®</sup> Models 33, 35, 37
UTS 20D (local workstation)
UTS 20 (remote workstation)
UTS 30 (remote workstation)
UTS 40D (local workstation)
UTS 40 (remote workstation)
<b>Batch Environment</b>
IBM 2780/3780/3741
UDS 2000 (IBM 3741 or 2780 emulation mode)

---

TELETYPE is a registered service mark of Teletype Corporation.

### 3.2.3. Interface Characteristics - How Terminals Communicate

Terminals communicate in different ways. They format their messages differently, encode them differently, and transmit them differently. It's here that ICAM conforms to your terminals. As we look at the interface characteristics, you should realize they're like items on a menu. Any one terminal uses some, but not all, of these characteristics. Even with a particular terminal, its interface characteristics aren't necessarily fixed. The BC/7 terminal-minicomputer, for instance, can act like a DCT 1000 terminal, a DCT 2000 terminal, a 1004 card processing terminal, a 9200/9300 terminal, an IBM 2780 terminal, a HASP terminal, or as itself. Because of this, we can't describe typical terminal interface characteristics any more than we can describe a typical meal.

Also, your computer must match the characteristics of the terminals it works with. If a terminal formats and transmits its messages in a certain way, then so must your computer. It's the job of ICAM and the single line communications adapter to handle the interface characteristics for your computer. Most of these characteristics do not affect your programs in any way.

#### Message Formatting

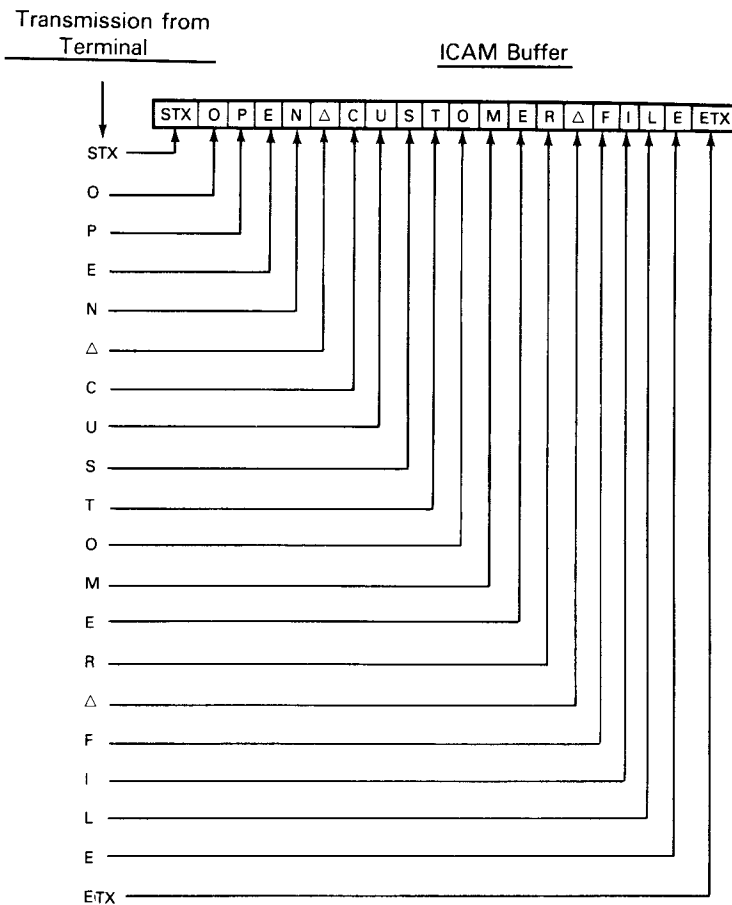
One of the most important characteristics of a terminal is the way it formats messages. At the heart of every message is the text, which may be a program instruction, like OPEN CUSTOMER FILE, or it may be program data, like H8954 12 BRASS BEDS 175.95. Text is like a letter; it requires an envelope before it is sent. Within this envelope, there are framing characters. Most of your terminals send a message with two framing characters. The first character, which is optional, is the start-of-text character (STX) preceding the text. Following the text is always the end-of-text character (ETX). So the minimum message formats with the framing characters look like this:

```
text EOT
or
STX text ETX
```

Our sample texts now look like this:

```
OPEN CUSTOMER FILE ETX
or
STX H8954 12 BRASS BEDS 175.95 ETX
```

Terminals using these formats are usually unbuffered - that is, they cannot temporarily store a message before transmitting or displaying it. These terminals transmit or receive one character at a time. As ICAM receives the message, it stores each character in a buffer until it has the complete message; for example:



**NOTE:**

The symbol Δ represents space characters transmitted as part of the message.

Unbuffered terminals have limited capabilities. You can't edit a message before sending it. It's difficult, but not impossible, to have more than one unbuffered terminal on a line. The solution is to put buffers into terminals. When it's buffered, a terminal adds more framing characters to the basic STX text ETX format. The first character in the header is the start of header (SOH). Most Unisys terminals then place a 3-character terminal address following the SOH. The address consists of:

- Remote identifier (rid), which identifies a group of terminals
- Station identifier (sid), which identifies a particular terminal in a group
- Device identifier (did), which identifies a particular input or output device on a terminal

More is said about the terminal address in 4.3.



A typical message header looks like this:

SOH rid sid did

The text is still preceded by a start-of-text character. But now, control information may be inserted in the text. For example, the UTS 400 terminal sends the location of the cursor before the text of the message. The format of the cursor location is:

ESC VT Y X SI

where:

ESC (escape)  
Specifies that the following characters are part of a control sequence.

VT  
Specifies that the next two characters are the cursor address.

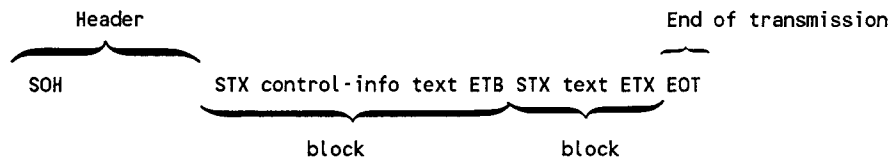
Y  
Is the Y coordinate identifying the horizontal line (or row) on the screen where the cursor is placed.

X  
Is the X coordinate identifying the vertical column on the screen where the cursor is placed.

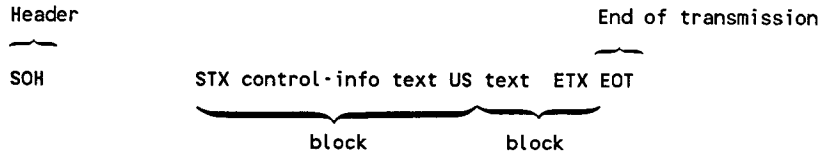
SI  
Indicates the end of the cursor address sequence.

This is an example of only one type of control information used by one kind of terminal. Other terminals place control information relevant to their hardware in the text.

Batch terminals generally send messages in blocks. To separate the blocks of text, they use an end-of-transmission block (ETB) character. One way to do this is to start each block with the start-of-text character and end it with an ETB character. Batch terminals do not require a terminal address (rid, sid, did). In this format, each message has three parts - a header, one or more blocks of text, and an end-of-transmission character, as shown here:



Unit separate (US) characters are also used to separate blocks:



As ICAM receives each block, it puts it in a buffer:

Figure 3-3 summarizes the message formats.

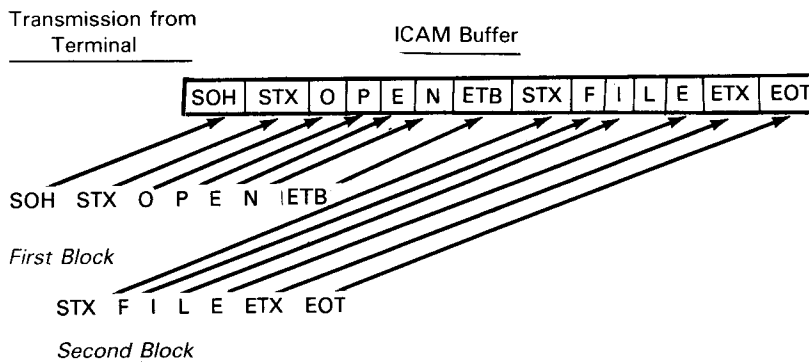


Figure 3-3. Common Message Formats

Looking at Figure 3-3, we can examine some of the message formats - going from the simple to the complex.

The simplest message is a string of individually transmitted characters comprising the text followed by an end-of-transmission character.

text EOT

More common is a message in which the text is flanked by a start-of-text character and an end-of-text character.

STX text ETX

Some terminals prefix the start-of-text character with the address of the terminal. The address consists of a remote identifier, a station identifier, and a device identifier.

rid sid did STX text EOT

More sophisticated terminals prefix the text with a header composed of a start-of-header character and the address of the terminal. The terminal frames each block of text with a start-of-text and an end-of-text character. The text consists of control information and the text itself. An end-of-transmission character appears at the end of the message.

SOH rid sid did STX control-info text ETX STX text ETX EOT

Another way of blocking messages is to separate each block of text with a unit separator. In some formats, the end-of-transmission character isn't needed, and the message ends with an end-of-text character.

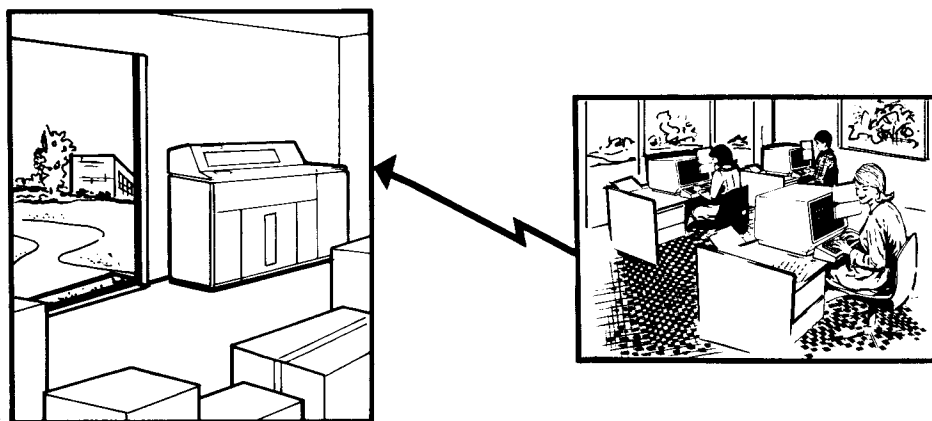
SOH rid sid did STX control-info text US text US text ETX

On output, your programs write the text and provide the control information. ICAM takes this core and builds the rest of the message around it, adding the header, start-of-text character, end-of-transmission character, and whatever else the format requires. On input, ICAM strips this away and gives your program the text and control information.

### Communications Direction

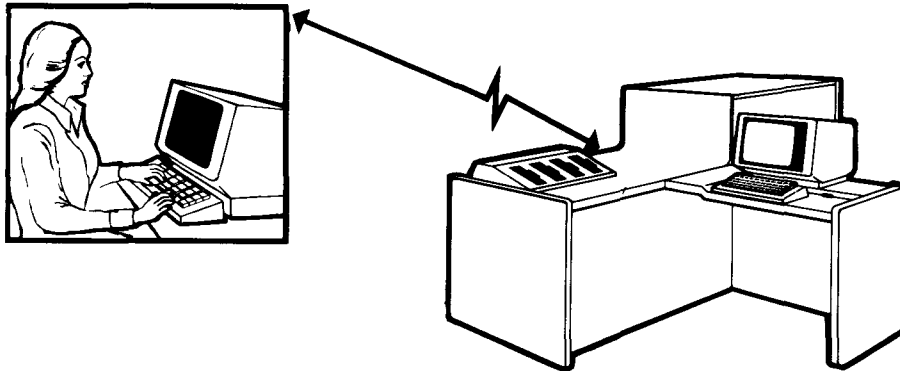
So far, we've talked about terminals as if they all send and receive. While most do, some are send-only and others are receive-only. Among the terminals that send and receive, a few can do both at the same time. Based on the directions of communications possible, there are three types of terminals:

1. *One-way (simplex) terminals* - Either send or receive but not both. You might, for example, want a terminal on a loading dock for printing shipping orders. Because your program isn't interested in receiving messages from the loading dock, a receive-only printer is installed. A picture of a simplex operation could look like:

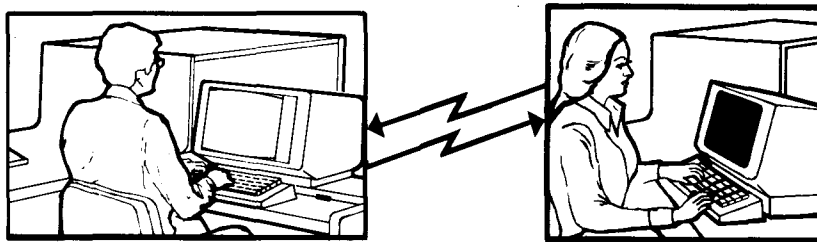


The only difference between this terminal and one that sends and receives is this one doesn't have the circuitry for sending messages. A send-only terminal is just the opposite: It doesn't have the circuitry to receive messages.

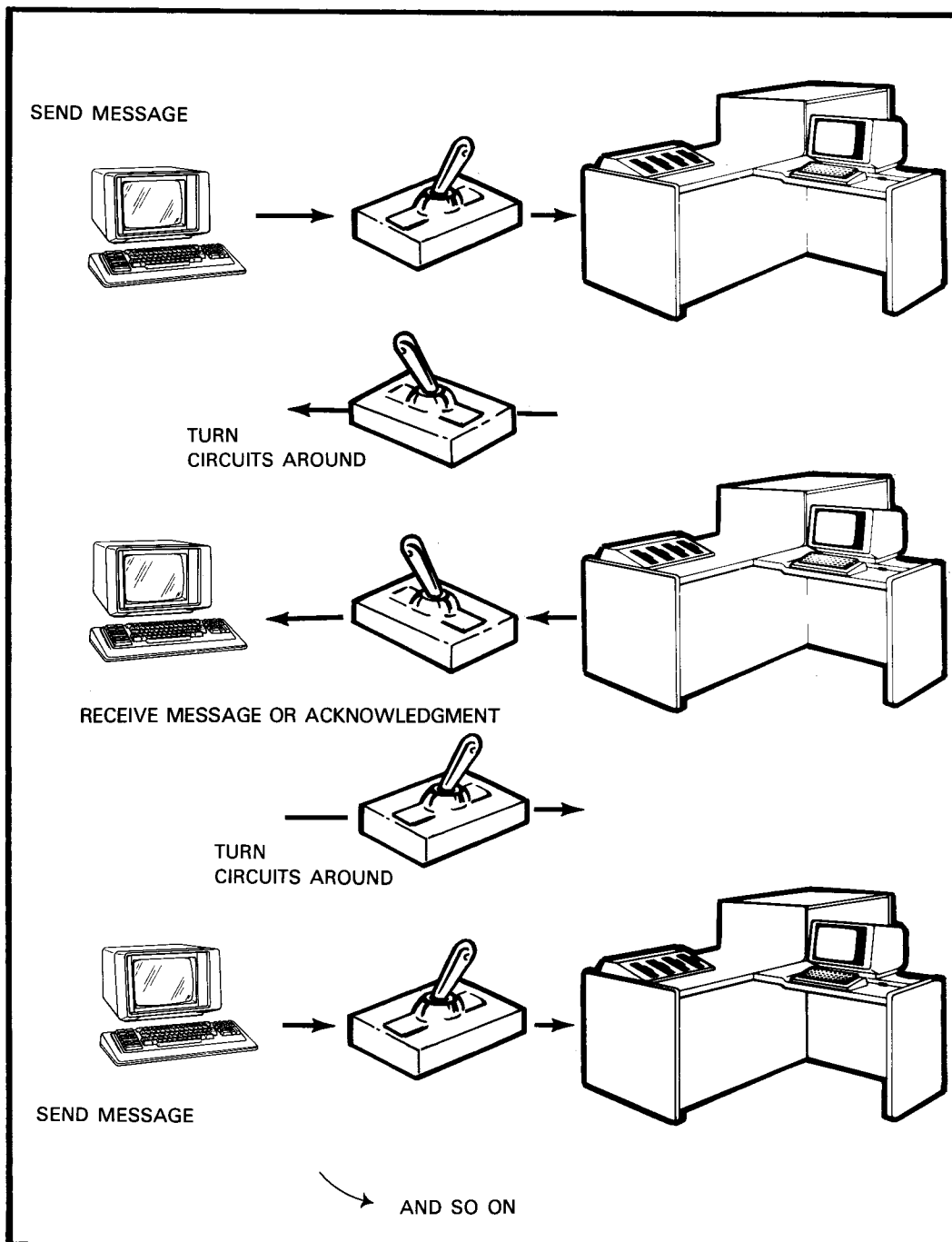
2. *Two-way alternate (half-duplex) terminals* - Receive and send, but not both at the same time. They're like a citizen's band radio. When you send, the circuitry is tied up and the terminal can't receive. When you receive, the circuitry is again tied up and you can't send. Most terminals are 2-way simultaneous. You can use a half-duplex terminal as a simplex terminal by not using its send or receive capabilities. A picture of a 2-way alternate operation could look like:



3. *Two-way simultaneous terminals (also called duplex or full-duplex terminals)* - Send and receive simultaneously. Basically, they combine the circuitry of a receive-only terminal and a send-only terminal into a single unit that receives on one line and sends on another. The 2-way simultaneous terminals send and receive messages faster than 2-way alternate terminals even when they use identical lines. This is because acknowledgment signals can be received on one channel while data is sent on the other. This saves time because the line doesn't need to be turned around each time to acknowledge a message. A picture of a 2-way simultaneous operation might look like:



When most terminals send a message, they expect the host computer to return an acknowledgment and vice versa. With a 2-way alternate terminal, the circuits must be turned around before an acknowledgment is made. The process is:



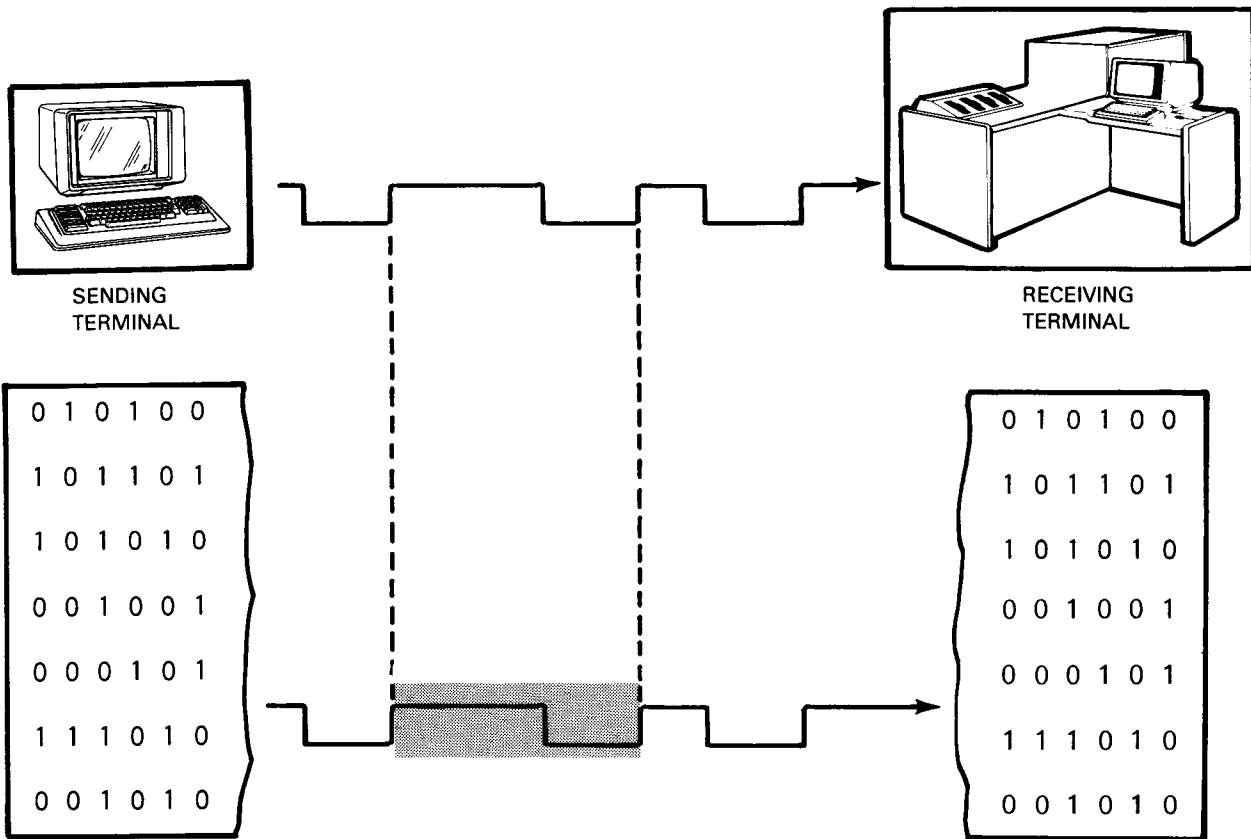
Looking at this picture, we see that every time the circuits are turned around, it wastes time. With 2-way simultaneous terminals, the circuits need not be turned around, and the turn-around time is used to send messages.

ICAM controls the communications direction with the terminals. It simply needs to know what kind of terminals you're using. When you generate ICAM, you must declare a terminal as either half-duplex or duplex. Simplex terminals are declared as half-duplex since your system is designed to accommodate more sophisticated transmission.

The 2-way alternate and 2-way simultaneous terminals are indistinguishable to your programs. If you have simplex terminals, then your programs can only send to a receive-only terminal or receive from a send-only terminal.

### Synchronizing Transmission

Another problem is synchronizing transmission between the terminals and the host computer. Without getting too deeply into the technicalities of synchronization, the problem is actually two related problems. First, terminals send messages in bits that may last for a millisecond or less. If the sending terminal is slightly out of synchronization with the receiving terminal (remember the host computer is a terminal), bits are lost. The following illustration shows two devices that are synchronized.



Second, if the receiving terminal misses one or more bits, possibly because of line interference, it can't tell which bits belong to which character because a typical transmission is just a string of bits without markers between characters. For example:

SENDING

JOE IS HERE

RECEIVING

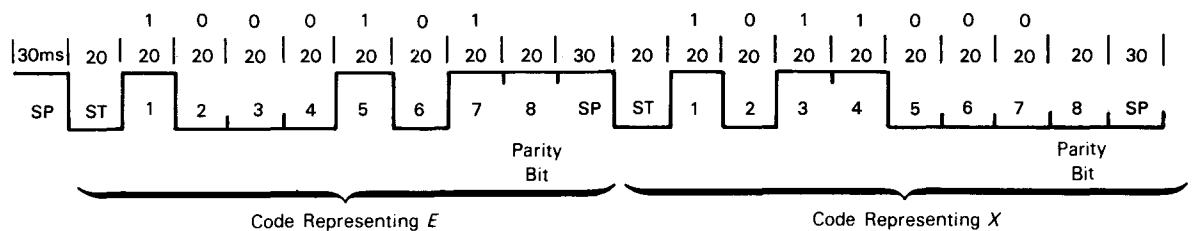
JOE I H E

Here, since there are no markers, the receiving terminal does not know where the first word starts or last word ends. JOE could be a fragment or the entire word, as well as I, H, and E.

One method of synchronizing terminals is misleadingly called asynchronous transmission.

Asynchronous doesn't mean the terminals aren't synchronized - it means that each character is synchronized rather than entire blocks of characters as in synchronous transmission.

When you send the character in an asynchronous system, a *start* and a *stop* bit are added to each character by the hardware. Whenever the receiver detects a start bit, it knows that a character is coming in, and it knows when to look for the next bit. In this manner, the sender and the receiver are synchronized. Most terminals using asynchronous transmission also follow the last bit in each character with a stop bit. So, if the letters E and X were sent asynchronously in ASCII, the line transmission might look like this:



LEGEND:

SP Stop bit  
ST Start bit

What distinguishes the stop bit from any other bit is its duration; it lasts 30 milliseconds while every other bit lasts 20 milliseconds. There are other ways of using asynchronous transmission. The start bit can be no longer than any other bit or the stop bit can be eliminated entirely. Normally, asynchronous transmission is associated with short data transfers such as queries and acknowledgments.

The other way of synchronizing terminals is through synchronous transmission. Terminals using this method are more sophisticated than asynchronous terminals. Once the sender synchronizes with the receiver, they remain synchronized for up to hundreds of bits. Entire messages or blocks of messages are transmitted before the terminals need resynchronization.

In synchronous transmission, two or more synchronizing (*sync*) characters precede message transmission. When the receiver and sender recognize they are synchronized, message traffic begins. A message with the text EXECUTE might look like this in synchronous transmission:

```
sync sync sync sync SOH rid sid did STX E X E C U T E ETX EOT
```

The single line communications adapters handle synchronization for the host computer. ICAM needs to know whether you are using asynchronous or synchronous transmission, but it has no role in synchronizing transmission. Synchronization has no effect on your programs.

### Line Control

Line control is a major terminal interface characteristic. At any given moment, a line is used to either receive messages from a terminal or send them to a terminal. The host computer and terminal can't send at the same time. If they try, both of the messages are lost. The solution is to let the host computer control message flow on a line. ICAM, through the communications hardware, determines whether a line is set for receiving messages from or sending messages to the host computer. As with every other terminal interface characteristic, there's more than one way of controlling a line.

The first is called *uncontrolled*. ICAM controls the direction of message flow, but it doesn't control when the terminal sends - hence, the name uncontrolled. Normally, unbuffered, asynchronous terminals are uncontrolled lines.

In an uncontrolled system, ICAM constantly has the line ready to receive messages from the terminal except when the host computer is actually sending to the terminal. In other words, you, at the terminal, can send a message to the host computer any time it's not sending to you. If you try sending a message when the host computer is sending, the message is ignored because the line is set the wrong way.

Uncontrolled lines make multiterminal lines impractical. This leads us to the second method of controlling a line, appropriately called *controlled*. Just as *uncontrolled* refers to when a terminal can send, so does *controlled*. Put simply, on a controlled line, terminals speak only when spoken to.

The heart of a controlled system is a set of rules - protocols - governing all transmissions between the host computer and a terminal. While protocols vary from terminal to terminal, the basic rule of all of them is that the host computer initiates all traffic on the line through the use of polls. A poll, which is a message sent by the host computer to a terminal, requires a response from the terminal.



Most protocols have at least the following kinds of polls:

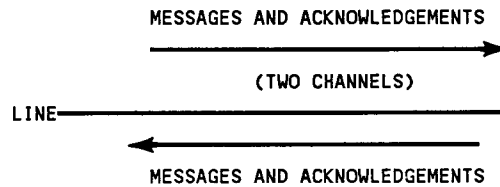
- A *status poll* asks a terminal its status. Typical responses might be the equivalents of "I'm busy"; "I have a problem and cannot accept any messages"; "My diskette drive is down"; or "I can accept a message". Status polls establish what devices in a terminal are up or down and whether the terminal can accept a message.
- A *traffic poll* asks a terminal to send any messages it has ready to send. The terminal sends either a message or a response saying it has no messages.
- A *retransmission request poll* asks a terminal to retransmit the last message it sent. Let's look at how polling works between ICAM and a UTS 400 terminal. For details of UTS 400 protocol and poll formats, see the *UTS 400 Programmer Reference* (UP-8359).

ICAM starts by sending a general poll to all the terminals on the line:

<u>Poll</u>	<u>Response</u>	<u>Description</u>
SOH 1 P p ETX	SOH 1 a p STX text ETX	The poll consists of five characters: a start-of-header character, a 3-character address, and an end-of-text character. Even though the poll doesn't have any special characters, a terminal understands it means, "Does any terminal have a message to send?"  The terminal with the address 1 a p responds by sending this message.
SOH 1 P p DLE 1 ETX	EOT EOT ETX	Once ICAM has the message, it returns with a poll acknowledging the receipt of the message and asking if any terminal has another message.  In this case, no more messages are waiting, so the multiplexer on the line responds with a no-traffic message.
SOH 1 a p STX text ETX	SOH 1 a p DLE 1 STX text ETX	Because there is an output message waiting, ICAM sends a message to the terminal with the address 1 a p.
SOH 1 a p ETX		ICAM sends out a poll asking the terminal if the message was received without error.  The terminal then responds with a positive acknowledgment and a new message.

Terminals on controlled lines usually have internal buffers to hold messages until ICAM polls them. Normally, these terminals send synchronously.

For 2-way simultaneous terminals, line control (through polling) establishes terminal status and acknowledges messages. The terminal and host computer can send simultaneously; one sends messages (and acknowledgments) down one channel of a line as the other acknowledges messages (and sends other messages) on the other channel of the line. The advantage of 2-way simultaneous terminals over 2-way alternate terminals is the line doesn't have to be reversed after each message or poll, saving considerable time. However, it doubles the circuitry and the cost for the equipment must be weighed against telephone line usage charges.



### Additional Terminal Interface Characteristics

In addition to the interface characteristics discussed in the previous subsections, two more affect terminal interfaces in a communications system. The first is the transmission rate, measured in bits per second, at which the terminal sends or receives. Unless you're working with an exceptionally fast terminal, the communications hardware takes care of this. If you have a very fast terminal, you should declare extra buffers (see Section 5). The way a terminal encodes its messages is the other interface characteristic. Most of the newer terminals use 7-bit ASCII code established by the American National Standards Institute (ANSI). Another is 8-bit EBCDIC, which your computer uses internally. When a terminal uses any supported code but EBCDIC, ICAM performs the translation from that code to EBCDIC.

To work with a terminal, ICAM uses a routine called a remote device handler written to match the interface characteristics of the terminal. The remote device handler takes care of message formatting, line control, and code translation. Because each terminal is different, each remote device handler is different. ICAM has built into it the remote device handlers for the terminals listed in the next paragraph. If you decide to use a terminal not on the list, its interface characteristics must be the same as one of the terminals on the list, or you'll have to write your own remote device handler to support it. For more on remote device handlers, see 4.1.

## 3.3. Communications Lines

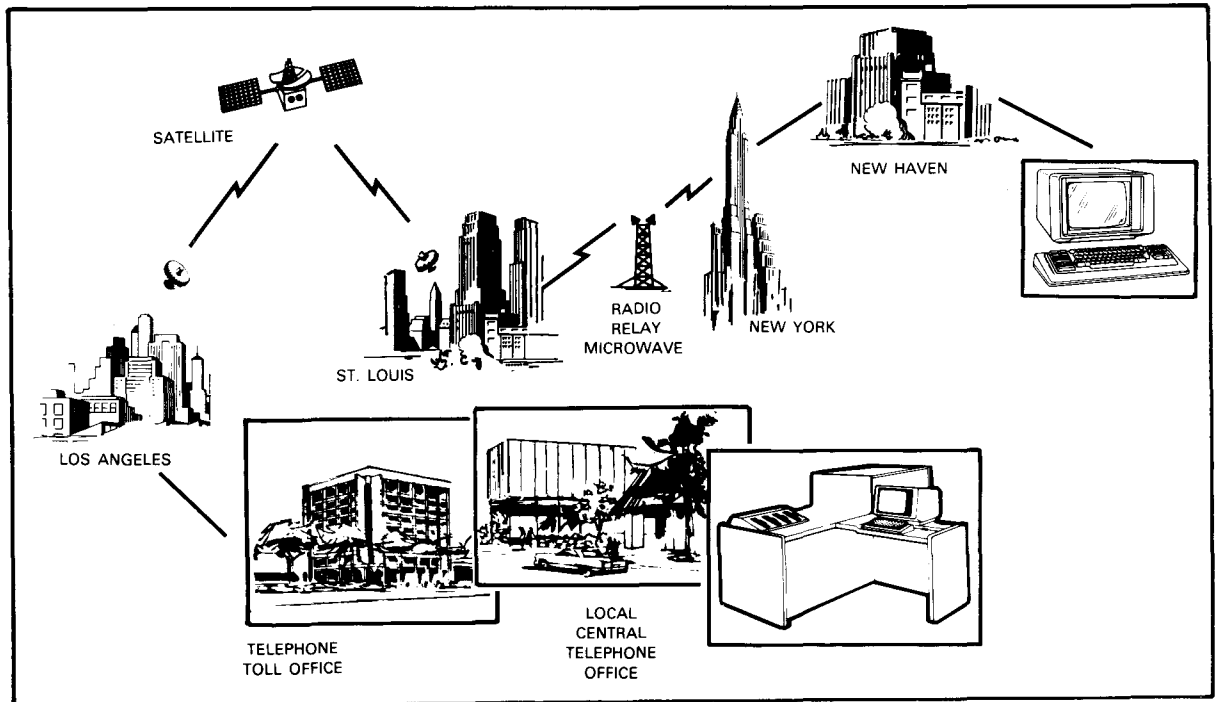
Throughout this section, we've used the word *line* without really defining what it is. That's because a line is merely the communications link between a host computer and a terminal that allows them to exchange messages. You'll notice that the definition ignores the mechanics of establishing a line - you can do that by stringing wires between your host computer and terminals, by using the phone system, or by using a satellite relay. A line is not the communications hardware that establishes a circuit capable of carrying a message. Rather, it's a logical idea saying that a host computer and a terminal are somehow tied together and can communicate.

We aren't going to look very hard at lines, primarily because the way you establish them doesn't have much effect on your software or your hardware. But we will look at those ways communications lines influence ICAM.

To have a line between a computer and terminal, there must be a circuit tying them together. The simplest kind of circuit is established by stringing wires between the computer and terminal. Over distances of less than a few tens or hundreds of miles, this is probably the way your circuits are established. But stringing wires is expensive, and the signal must be boosted every few miles. This problem is eliminated by sending messages via high frequency microwaves between a series of relay transmitters or, more recently, through communications satellites.

If your terminals and host computer are close together - less than 2000 or 3000 feet - you can set up your own lines by connecting them with cables. Beyond short distances, the cost of stringing the cables and obtaining the rights-of-way becomes prohibitive. At this point, you use the facilities of a common carrier, a company whose business is to provide communications services. Until a few years ago, the telephone and telegraph companies were the only existing common carriers, and they're still the biggest. In the last decade or so, however, these common carriers were joined by a number of new companies specializing in computer communications.

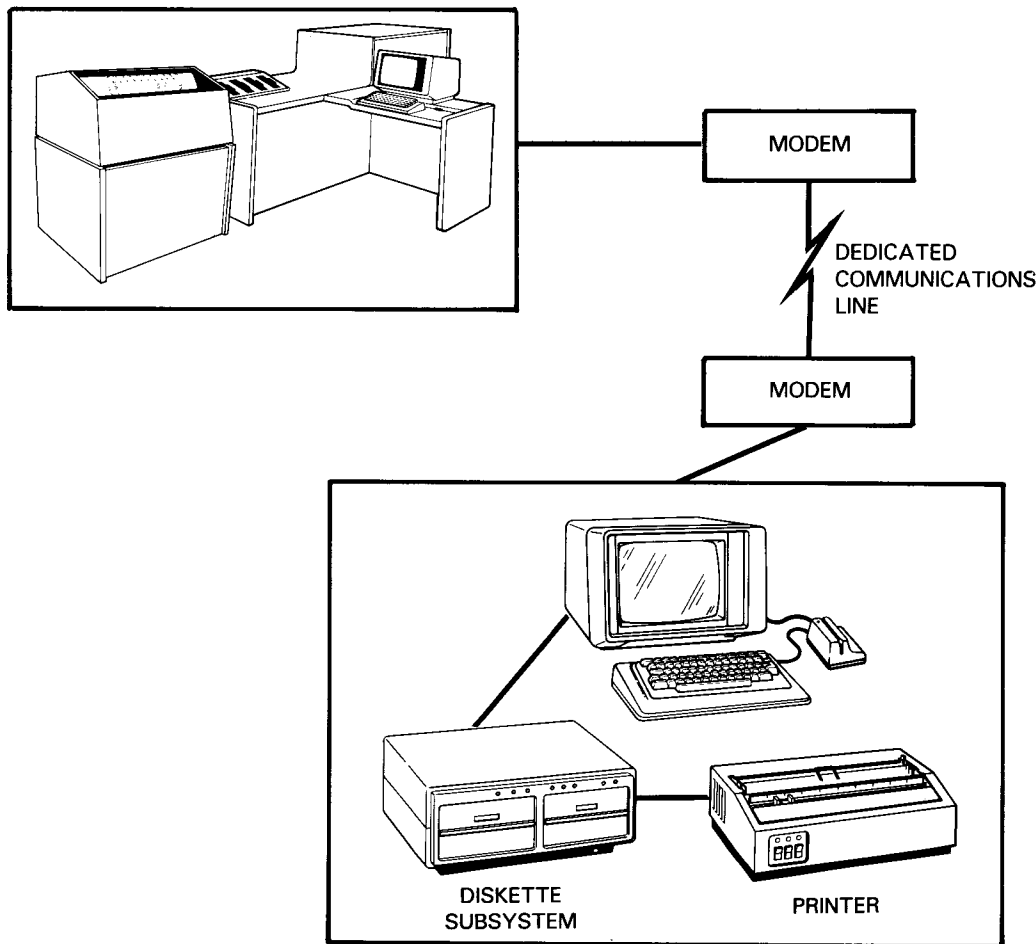
All the common carriers tie available circuits together to create one circuit connecting your computer and terminals. Often, the circuits combine different types of communications facilities, as in this example:



Most books on communications cover the circuit hardware in detail. From an overall perspective of data communications, this is important. From the perspective of software, it's not. Neither ICAM nor your programs are any more aware of the physical circuit than you are when you make a phone call. The common carriers offer three basic kinds of service: dedicated circuits, switched circuits, and public data networks.

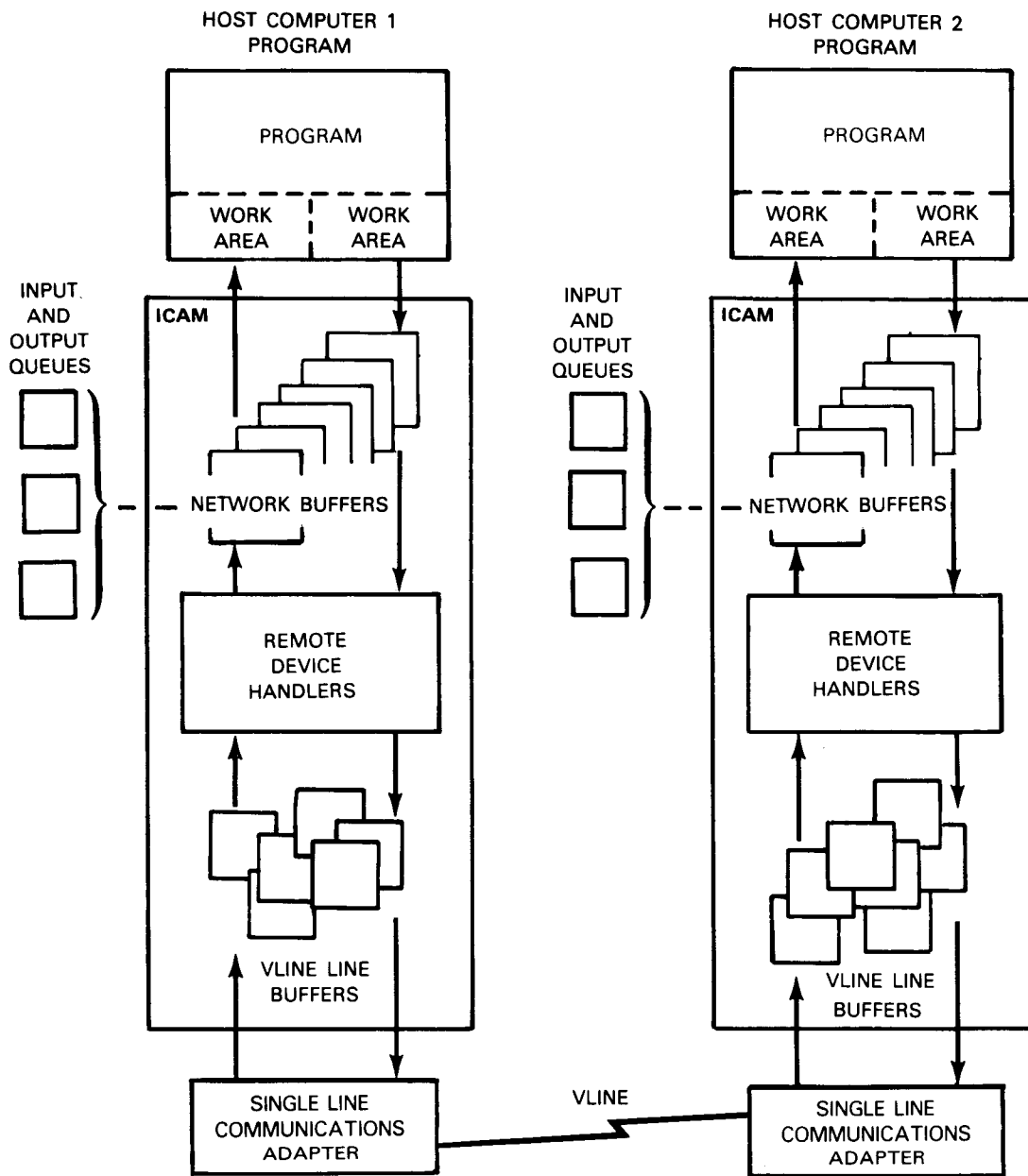
### 3.3.1. Dedicated Circuits - Dedicated Lines and VLINEs

Dedicated circuits - also called dedicated lines, private lines, and leased lines - are the simplest to understand. In this case, either you or a common carrier creates a permanent circuit that's always ready to carry your messages. As long as the equipment on both ends of the circuit is turned on, the line is established. An example of the use of a dedicated circuit is this UTS 40 single station terminal, which is always connected and ready to operate:



One special use of a dedicated circuit is a VLINE. VLINEs are used to connect ICAM global networks located in different computers. They are also used to connect an ICAM global network to a public data network (PDN). Dedicated circuits used in this way always operate in 2-way simultaneous (full-duplex) mode.

The term VLINE or virtual line comes from the way the circuit is used (that is, many logical circuits can be carried on a single dedicated circuit) not how the line is physically constructed.



NOTE:

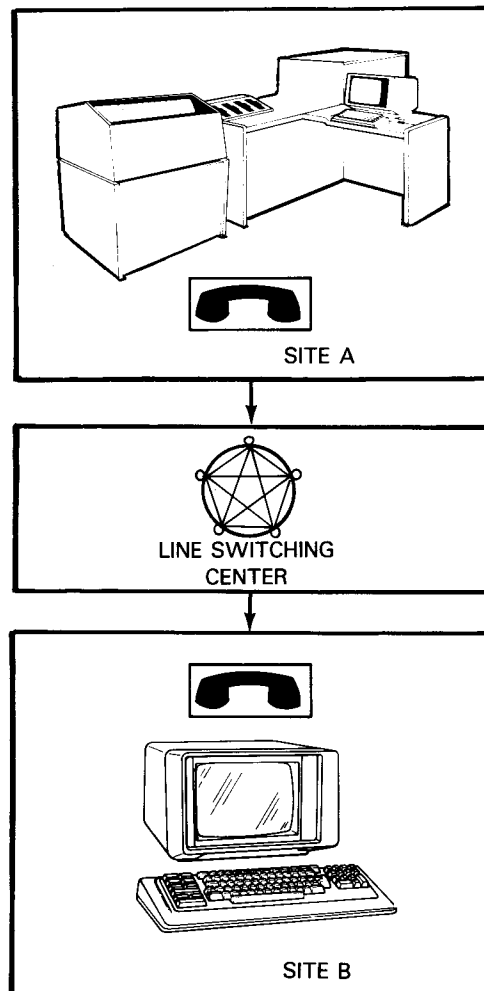
Additional global elements are introduced in subsequent subsections.

ICAM uses the following protocols with VLINEs:

- X.25 link access procedure defined by the Committee Consultative International Telephone et Telegraphic
- X.25 link access procedure B defined by the Committee Consultative International Telephone et Telegraphic
- Universal Data Link Control Asynchronous Balanced Mode described in the *Universal Data Link Control General Description (UP-8554)*.

### 3.3.2. Switched Lines

Circuit connections over switched lines are established each time you want to connect a terminal to your host computer. If you use the telephone system, the telephone company's switching equipment connects available circuits to give you a line. You use the circuits until you hang up. When you do that, the circuit is broken and is available to other users. The next time you call the same location, you will probably get different circuits. A typical switched-line network is:



In most respects, dedicated and switched circuits work the same. Both tie your host computer directly to your terminals (or to another computer), giving the host computer control over the communications system. With both, the transmission rate of the line must be the same or greater than the transmission rate of your terminals. Also, the communications direction of the circuit must support the communications direction of your terminals: 2-way alternate circuits for simplex\* and 2-way alternate terminals (no common carrier offers simplex circuits), and 2-way simultaneous circuits for 2-way simultaneous terminals.

Some major differences exist between dedicated and switched circuits:

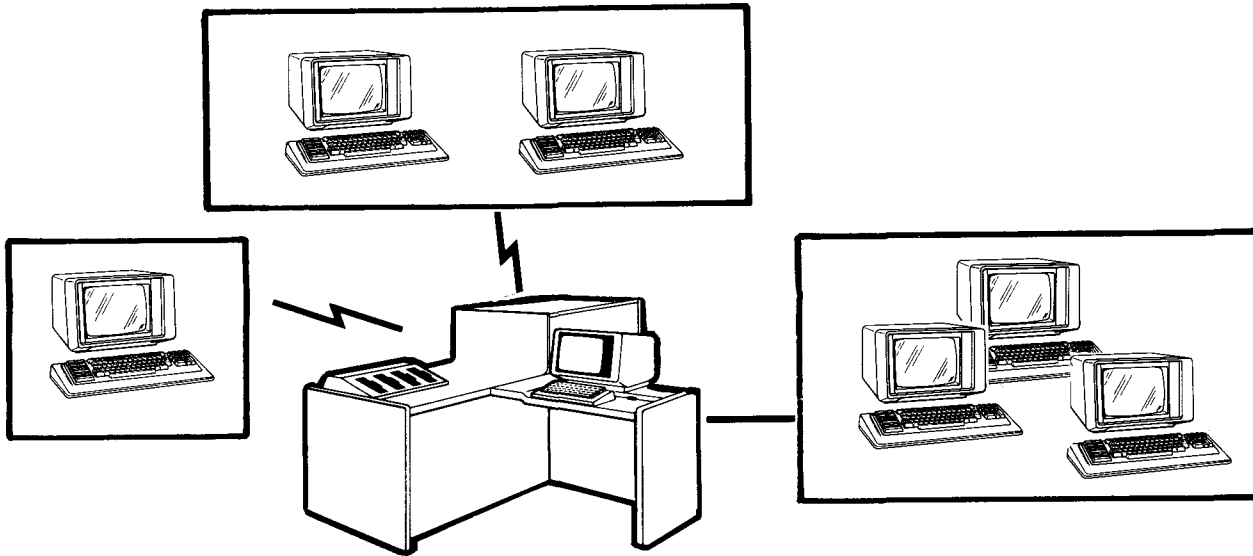
- Because different circuits are used each time a switched circuit is established, you never know what circuit quality will be. To give an example, sometimes when you make a phone call, you get a poor connection. While this causes you inconvenience, when this happens in data communications, the messages often become unintelligible. With dedicated lines, you always know the characteristics of your circuits. Also, with dedicated circuits, you can use equipment designed for data communications rather than for voice transmission - although you'll pay extra. You can't do this with switched circuits.
- If you need to use a line for short periods only, it's normally less expensive to use switched circuits. If you need a line for large parts of a day, it's less expensive to use dedicated circuits.
- With dedicated circuits, the line is always established, so there's no problem with connecting the circuits. With switched circuits, the line is reestablished each time it's used. This is done three ways:
  1. The computer operator dials out to the terminal.
  2. Automatic equipment in the communications hardware dials out to the terminal.
  3. The terminal operator dials into the processor.

If you use either a dedicated or switched circuit, it ties your host processor to your terminals so they transmit directly to each other without an intermediary. It's like talking on the phone - your words go directly to the person on the other end.

---

\* Simplex means data always flows in one direction only. A 2-way alternate terminal means that data flow alternates in both directions as necessary.

A drawing of a network using either kind of circuit would look like this:



### 3.3.3. Public Data Networks

In many countries, the government or private companies provide data communications network services that can be used by any organization willing to subscribe to the service. These services are called public data networks (PDNs). User equipment that accesses a PDN is called data terminating equipment (DTE). A DTE can be a computer, a programmable terminal controller, or an intelligent terminal. Within the PDN, the equipment that interfaces the data terminating equipment is called the data circuit terminating equipment (DCE). A DCE can be a data set, a modem, or any other equipment that provides the functions needed to transfer data between the DCE and the DTE. See Figure 3-4.

Two types of PDNs are supported by ICAM: circuit-switched and packet-switched. However, they must reside in separate load modules if both are going to be used.

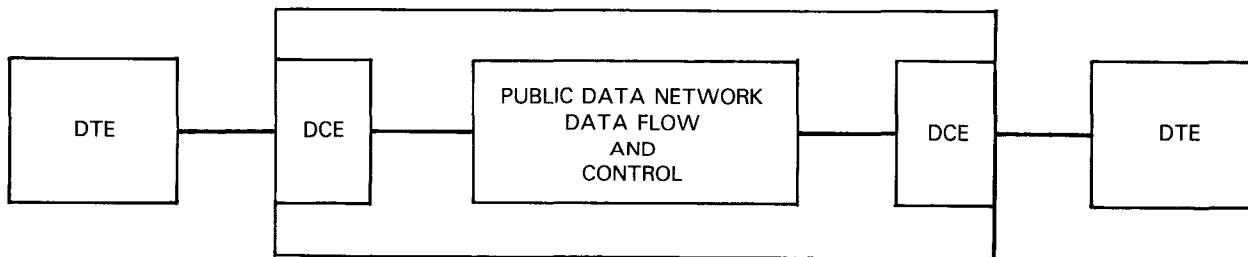


Figure 3-4. Public Data Network Basic Configuration



When a user of a circuit-switched PDN sends data to another end user, a temporary dedicated connection is established within the PDN. The data is sent in much the same way as over a leased line except that the line is connected only when necessary and it is disconnected when there is no more traffic. Usually, this connection can be established quickly.

In a packet-switched PDN, messages are broken into packets and then sent to their destination. Packets are blocks of data contained in a defined format with a maximum size that includes a header. The header controls the destination of the packet and identifies the sender. At the receiving end, the messages are reassembled from the packets by the PDN and sent to the receiving data terminal equipment. This process is known as packet assembler/disassembler (PAD).

Circuit-switched PDNs require the sending equipment to wait until a circuit connection is complete before message transmission can begin. Packet-switched PDNs allow message transmission to begin immediately, because all of the packets of each message are held by the PDN until the receiving data terminal equipment successfully receives it.

In addition to the way in which data is transmitted, circuit switching and packet switching have some other significant differences. One of these is the way PDNs charge for service. Circuit-switched PDNs usually charge for the time that a circuit is connected (connect time). Packet-switched PDNs charge only for the number of packets that they transfer.

### **Circuit-Switched Public Data Networks**

Circuit-switched PDNs provide dedicated connections between a host processor used as data terminal equipment and other data terminal equipment (such as a terminal) when message traffic is flowing between them. (See Figure 3-5.) Only one circuit-switched link can be used between System 80 processors.

When a message is ready to be sent, a circuit is quickly established through the PDN and transmission begins. When the transmission ends and the delay for tariff optimization expires, the physical connection is cleared. When a new message is ready to be sent, a new connection is established.

A circuit can be connected or disconnected in a fraction of a second. This is much less time than required with dialed facilities. Therefore, single circuit-switched links are shared to provide concurrent communications between a host and a number of terminals.

Figure 3-5 illustrates how the same communications link within a circuit-switched PDN can be shared by more than one set of data terminal equipment. In the top half of the illustration, a message is sent between host processor A (DTE A) and terminal B (DTE B). The PDN establishes the connection between the two DTEs, the data is sent, and the call is cleared. Now DTE A can send more messages to DTE B or it can send a message to DTE C.

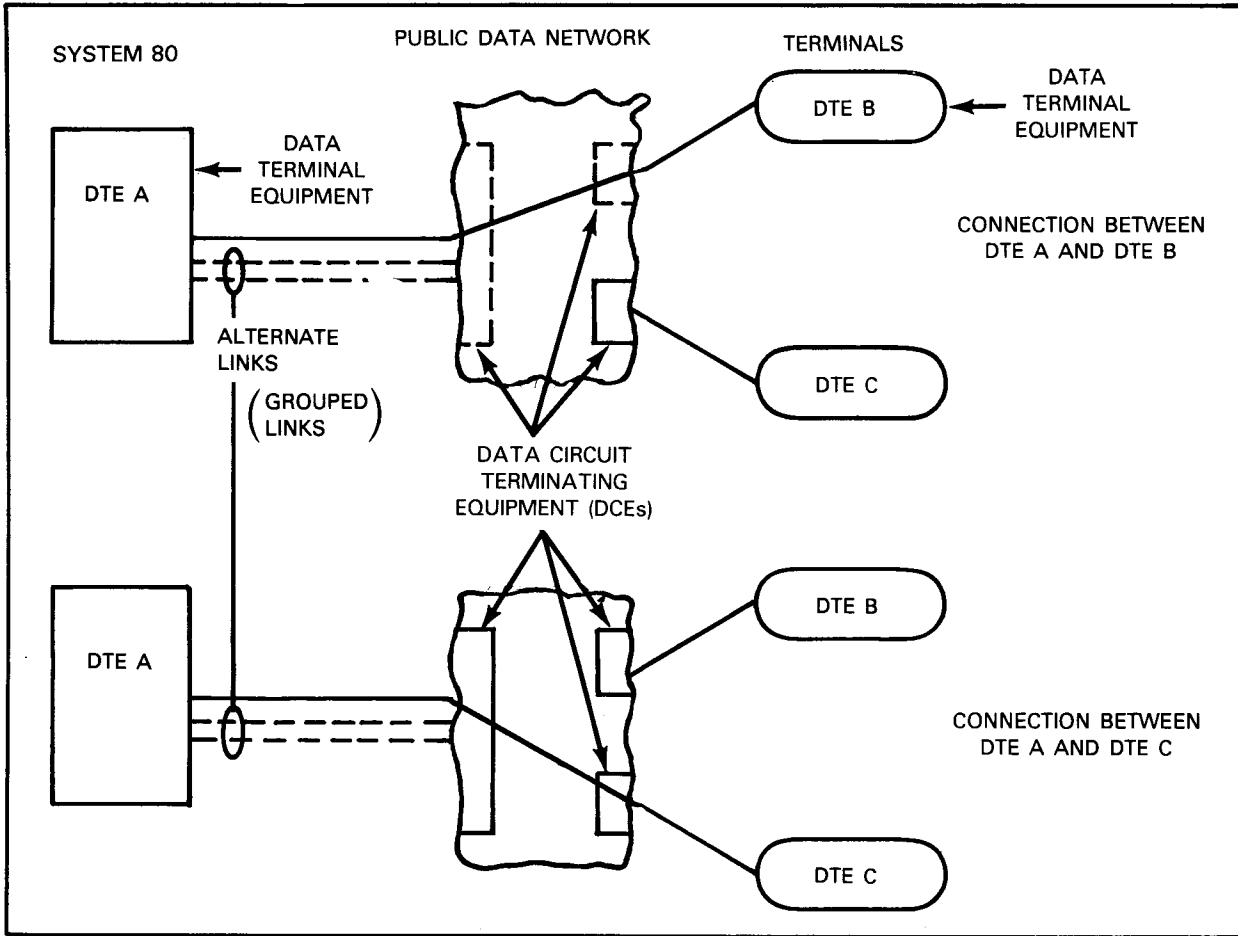


Figure 3-5. Circuit-Switched Public Data Network

You can have many terminals connected to the PDN; and this could cause busy conditions on incoming calls due to sharing of the link between the host processor and the PDN. You solve this problem by having several links between your host processor DTE and the PDN DCE. This capability is shown as a series of dashed lines in the figure.

These links may be addressed individually or as a group. To complete a connection, the called PDN DCE automatically selects an unused link when the call is directed to a group. Your DTE automatically selects an unused link when it sends a message.

The specifications for the currently supported circuit-switched networks are given in Table 3-2.

Table 3-2. ICAM Circuit-Switched Public Data Network Support

PDN Name	Physical Interface	Protocol	Country of Origin	ICAM Interface	SLCA Required
DATEX-L	RS-232-C	UNISCOPE	Germany	STDMCP	F2788-02 and F3794 Autodial
NORDIC	RS-232-C	X.21	Denmark, Finland, Norway, Sweden	DMI, STDMCP, TMI	F-2798-00

### Packet-Switched Public Data Networks

Packet-switched PDNs consist of switching nodes and high speed digital communications trunks (Figure 3-6). Under OS/3, ICAM acts as an interface between your program in your computer (the data terminal equipment or DTE) and the packet-switched PDN node (the data circuit terminating equipment or DCE).

ICAM allows users to establish sessions and transfer data between end users in different computers using OS/3. Dynamic sessions are supported between the following end users:

- *Local user program to remote user program* - Programs using the ICAM standard interface or IMS programs in a local OS/3 system can establish or disestablish sessions with similar programs in a remote computer using OS/3. In addition, they can make use of the distributed data processing (DDP) functions. (See 7.4.)
- *Local terminal to remote user program* - Terminals attached to a local System 80 computer can establish or disestablish sessions with a user program using the standard interface or an IMS program located in a remote computer.
- *Local user program to remote terminal* - Programs using the ICAM standard interface or IMS programs can establish or disestablish sessions with terminals in a remote computer running OS/3.

Sessions are not supported between the following end users:

- Terminal to remote terminal
- Terminal to remote process file
- Terminal to remote interactive services (locap file)

In addition, packet-switched PDN support requires the use of an ICAM global network. The 32-byte packet size is not supported, and the packet assembler/dissambler (PAD) is not supported.

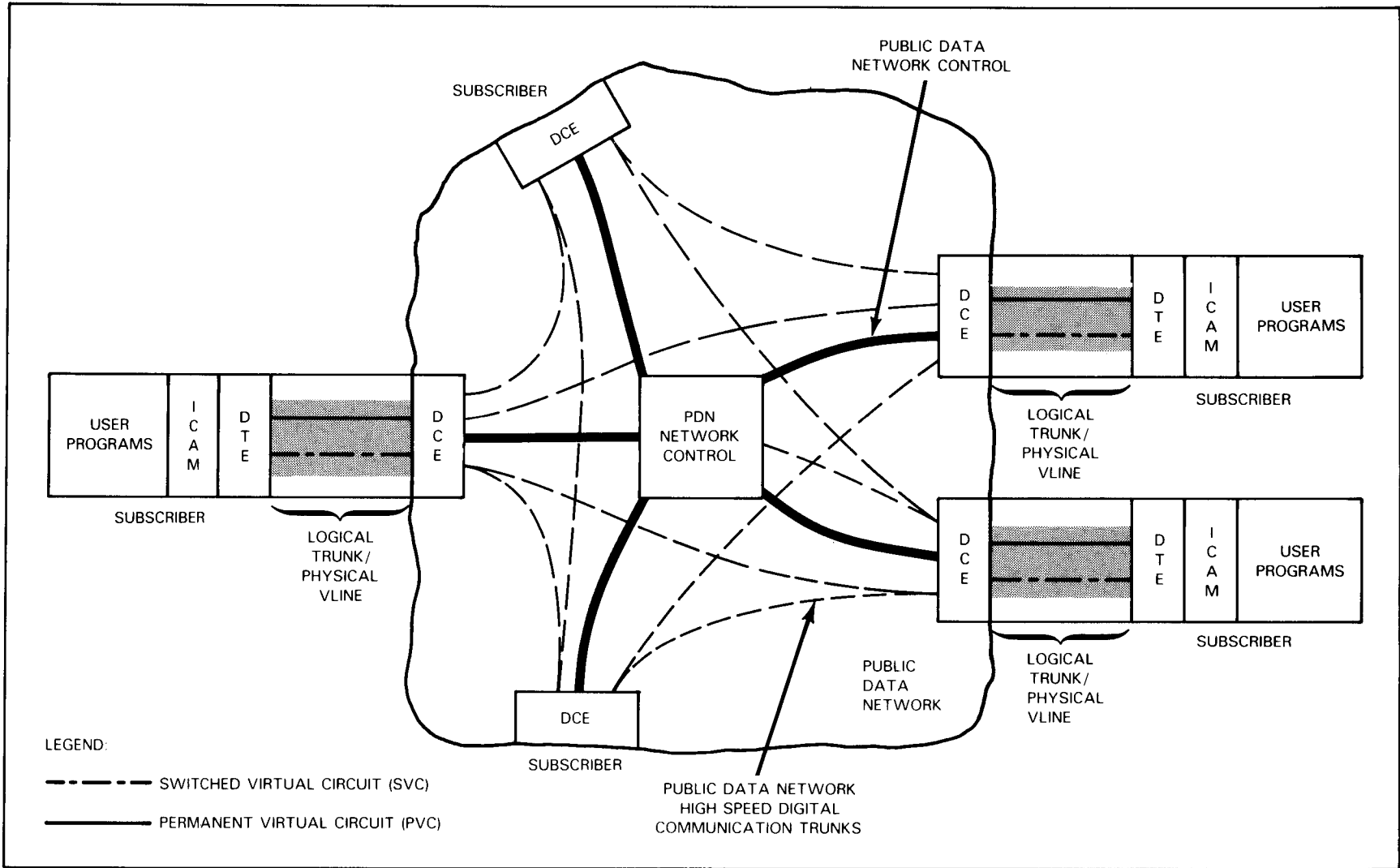
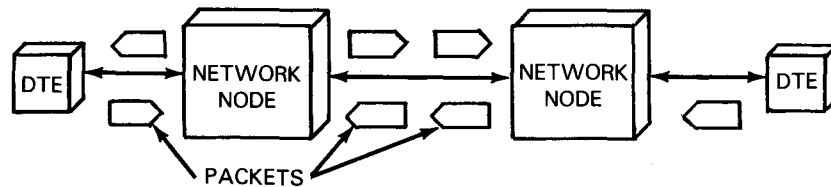


Figure 3-6. Packet-Switched Public Data Network

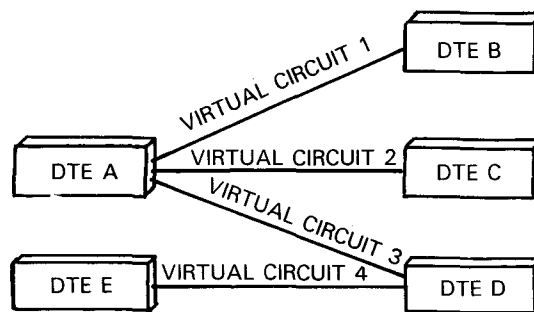
A virtual circuit is a connection between two end users of a packet-switched PDN for 2-way simultaneous exchange of data. Virtual circuits are established and normally held for the duration of the data exchange between two end users. Transmitted data is broken into fixed size packets like this:



When a virtual circuit is terminated, sessions assigned to that circuit terminate abnormally.

Each packet contains a network-defined header that identifies the virtual circuit, thereby identifying the sender and the receiver of the information. The packet-switched PDN controls the number of packets it accepts for transmission from the data terminal equipment; the data terminal equipment controls the number of packets it accepts from the PDN. These controls are by virtual circuit. They are implemented by ICAM in the data terminal equipment and by the PDN data circuit-terminating equipment.

ICAM can support many active virtual circuits at the same time to different remote data terminal equipments:



Two kinds of virtual circuits exist in packet-switched PDNs:

- Switched virtual circuits
- Permanent virtual circuits

Switched virtual circuits are established dynamically through call requests by ICAM when your program requests that a session be opened with another end user. Circuits are closed on request by your program.

Permanent virtual circuits are established by agreement with the PDN when you subscribe to the service. These provide fixed virtual circuits between two end users.

Up to 4095 virtual circuits can be specified to share a single link between ICAM and a packet-switched PDN. However, the number that can be used is limited by tariff and buffer constraints.

ICAM provides two types of network access controls for PDNs. The first allows you to specify a virtual circuit as input only, output only, or both input and output. That is, a virtual circuit can be used to handle only incoming calls, only outgoing calls, or both.

The second type of network access control is the closed user group. A closed user group allows you to form a private network within a packet-switched PDN. You can designate data terminal equipment with associated virtual circuits as belonging to a closed circuit group. This provides an additional level of security within the PDN.

The specifications for currently supported packet-switched PDNs are given in Table 3-3. In addition, the Universal Terminal Systems 4020 (UTS 4020) and 4040 (UTS 4040) Cluster Controller are supported for DATEX-P, TRANSPAC, and PSS.

**Table 3-3. ICAM Packet-Switched Public Data Network Support**

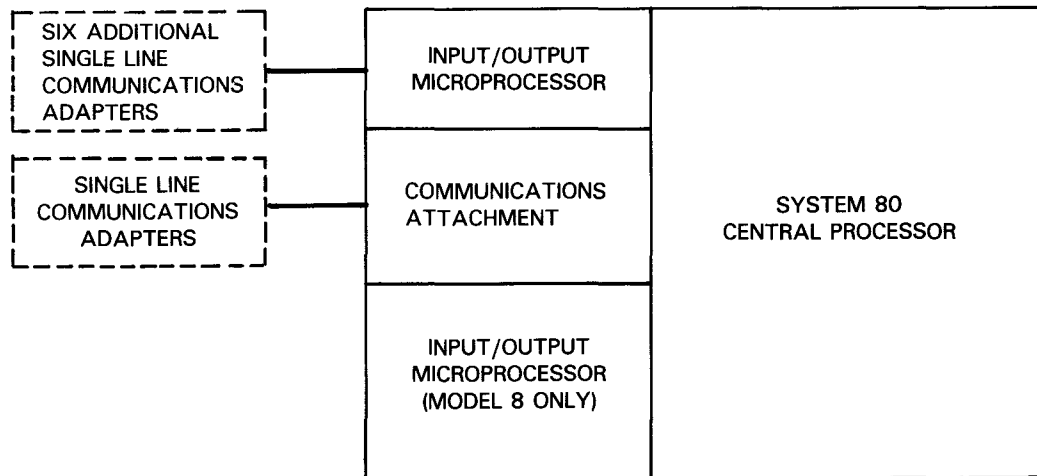
PDN Name	Physical Interface	Protocol	Country of Origin	ICAM Interface	SLCA Required
DATAPAC	RS-232C	X.25	Canada	STDMCP user, DDP, IMS	F2798-00
DATEX-P	RS-232C	X.25	Germany	STDMCP user, DDP, IMS	F2798-00
DDX-P	RS-232C	X.25	Japan	STDMCP user, DDP, IMS	F2798-00
PSS	RS-232C	X.25	United Kingdom	STDMCP user, DDP, IMS	F2798-00
TRANSPAC	RS-232C	X.25	France	STDMCP user, DDP, IMS	F2798-00
IBERPAC	RS-232C	X.25	Spain	STDMCP user, DDP, IMS	F2798-00

### 3.4. Single Line Communications Adapters

The basic System 80 communications system supports two single line communications adapters to coordinate data transfer, status, and commands on a per line basis. An optional input/output (I/O) microprocessor extends the capability of the System 80 communications system. It provides support for six additional single line communications adapters to support up to eight lines.

The System 80 models 8, 10, 15, and 20 support up to two I/O microprocessors, and each one can support up to 14 single line communications adapters. Thus these System 80 processors can support up to 28 communications lines.

Figure 3-7 shows the relationship between the I/O processor and the System 80 central processor.



**Figure 3-7. Input/Output Microprocessor Interface within System 80**

The input/output microprocessor is the interface between the single line communications adapter and central processor. All communications operations are handled by the input/output microprocessor for single line communications adapters in systems with more than the basic two.

Some of the functions performed by a single line communications adapter are:

- On input, it assembles bit-serial code from the communications line into characters for the processor.
- It disassembles characters from the processor into bit-serial for the communications line on output.
- It establishes character synchronization on synchronous lines.
- It activates and deactivates communications lines as instructed by ICAM.

Table 3-4 lists the different kinds of single line communications adapters available. The list is continually being expanded, so you should consult your Unisys representative if you don't find one you need in the table.

**Table 3-4. Single Line Communications Adapters**

Feature Number	Interface Specification	Transmission Rate (bits/s)	Half Duplex (HD) Full Duplex (FD)	Async/ Sync	Bit/ Byte	Protocol (See notes)	Microcode Name <sup>4</sup>
F2899-02	RS-232	9600 4800	HD FD	Sync	Byte	1	CMM1
F2788-03	MIL-STD-188	9600 4800	HD FD	Sync	Byte	1	CMM1
F2788-04	RS-232	9600 9600	HD FD	Sync	Byte	NTR	NTR1
F2788-05	MIL-STD-188	9600 9600	HD FD	Sync	Byte	NTR	NTR1
F2798-00	RS-232	19200 9600	HD FD	Sync	Bit	2	UDLC
F2799-00	RS-232	9600 4800	HD FD	Async	Byte	3	CMM2
F2799-01	MIL-STD-188	9600 4800	HD FD	Async	Byte	3	CMM2
F2986-02	CCITT V.35	56000	FD	Sync	Bit	5	BIT5
F2986-05	CCITT V.35	56000 56000	HD FD	Sync	Byte	NTR	NTR5
F2798-06	RS-232	19200	HD	Sync	Byte	UNISCOPE	UNI3A



### **3.5. DCP Channel**

System 80 models 8-20 support a high-speed selector channel interface to a DCP front-end processor.



# Section 4

## Line and Terminal Support

### 4.1. Remote Device Handlers

When we talk about the ICAM line and terminal support, we're really talking about what its remote device handlers do. They are the link between the software and the hardware. These routines:

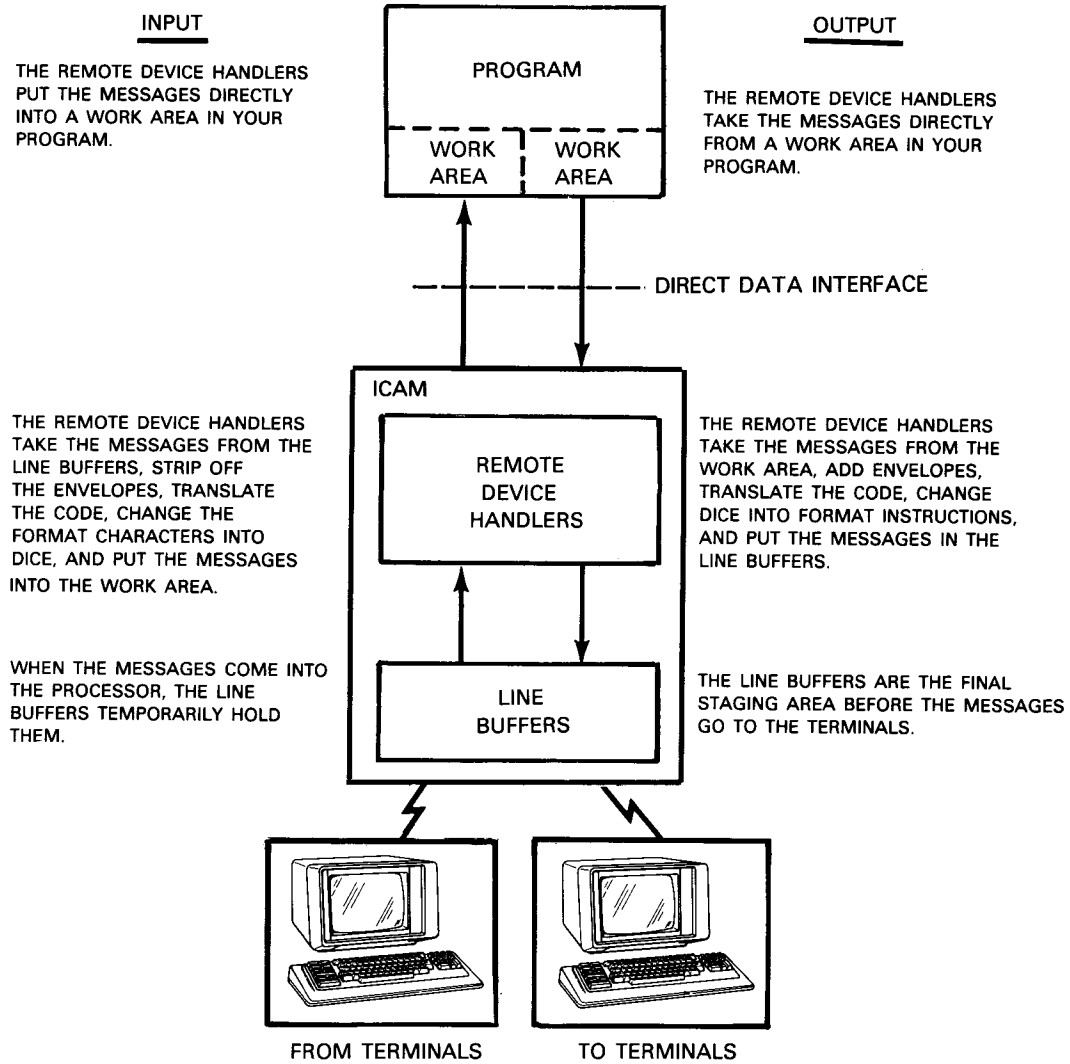
- Service line/terminal protocol
- Handle all input and output
- Translate the code of your messages from the EBCDIC code used in the computer to whatever line codes your terminals use; that is:

Output:    your program → EBCDIC → line code → terminal

Input:     your program ← EBCDIC ← line code ← terminal

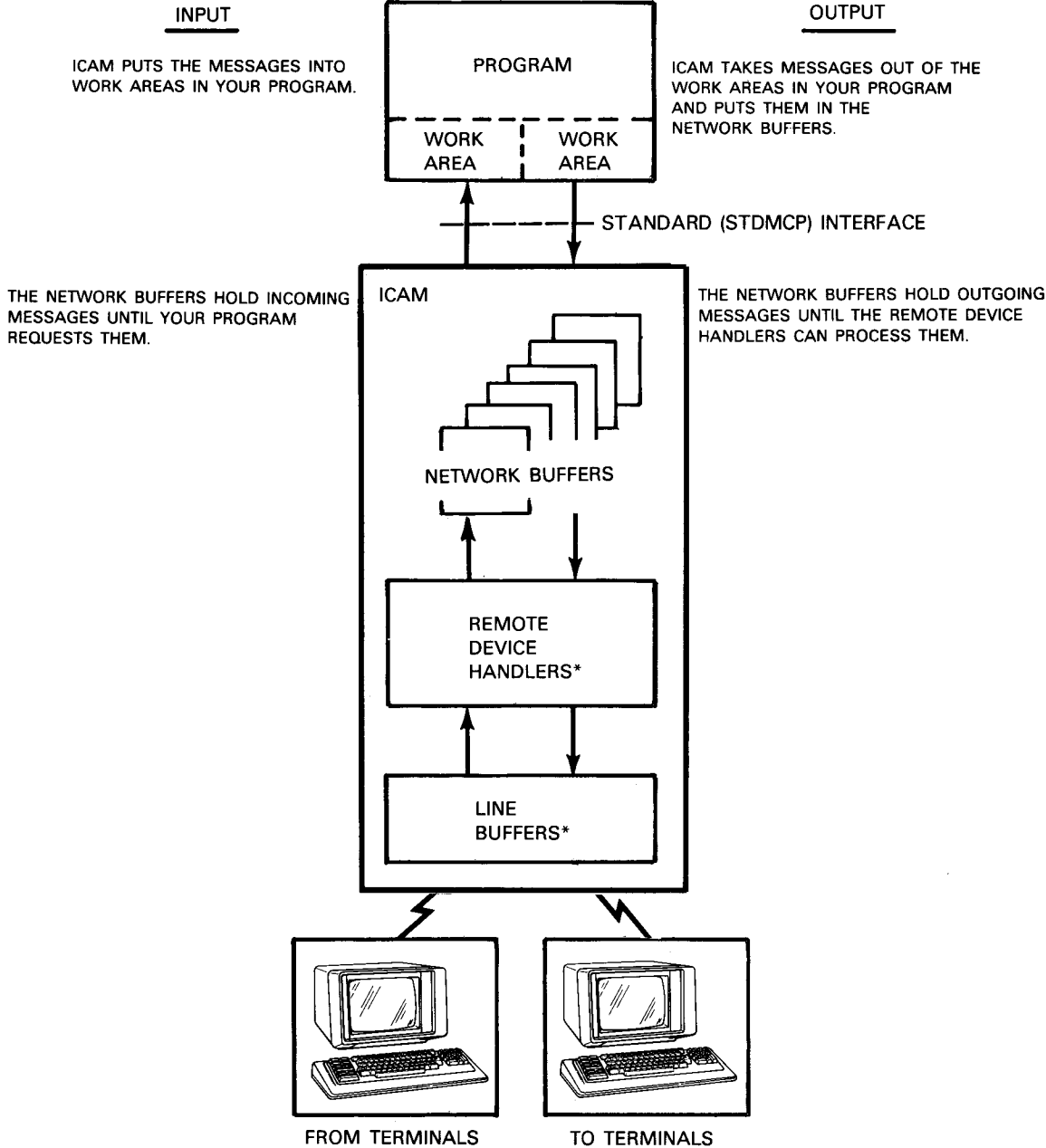
- Help your programs provide control information for formatting your messages through the device independent control expressions (DICE) and format edit
- Provide for error recovery occurring during input or output

The relationship of your program with the remote device handlers depends on the interface it uses. In the communications physical interface, the remote device handlers don't even exist, and your program must perform the handler functions. A level above, in the direct data interface, remote device handlers exist, but without any network buffers between them and your program. The relationship looks like this:



(No program, regardless of the interface it uses, works directly with the remote device handlers. In the direct data interface, all communication between a program and any routine in ICAM is through an ICAM routine called the direct data interface controller.)

In the standard interface, network buffers separate the remote device handlers and your programs.



\* The remote device handlers and line buffers work the same as in the direct data interface except that they move data into and out of network buffers instead of work areas in your program.

The network buffers literally protect your programs from the problems of input and output. If your program is using the direct data interface, it must initiate any required error recovery during input or output; ICAM initiates error recovery for standard interface users. Under the direct data interface, your program must specifically request the remote device handlers to read or write a message to the terminals; under the standard interface, ICAM handles all input and output. As we said before, the direct data interface gives your programs more control over the communications system, but the standard interface makes the job of writing programs much easier.

We've been talking about remote device handlers as if they're a standard set of routines. Actually, each one is a group of subroutines put together to support the characteristics of the terminals on a single communications line. A remote device handler for a line with a DCT 1000 terminal attached differs substantially from one for a line with an IBM 2780 terminal. Two slightly different sets of code support a UNISCOPE terminal capable of displaying 960 characters and one capable of displaying 1024 characters. Because each terminal needs different support from the remote device handlers, each line must have terminals with similar characteristics on it. You can mix DCT 1000, UNISCOPE, UTS 400, and UTS 4000 terminals on the same line, but you can't mix UNISCOPE and teletypewriter terminals on the same communications line. When you describe your communications system for ICAM generation, most of your effort probably will go towards describing the characteristics of your terminals so the remote device handlers can support them. If you use terminals that ICAM doesn't support, you must write your own remote device handler to support them. For more information on this, see the *ICAM Interfacing a Remote Device Handler Programmer Reference* (UP-8424).

## 4.2. Line Connections

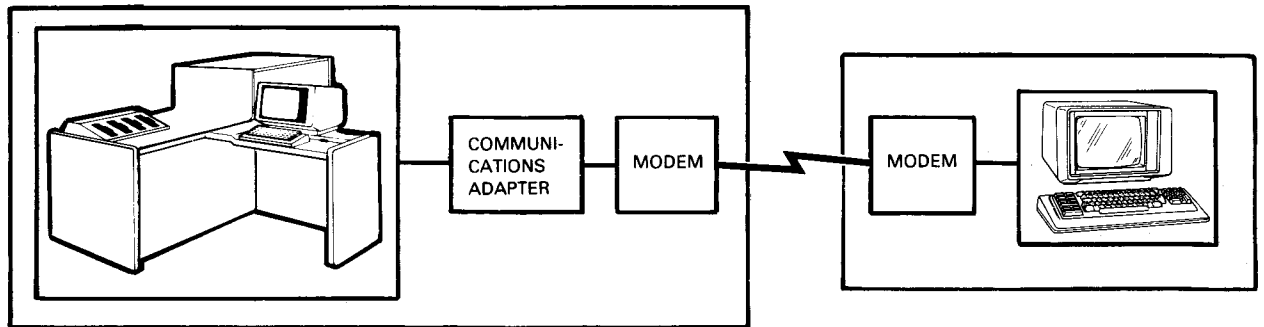
The remote device handlers provide the control needed to initiate the lines for sending messages. All the other characteristics of lines - like 2-way alternate/2-way simultaneous, asynchronous/synchronous, and line speed - are hardware features controlled by the single line communications adapter.

Connecting the lines is a 2-step process.

First, ICAM must be asked to support the line by having the line requested. With dedicated networks, your program can request that all lines be activated automatically when it requests the network, or it can request each line individually once the ICAM network is loaded. With global networks, the system operator requests lines by commands typed in from the system console once both ICAM and GUST are loaded. Once a line is requested, ICAM prepares the single line communications adapter to support each line by loading it with the commands and data it needs.

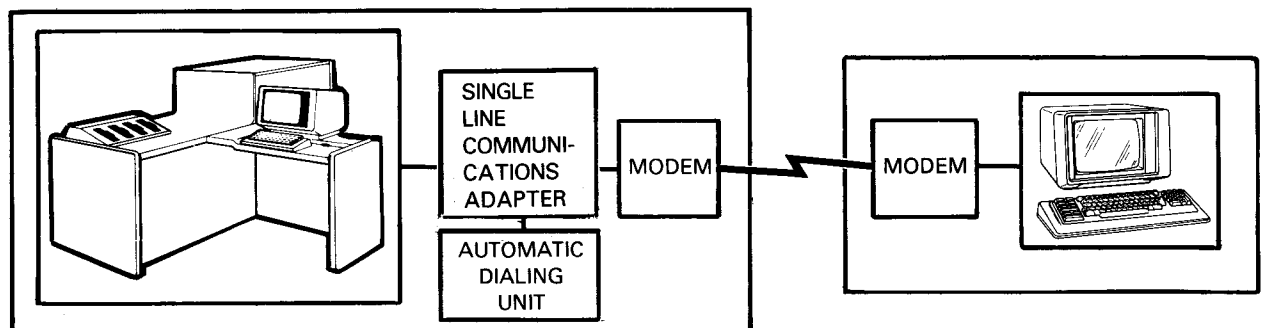
The second step of the line connection procedure is establishing the circuit. This is done in one of four ways, depending on the type of line:

1. *Dedicated lines* - Dedicated lines are the simplest of the four because there's no dialing required.

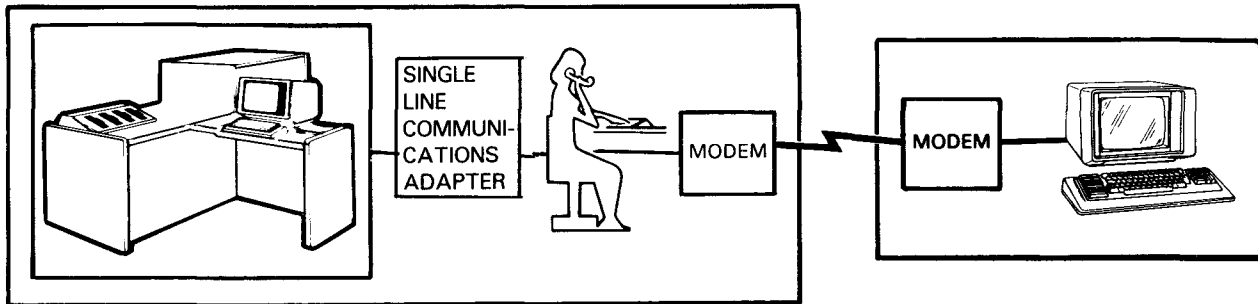


When the line request is made, the communications hardware establishes the circuit. If the circuit can't be established (probably because a modem is turned off), ICAM notifies both your program and the system operator.

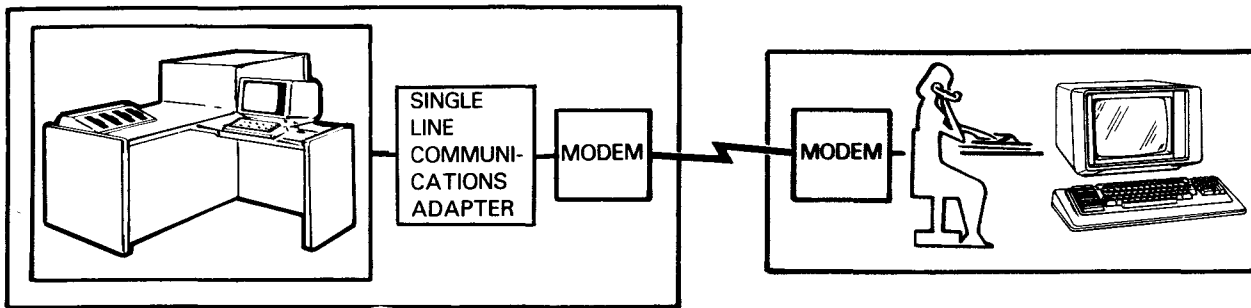
2. *Automatic dialing lines* - When the line request is made, the single line communications adapter establishes the circuit using the automatic dialing unit. If the circuit is busy or there's no answer, ICAM notifies your program and the system operator that the circuit wasn't established.



3. *Manual dialing* - With manual dialing, the line request generates an operator message telling the system operator to make the call. After trying, the operator tells ICAM from the system console whether or not the call was successful. If it wasn't, ICAM does *not* notify your program. The remote device handler doesn't begin polling the terminals on the line until the system operator notifies ICAM that the circuit is established.



4. *Unattended answering* - Unattended answering differs from other methods of line connection because it's the only one where the terminal operator establishes the circuit by dialing the host computer. As with the other methods, however, the line must be requested before the terminal operator makes the call. Except for direct data interface programs, ICAM doesn't notify your program when the circuit is established. Your program must periodically check input queues to see whether any messages have been received or issue a deferred request (e.g., a deferred GETCP) and yield control to ICAM. In this case, ICAM gives your program control at the address you specify in the deferred request when a message is received.



Once the circuit is established, the remote device handler begins polling the terminals (provided they're pollable). Any time all the terminals are marked down, ICAM logically marks the line down. The only effect of this is that the normal polling sequence stops, and ICAM begins to perform a slow poll of the terminals. (As described in 4.3, this happens with any terminal that's marked down.) Once any terminal responds to a poll, ICAM marks the line up.

If the line has uncontrolled - nonpollable - terminals, the remote device handlers put up a read for input once the circuit is established. Because there are no polls, the line cannot be logically marked down.



The line connection is broken by one of three events:

1. *A line release is issued.* - In dedicated networks, your program releases the lines. In global networks, the system operator requests line releases via commands typed in at the system console. When the line release is made, the single line communications adapter for that line breaks the circuit and ICAM marks the line down. In global networks, ICAM notifies all programs linked to it that the line is down. This is the proper method of breaking a line connection.
2. *Someone hangs up the phone on a switched line.* - Because this breaks the circuit, ICAM physically marks the line down. The status code for this is different from the logical line-down status code caused by terminals not answering polls. To recover in a dedicated network, your program must logically release the line and then rerequest it. This is automatic in global networks.
3. *A hardware error occurs.* - Because this breaks the circuit, ICAM physically marks the line down, as it would if someone hangs up the telephone. (ICAM can't distinguish between the two.) Recovery procedures are the same.

You should avoid breaking the line connections by hanging up the phone because ICAM cannot distinguish between this and a hardware error. If the phone is hung up, the connection isn't needed and shouldn't be reconnected. If a hardware error occurs, you should try to reconnect the line if possible. To avoid the entire problem of whether or not a line should be reconnected, you can use the following procedures:

- In dedicated networks

IF

your program wants to release the line,

your terminal operator wants to release the line,

your system operator wants to release a line,

a physical line down condition occurs and you follow these procedures,

THEN

it releases the line.

the operator sends a message to your program, instructing it to release the line.

the operator can use any terminal connected to ICAM to send your program a message, requesting it to release the *line*.

your program knows a hardware error has happened. It logically releases the line and rerequests it. If the line isn't reconnected, your program sends an error message to the system console.

- In global networks

### IF

your program wants to release a line,

your terminal operator wants to release a line,

your system operator wants to release a line,

a physical line-down condition occurs,

### THEN

it sends a message to the system console, telling the system operator to tell GUST to release the line.

the operator sends a message to your program instructing it to release the line. Your program then sends a message to the system console, telling the system operator to tell GUST to release the line.

the operator sends a message to GUST, telling it to release the line.

your program knows a hardware error has occurred but takes no action. ICAM has tried to reconnect the line and failed, and has informed the system operator of the hardware error.

## 4.3. Terminal Polling

Polling is explained in "Line Control" in Section 3, but we'll summarize what it does here. Polling is a method of controlling transmission on a line supporting one or more controlled terminals so that (1) only one device transmits at any one time and (2) the host computer is ready to receive the messages the terminals send.

The remote device handlers control polling. They poll the terminals, asking them:

- If they have messages to send
- The status of their input/output devices (up, down, busy)
- To retransmit messages garbled during transmission

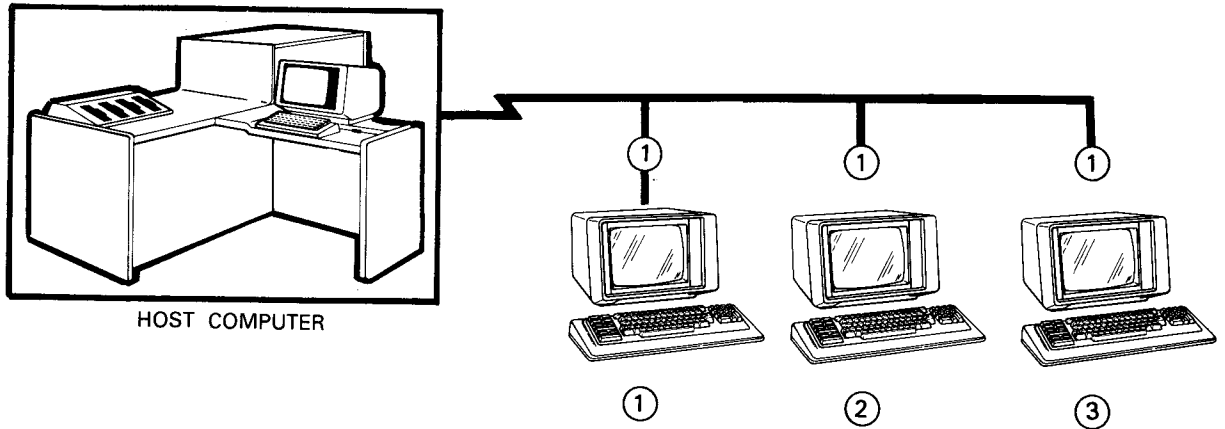
The remote device handlers and terminals acknowledge the successful receipt of messages.

The following discusses the three factors controlling polling:

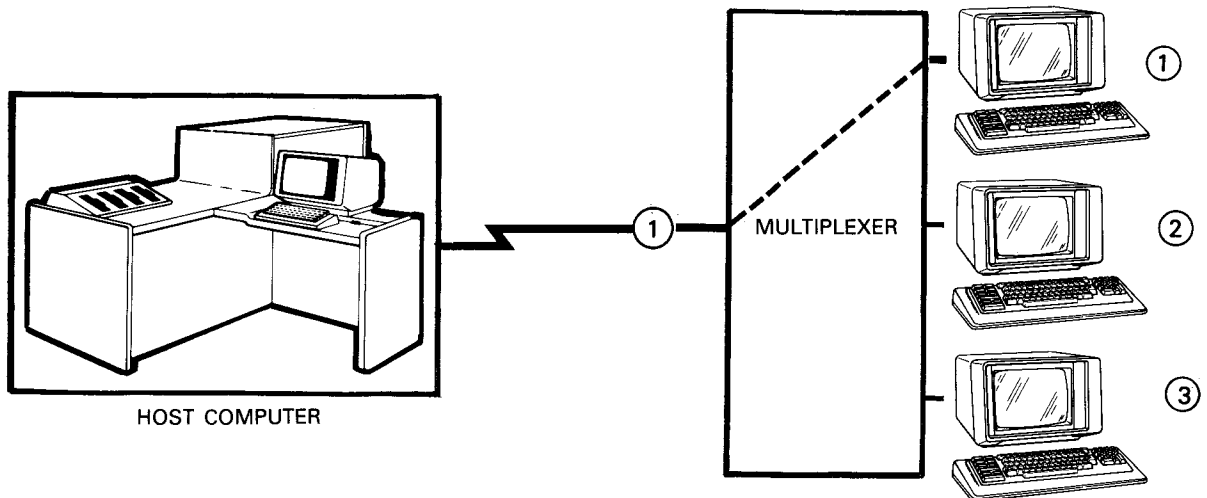
1. How you group the terminals into polling groups
2. Polling interval
3. The polling algorithms of the remote device handlers

### 4.3.1. Polling Groups

When a line has more than one terminal on it, there are three ways to arrange the terminals. One is to have multiple drops off the line, which is like a party line. Each terminal hears every message and poll, but responds only to those addressed to it.

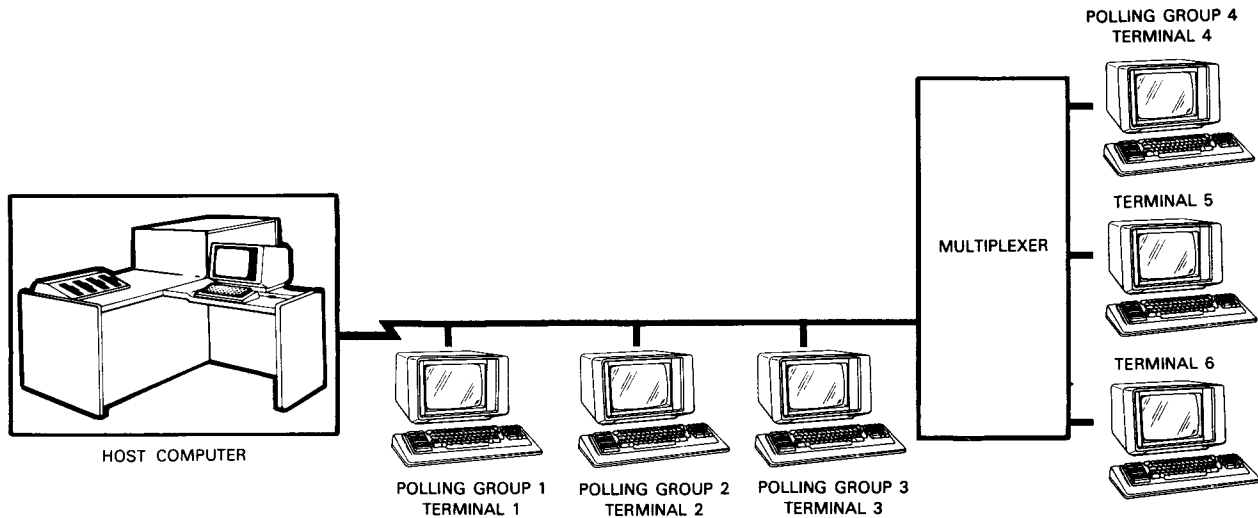


Another way is to have a multiplexer connect several terminals to a single drop on a line. The multiplexer passes only those messages addressed to a particular terminal to that terminal.



And you can combine the two methods:

Each drop on a line, whether it's to a single terminal or several terminals connected by a multiplexer, forms a separate polling group. For example:

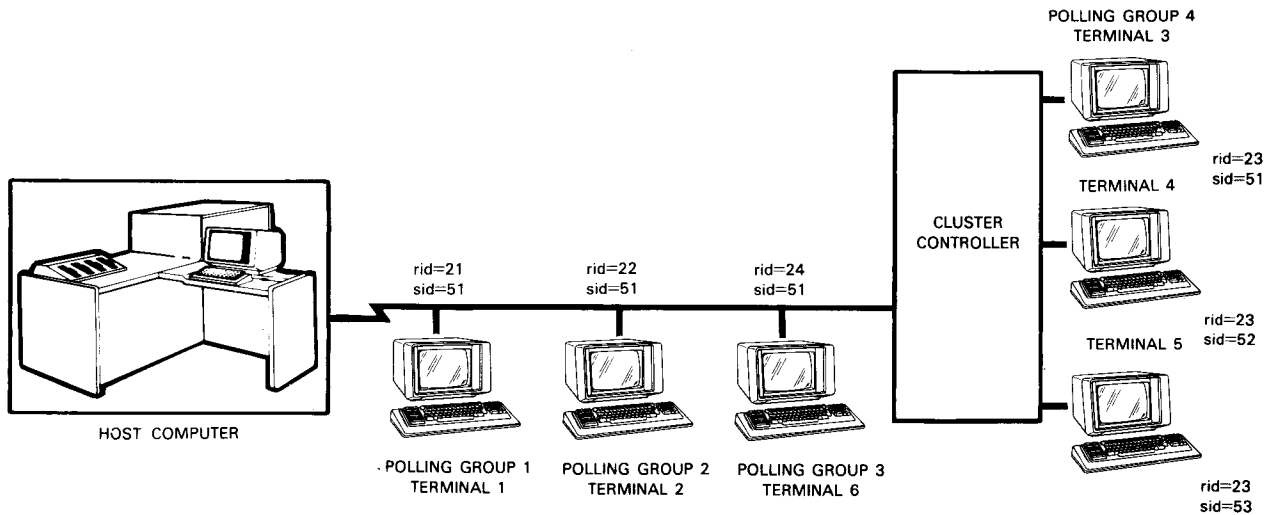


A drop could contain multiple terminals without a multiplexer. In this situation, there can be multiple poll groups.

The remote device handler polls each polling group separately, a subject further discussed in the next two subsections, 4.3.2 and 4.3.3.

In Section 3, we talked about the terminal addresses. There, we said the address consists of a remote identifier (rid), a station identifier (sid), and a device identifier (did). The remote identifier corresponds to a polling group, and the station identifier corresponds to individual terminals in the polling group. (The device identifier corresponds to individual devices connected to a terminal and is explained in 4.4.) The terminals in a polling group share the same remote identifier, and each terminal in the polling group has a unique station identifier. Look in your terminal manuals for the permissible addresses for your terminals.

If the terminals in the previous illustration are UTS 4000 terminals, their addresses could be:



Each polling group has a different remote identifier. Terminals 3, 4, and 5 share the same remote identifier because they're in the same polling group. Terminals 1, 2, 3, and 6 can have the same station identifier because they're in different polling groups. Terminals 3, 4, and 5 must have different station identifiers because they're in the same polling group.

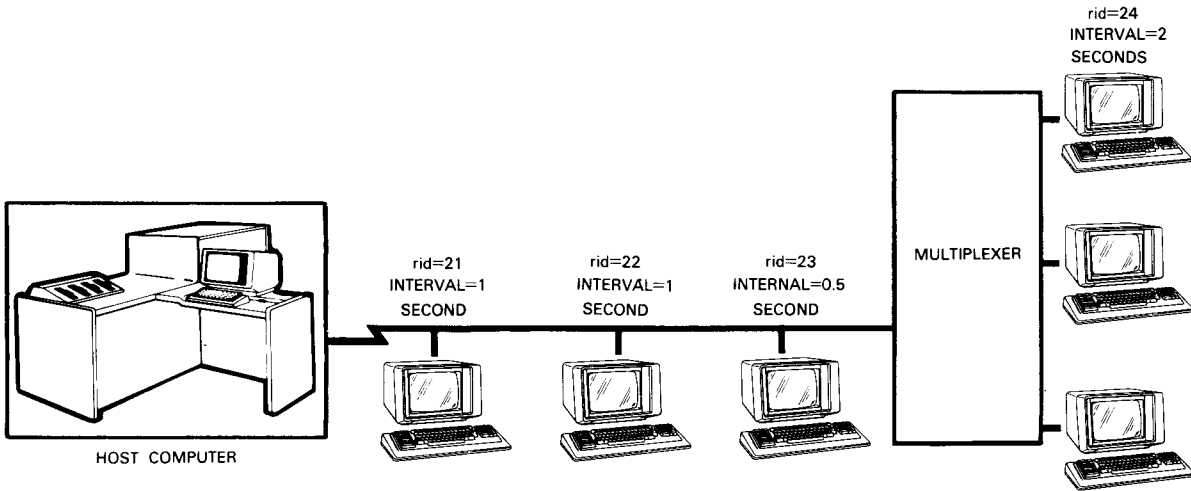
Whenever ICAM sends a message on a line, all the terminals on the line receive the address portion of the message. If this address is not recognized, the terminal ignores it. It's as if the terminals are part of a conference call. Each hears everything that is said but acts only if specifically addressed.

### 4.3.2. Polling Interval

The polling interval is the time between polling sequences within a polling group. (For examples of polling sequences, see the next subsection, 4.3.3.) When you define the network, you specify a polling interval for each polling group that may be 1/10 second, 1/5 second, 1/2 second, 1 second, or any integer value up to 255 seconds. Once the line is connected, the remote device handler polls each polling group at the interval specified. (The DCT 500 terminal is an exception - see "Polling with Unbuffered Interactive Terminals" later in this section.)

Because polling groups can have different polling intervals, the remote device handlers can poll some more often than others. You can give polling groups different priorities. Groups polled more often can send and receive messages more often.

If you have the following intervals for your polling groups:



then the remote device handler polls them in the following order:

<u>Seconds</u>	<u>Polling Groups Polled</u>
0	none
0.5	rid=23
1	rid=21, rid=22, rid=23
1.5	rid=23
2	rid=21, rid=22, rid=23, rid=24
2.5	rid=23
3	rid=21, rid=22, rid=23
3.5	rid=23
4	rid=21, rid=22, rid=23, rid=24

This polling scheme is the ideal; a number of factors affect the actual polling intervals. Of these, the polling algorithms discussed in the next subsection (4.3.3) are the most important. The rest are discussed here.

When the remote device handler sends out a poll, it expects a response of some kind, even if the polling group terminals don't have any messages to send. It waits a specific amount of time (you determine the interval at network definition) for a response. If none comes, *time-out* occurs. The remote device handler retries the poll several times (again, you specify how many times) at the normal polling interval. If the terminals still don't respond, either the terminals were turned off, they suffered hardware

failures, or the rid/sid/did is incorrect. In any case, further polling at the normal interval wastes processor time. The remote device handler begins slow polling the poll group every minute to minute-and-a-half (depending on the type of terminal) in case one of the terminals becomes capable of sending a message. When one of them does, the remote device handler resumes polling at the normal polling interval. (If the remote device handler is slow polling every poll group on a line, ICAM logically marks the line down. When any terminal answers a poll, ICAM marks the line up.) If your terminals or remote workstations are connected to a UNISCOPE or remote workstation line, you can specify the slow polling interval when you define the polling group.

Another factor influencing the polling rate is the number of network buffers available. If all the network buffers are full, incoming messages could be lost because there isn't any place to hold them. The remote device handlers stop polling to solicit input (and/or inhibit output from your program) when the number of buffers containing messages reaches a threshold value (which you specify at network definition). After a large number of the network buffers are emptied, the remote device handlers resume normal polling. (See the network buffer description, 5.4, for more on the threshold value.)

If you use the standard interface, you can affect the polling interval with the *inhibit* feature, which allows your programs to carry on a one-to-one conversation with a terminal. After an inhibited terminal sends in a message, the remote device handler stops polling it until your program responds to the message. Normal polling then resumes until the next message from the inhibited terminal, when polling again ceases until the program responds. You specify inhibit on a terminal basis - that is, you inhibit the polling of one terminal on a line without affecting the polling of the other terminals on the line. The inhibited terminal, however, must be the only terminal in its poll group. If there are other terminals in the poll group, the remote device handler continues to poll the group even if the inhibited terminal is waiting for a response. When the group is polled, the inhibited terminal can send a message before it receives a response.

The last factor affecting the polling interval is the nature of the direct data interface. You'll remember that, if you use this interface, no messages can come from a terminal until your program requests input. Another way of putting it is that the remote device handlers don't begin polling a line until your program requests input. When your program requests input, the remote device handler polls normally until a terminal sends a message. At this point, polling stops until your program requests more input. If there is more than one polling group on a line, it's possible that some groups are never polled because the first groups polled always have input. You should avoid more than one poll group per line if you use the direct data interface.

### 4.3.3. Polling Algorithms

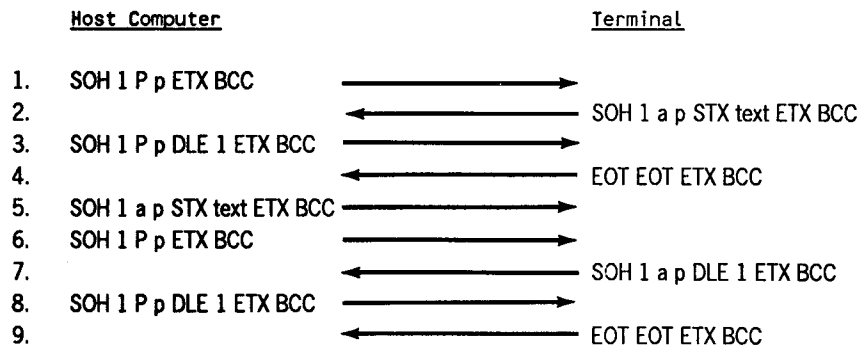
A polling algorithm is the method used by a remote device handler to control transmission on a line. The following two subsections describe the polling algorithms for ICAM-supported interactive terminals. Because each batch terminal has a unique polling algorithm, this manual doesn't describe their algorithms. You should check the terminal manuals for their algorithms.

### Polling with Buffered Interactive Terminals

The protocol used with the buffered interactive terminals is complicated. The UTS 4000 protocol, for instance, has 27 rules governing its interaction with the host computer, and the other protocols are similarly complex. But each protocol follows five basic rules:

1. A terminal must respond to an error-free poll.
2. The host computer must acknowledge messages a terminal sends in response to a poll, except for the no-traffic response.
3. A terminal will not send two consecutive messages containing text. It does not respond with a message to a poll that includes the acknowledgment to the previous message.
4. Upon sending to a terminal, the host computer must poll the terminal to verify proper receipt of the message.
5. Unless polling is inhibited for some reason, the host computer polls each polling group at the interval specified when you define your ICAM network.

For the first two examples, we'll look at polling with a polling group containing a single terminal.



This example shows the simplest kind of polling for buffered interactive terminals: The polling group contains only one terminal, and no errors occurred during transmission by either party. The following explains each poll and response:

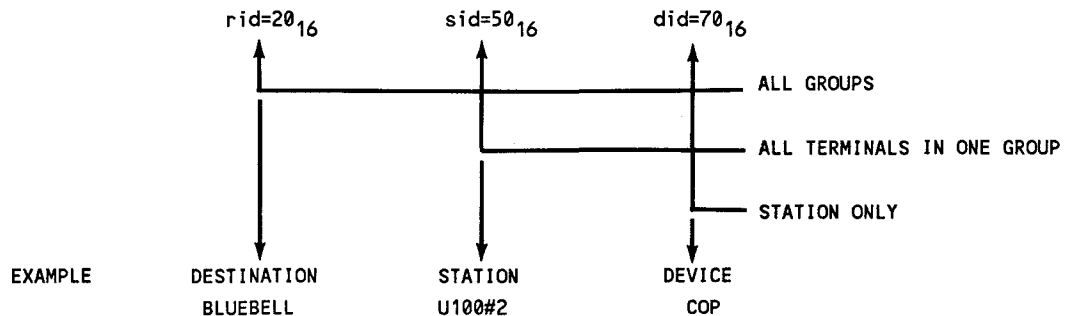
1. The host computer sends a general poll asking for input from the polling group with the rid of 1.
2. The single terminal in the group replies with a message containing text.
3. The host computer acknowledges the successful receipt of the message and asks for more input.



4. The terminal replies that it has no messages waiting (the no-traffic response). It must give this reply regardless of whether it has a message waiting or not. You'll remember that one of the rules of protocol says that no terminal can send two consecutive messages containing text.
5. The host computer now sends a message containing text to the terminal.
6. The host computer follows up its text message with a poll requesting input.
7. The terminal has no messages to send, but it acknowledges the successful reception of the text message.
8. The host computer acknowledges the reception of the previous response and asks for more input.
9. The terminal sends a no-traffic response.

You may have noticed that the addresses in most of the polls from the host computer don't match the addresses in the responses from the terminal. These are general polls; the remote identifier matches that of the polling group, but the station identifier is general. Any terminal in the polling group can respond to a general poll. In polling groups with just one terminal, only that terminal responds. In polling groups with several terminals, however, any one can respond. (The multiplexer controls which terminal responds to a poll.) This way, the remote device handler doesn't have to poll each terminal in a polling group. It continues to poll until it gets a no-traffic response, meaning that none of the terminals in the group has a message ready to send.

## SUMMARY OF gids:



**Note:** The rid, sid, did address trio may make use of a general identifier (gid). This gid may be used in place of the rid, sid, or did. For the rid, a gid = 20<sub>16</sub> is recognized by all remote stations as its rid. For the sid, a gid = 50<sub>16</sub> is recognized by all terminals at an installation. For the did, a gid = 70<sub>16</sub> merely addresses the station without selecting an auxiliary device.

The one case in the previous example where the host computer uses a specific address is when it has output to go to the terminal. Whenever the host computer has output, it uses the specific address of the terminal that's to receive the message.

The nine steps given here are a single polling sequence for one polling group. The majority of the steps just give the terminal permission to send messages and acknowledge the receipt of messages. If you have a number of polling groups and your remote device handlers reside in the host computer, then a substantial amount of computer time is spent polling. This doesn't include the time it takes the single line communications adapter to turn the line around after each poll or response. Given these considerations, you can make the most efficient use of your communications system by having as few polling groups per line as possible, and by specifying the highest polling interval you can without backing messages up at your terminals.

If your remote device handlers are standard handlers provided by Unisys, these suggestions aren't as important. But if you are providing your own unique handlers, remember that the single line communications adapter has a limited amount of processing power, and it can become saturated. When it does, response time goes up.

Another problem that will become clearer later is that it's possible to lock up your system just by polling. Let's say you have five polling groups on a line, and each polling session takes 2 seconds. The fastest polling interval possible under these circumstances is 10 seconds, regardless of the polling interval you specified at network definition. Response time in these circumstances can easily be close to half a minute or more, as the following shows:

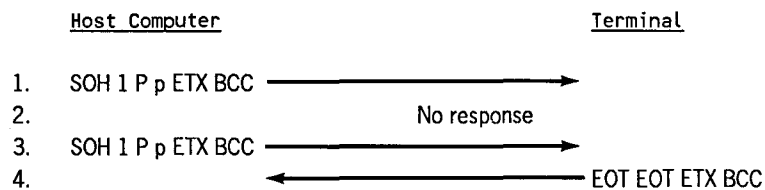
Time between the instant the terminal operator hit the transmit key and the instant the terminal was polled:	10.0 seconds
Time message sits in a network buffer waiting to be processed by the program:	2.0 seconds
Processing time:	0.5 second
Time message sits in a network buffer waiting to be sent:	11.0 seconds
Total:	23.5 seconds

Normally, polling works so fast that you never notice waiting time. If your response time is slow, however, you might check whether you're overloading the communications system with polling. To do this, check the network buffer statistics (5.4.) to see whether the network buffers are filling faster than they're emptying. If not, the problem may be the number of polling groups. To test this, reduce the number of polling groups you have on a line by turning off the terminals in one or more groups. Then use the remaining polling groups to send in the same number of messages as all the groups normally do. By using this strategy, the message volume remains the same, but the polling volume is reduced. If the response time improves, examine ways of reducing the number of polling groups.

The previous example showed an errorless polling sequence. When an error occurs, the poll is retried several times. (You specify the number of retries at network definition.) If the problem disappears, normal polling resumes. But, if the problem continues, the remote device handler assumes the terminal was turned off or suffered a hardware failure. The terminal is marked down and receives no output until it sends input.

When all terminals in a polling group are down, the polling interval becomes a minute to a minute-and-a-half. Once any terminal of the polling group responds to a poll, normal polling resumes for the entire group. Output to terminals still down, however, is not sent until they send input.

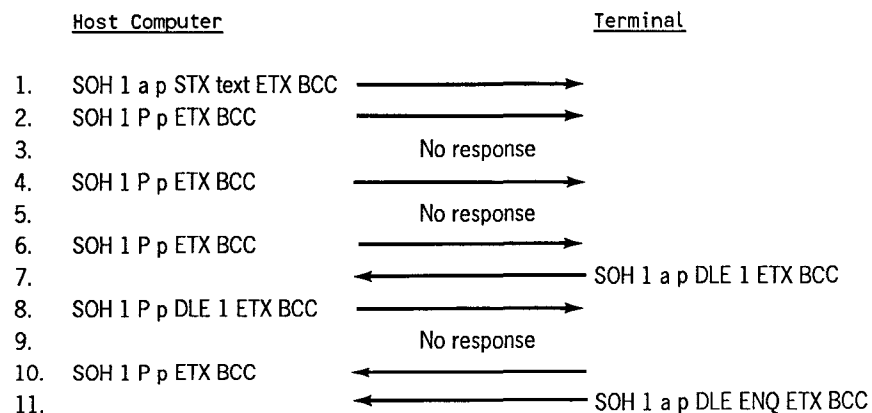
The following polling sequences show error recovery.

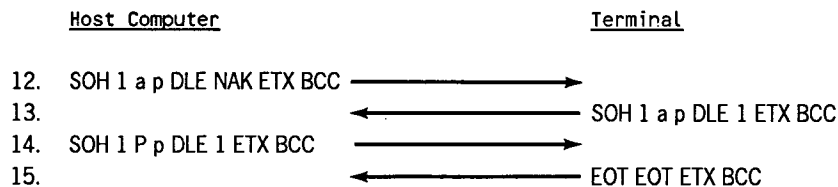


Explanation of the polling sequence:

1. The host computer sends a general poll requesting input.
2. The lack of response means that an error occurred during the transmission of either the poll or the response. (The remote device handlers treat any transmission error, even if it's only a parity error, as a no-response condition.)
3. The host computer retries the poll.
4. The reception of the no-traffic message from the terminal means error recovery was successful.

This polling sequence shows a more complex error recovery:

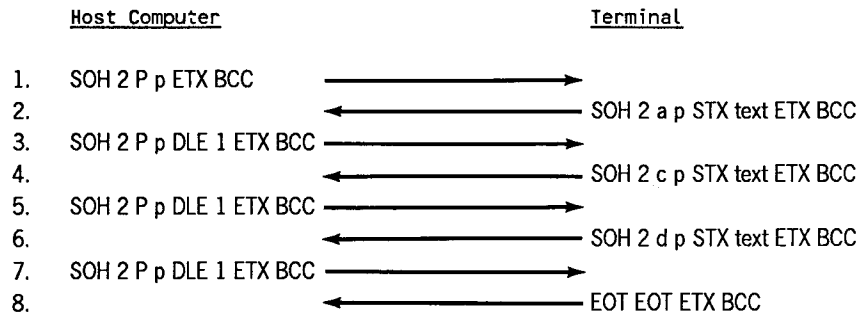
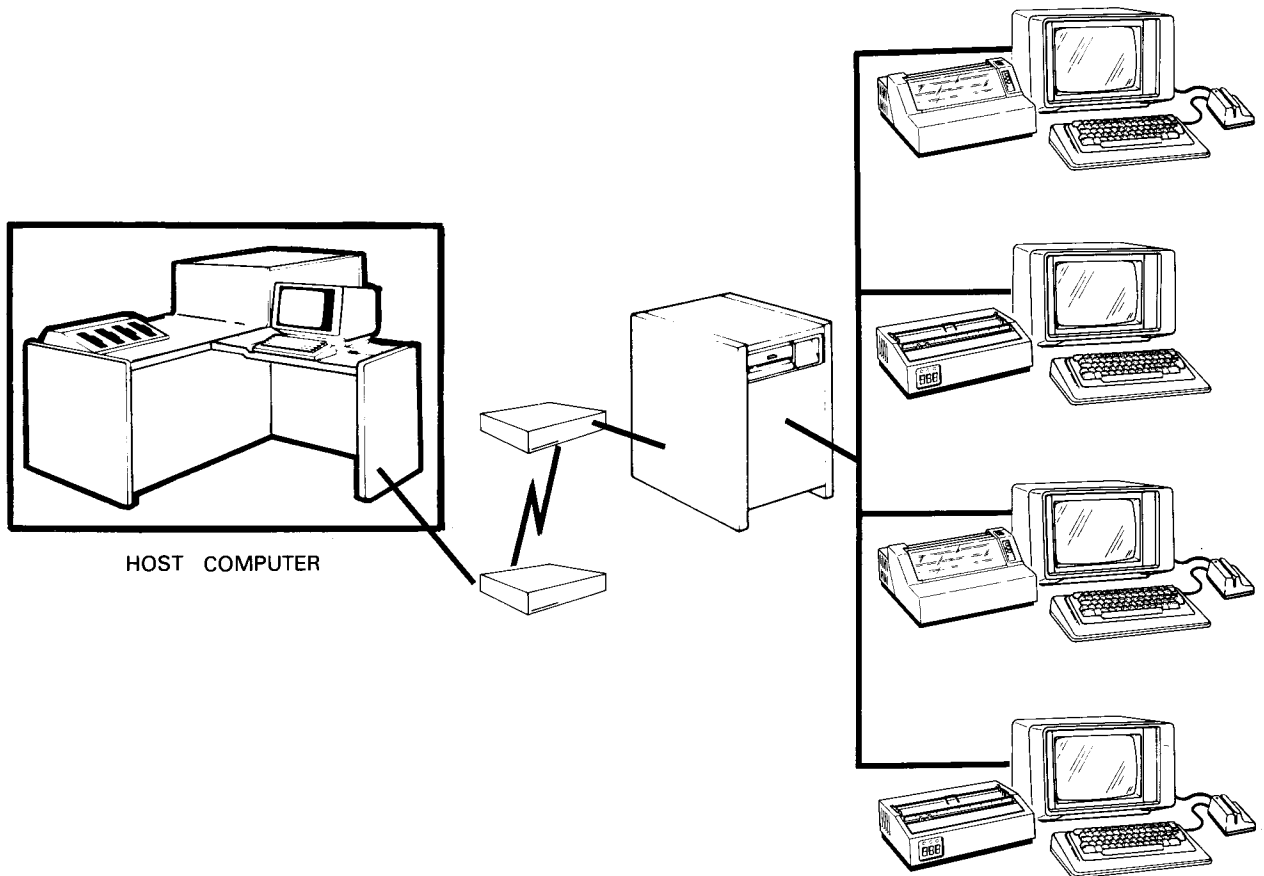




Explanation of the polling sequence:

1. Host computer sends a message to the terminal.
2. It then sends a general poll requesting input from the terminal. At the least, it expects an acknowledgment that the message it sent was successfully received.
3. The host computer doesn't get a response, meaning that either the poll or the response got garbled in transmission.
4. The host computer repeats the general poll.
5. There is still no response.
6. The host computer retries the general poll.
7. The terminal responds with an acknowledgment of the message sent at 1.
8. The host computer sends an acknowledgment as part of a poll for input. It expects either a message or a no-traffic response.
9. Nothing comes in from the terminal.
10. The host computer retries the poll from 8 without the acknowledgment.
11. The terminal responds with a request for an acknowledgment for its acknowledgment in message 7. It never got the host computer acknowledgment from message 8.
12. The host computer asks for an acknowledgment of its poll.
13. The terminal sends an acknowledgment.
14. The host computer acknowledges the acknowledgment and asks for input.
15. The terminal sends a no-traffic response, ending this polling sequence.

The polling sequence changes when a polling group contains more than one terminal. Now any terminal can respond to a general poll. Because the host processor acknowledges a terminal response with a general poll, polling continues as long as any terminal has input. The following example is based on the polling group shown:



**Explanation of the polling sequence:**

1. General poll requesting input from the polling group with the rid of 2.
2. The terminal with rid=2, sid=a responds with text.
3. The host computer acknowledges the reception of the message and requests further input.
4. The terminal with rid=2, sid=c responds with text.

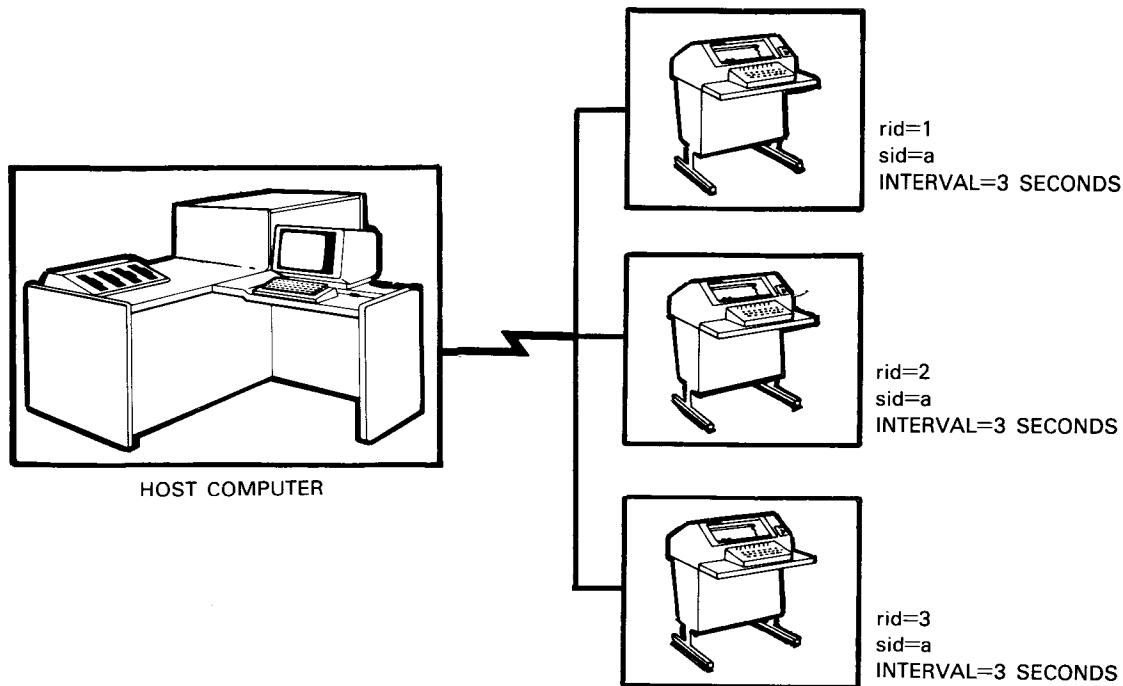
5. The host computer acknowledges the reception of the message and requests further input.
6. The terminal with rid=2, sid=d responds with text.
7. The host computer acknowledges the reception of the message and requests further input.
8. One of the terminals responds with a no-traffic message.

In this example, the total polling sequence is fairly short. If one or more terminals are dumping files from mass storage devices, however, the polling sequence continues until all the files are sent. The other polling groups are locked out because the computer wouldn't be polling them. You should take care that this situation doesn't occur.

### Polling with Unbuffered Interactive Terminals

ICAM supports the DCT 500 as an interactive terminal. Its lack of buffers forces the operator to wait until ICAM polls the terminal before entering a message. When polled, the terminal CLEAR TO SEND light goes on, and the operator must begin entering the message within a few seconds or wait until the next poll.

With the DCT 500, the polling interval gives the time after a poll that the remote device handler waits for input before polling another terminal. The polling interval is not, as it is with buffered terminals, the time between polls. Let's say you have this terminal arrangement:



When the remote device handler polls the first terminal  $rid=1$ ,  $sid=a$ , the terminal operator has 3 seconds to enter the first character of the message. Once the remote device handler receives that character, it accepts more characters until it receives an end-of-text character or until 8 minutes goes by with fewer than seven characters entered (that is, the terminal times out). If the terminal times out, the remote device handler treats what was received up to that point as the complete message.

The remote device handler polls the next terminal after the first terminal times out or finishes sending a message. The minimum time between successive polls of the same terminal is the number of terminals on the line times the polling interval. For the line shown, the time is  $3 \times 3 = 9$  seconds. If two terminals send messages, one taking 30 seconds to enter and the other 50 seconds, the time between polls of the first terminal is  $30 + 50 + 3 = 83$  seconds. (3 is the polling interval of the terminal without a message.)

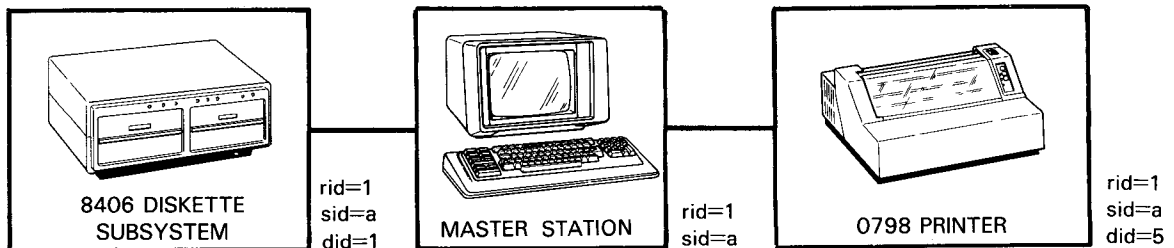
Two of the concerns of buffered interactive polling don't exist with DCT 500 polling:

1. *There is no error recovery.* - If a message is garbled in transmission, there's no back-up copy in the terminal's buffer. The remote device handler discards way messages with errors without notifying the operator. Direct data interface programs are notified of the error, but standard interface programs are not.
2. *DCT 500 terminals cannot be multiplexed.* - Each polling group contains just one terminal.

## 4.4. Input and Output Devices

All ICAM-supported terminals have primary input and output devices. On interactive terminals, the primary input device is always a keyboard; the primary output device is either a printer or a CRT screen (video screen). On batch terminals, the primary input device is typically a card reader but can be a diskette drive; the primary output device is a printer. In addition to the primary devices, most terminals also support one or more auxiliary I/O devices, such as tape cassette systems, diskette drives, printers, card readers/punches, and paper tape readers/punches.

A terminal address, you'll remember, consists of three characters: a remote identifier ( $rid$ ), a station identifier ( $sid$ ), and a device identifier ( $did$ ). The device identifier identifies a particular I/O device. The following example shows how it works:



The primary input and output devices (the keyboard and CRT) don't have their own device identifiers. Just as there are general remote identifiers and general station identifiers, there are general device identifiers. When a general device identifier is used, the message either comes from or goes to a primary I/O device. Auxiliary devices have their own device identifiers. The illustration shows the 8406 diskette subsystem with a device identifier of 1. What it doesn't show is that the diskette subsystem actually has device identifiers 1 through 4. A diskette subsystem holds two diskettes, each of which has separate read and separate write heads. In effect, the system has four I/O devices, and each gets a separate device identifier. The 0798 printer is simpler because it has only one device and gets a single device identifier, 5.

The following paragraphs summarize the functions ICAM supports with each input and output device.

- *Printers* - ICAM supports printers through the device independent control expressions described in 4.5.
- *CRTs (video screens)* - ICAM supports CRTs through the device independent control expressions described in 4.5.
- *Keyboards* - In addition to passing messages entered through a terminal keyboard to your program, ICAM informs your program if a function key was used to generate the message. The function keys are special keys on the UNISCOPE, UTS 400, and UTS 4000 terminals that generate 1-character messages. These messages have no inherent meaning; different programs interpret them differently. One program might interpret a message generated by function key 1 to mean, "Display complete customer file on the screen"- and another program might interpret it to mean, "The following messages deal with inventory."
- *Tape cassette systems* - You can send the following commands to a tape cassette system:
  - Backspace one block.
  - Search tape cassette for the key given in the message text.
  - Report current address from the tape cassette.
  - Read a block.
  - Write a block.
- *Diskette systems* - ICAM support for the diskette system is similar to that provided for the tape cassette system.
- *Paper tape readers/punches* - Your program can either read or write a block of data from the paper tape reader/punch.
- *Card readers/punches* - Your program can request that a card either be read or punched.



When a message comes in from any terminal device other than the primary input device, ICAM reports this to your program and gives it the device identifier of the device.

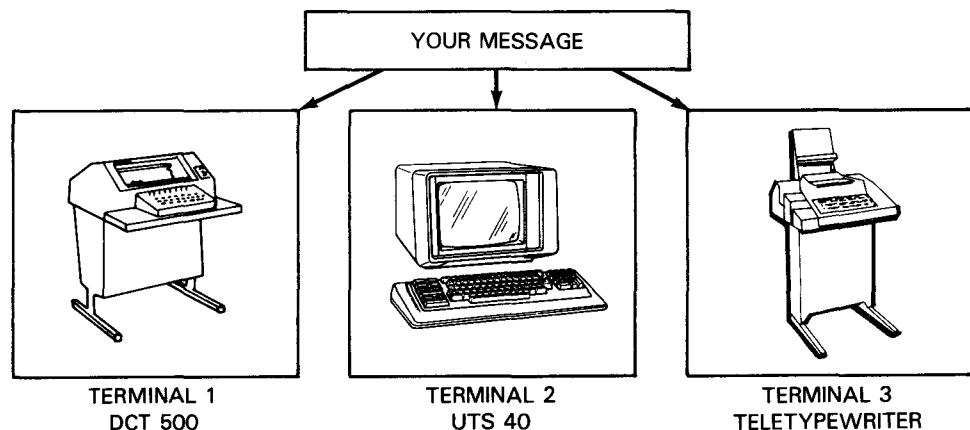
The functions described here for each input and output device are available with any terminal supporting the device. For information on how to use them, see the ICAM user guide for the interface that you're using.

## 4.5. Formatting Your Data

Formatting your data simply means telling your system what you want your output to look like. There are three methods for formatting your data:

- Device dependent control characters (4.5.1)
- Device independent control expressions (4.5.2)
- Format edit (4.6)

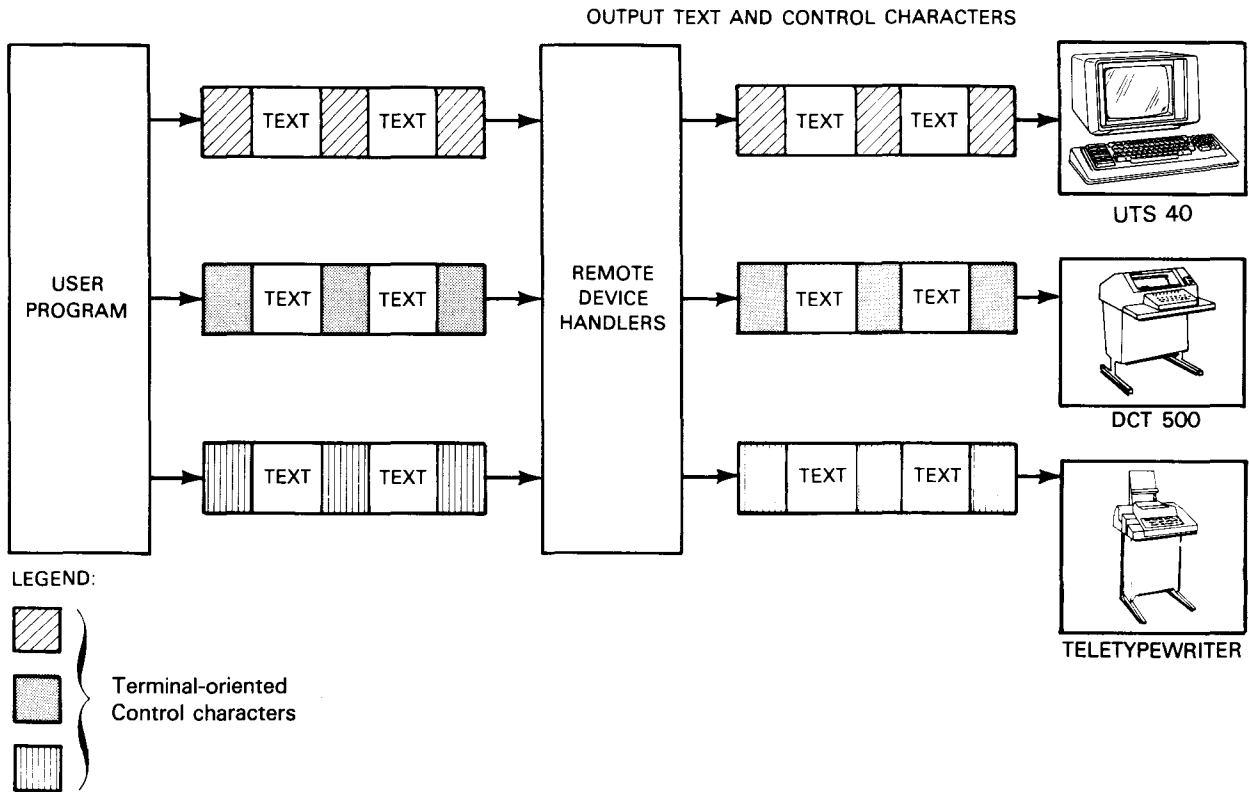
To illustrate the three types of formatting, let's assume you have a message and you want to output it to three different terminals in your network: a DCT 500, a UTS 400, and a teletypewriter.



Since they are different types of terminals, they have different control characters directing formatting. Let's look at how you would use each of the three types of formatting to send the same message to your terminals.

### 4.5.1. Device Dependent Control Characters

With *device dependent* formatting, you must embed unique control characters for each type of terminal you're sending the message to.



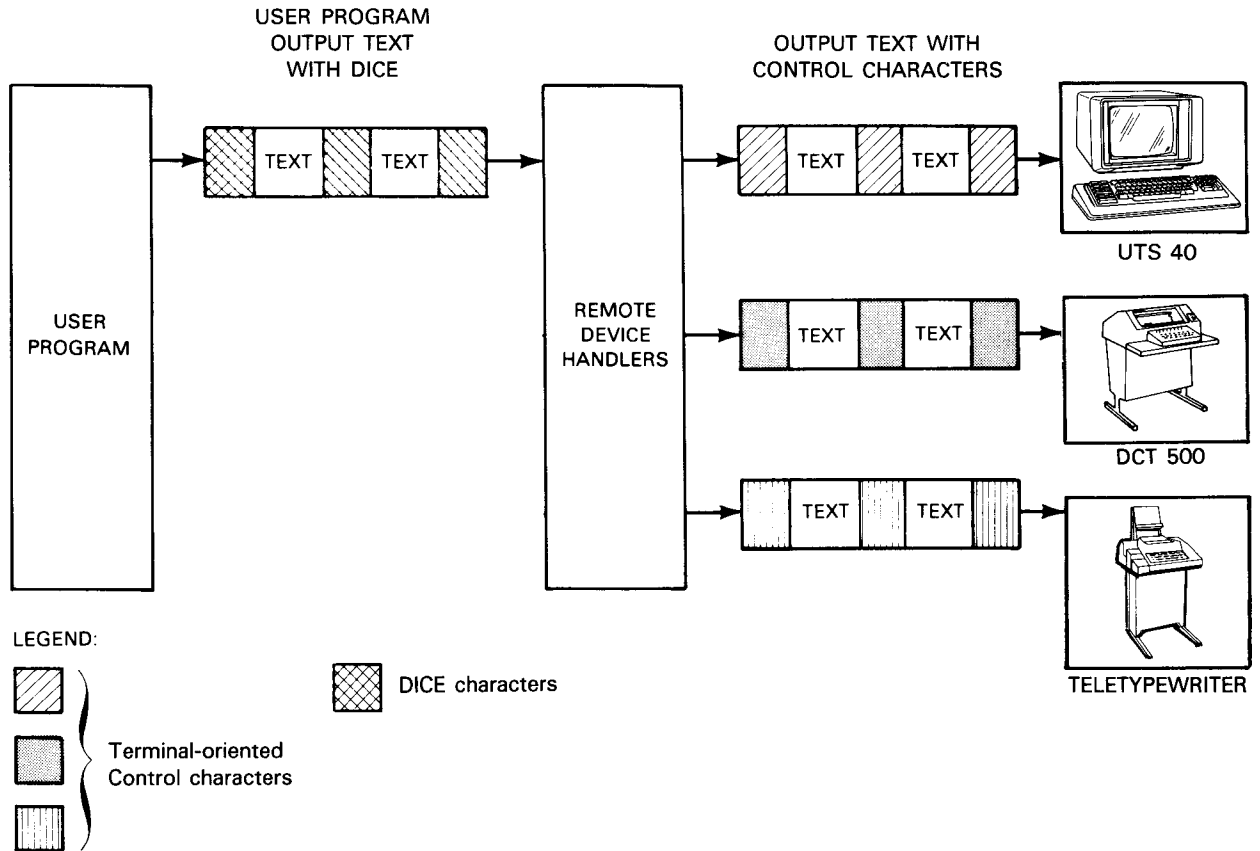
In this case, you'll need one set of control characters for your UTS 40, one for your DCT 500, and another for your teletypewriter.

This can become complicated because the control characters representing certain functions for one terminal can be entirely different for another terminal. The way to simplify this process is by using device independent control expressions (DICE).

### 4.5.2. Device Independent Control Expressions

*Device independent* control expressions (DICE), which you place in your output message, are 4-character expressions that format the message for each terminal you're sending to.

The format of DICE is changed by the remote device handler into control characters for each destination terminal. This eliminates the need to include control characters for each terminal because the remote device handler interprets the sequence into the unique code of the terminal you're sending to. The DICE sequences never appear on your terminal; you see only the message itself.



To use DICE on input, you can designate an ON or OFF state by specifying the DICE= operand of the TERM macro when you define your network. The default is DICE=(ON), which tells the remote device handler to create input DICE sequences according to your terminal cursor movements. This is particularly useful when you are receiving messages from one terminal and switching them to another terminal. If you specify DICE=(OFF), the remote device handler does not convert format control characters into DICE sequences for your program.

There are two ways to write DICE sequences:

- Hexadecimal notation (EBCDIC)
- DICE macroinstructions

The following two subsections describe both of these methods.

### Hexadecimal Notation

When using a hexadecimal notation, DICE sequences are presented in a 4-byte format:

DLE HEX 10	FUNCTION CODE	m FIELD	n FIELD
---------------	------------------	---------	---------

where:

**DLE**

Is the data link escape control character and always marks the beginning of a DICE sequence. This EBCDIC character has the hexadecimal value of 10.

**FUNCTION CODE**

Is always the second byte of a DICE sequence and contains the DICE operation code. On output, the remote device handler analyzes and converts this code into control characters to perform a certain function at your destination terminals. When your message is sent, it is formatted according to these control characters.

The function codes for DICE operations are contained in Table 4-1.

*Note: Table 4-1 gives the DICE functions for UNISCOPE formatting only. For formatting other types of terminals, consult the appropriate terminal hardware manual.*

**m FIELD and n FIELD**

Are operands of the function code that represent the horizontal (*m*) and vertical (*n*) coordinates for screen formatting.

**Table 4-1. DICE Codes and Functions**

Code (Hex.)	Function	Interpretation
01	Set coordinates	Move cursor to row <i>m</i> and column <i>n</i> .
02	Forms control	Move cursor to row <i>m</i> and column <i>n</i> .
03	Forms control with clear (unprotected)	Move cursor to row <i>m</i> and column <i>n</i> , and erase unprotected display.
04	New line control	Move cursor to beginning of next line and then <i>m</i> lines down and <i>n</i> columns to the right.

continued

Table 4-1. DICE Codes and Functions (cont.)

Code (Hex.)	Function	Interpretation
05	New line control with clear	Move cursor to beginning of next line and then $m$ lines down and $n$ columns to the right, erasing the display between the start and end positions.
06	Current position control	Move cursor $m$ lines down and $n$ columns to the right.
07	Current position control with clear	Insert $n$ "significant" spaces (DC3 characters).
08	Beginning of current line control	Move cursor to start of current line and then $m$ lines down and $n$ columns to the right.
09	Set tab stop	Set a tab stop at row $m$ and column $n$ .
0A	Forms control with clear (protected and unprotected)	Move cursor to row $m$ and column $n$ , and erase display.
0B	Erase line	Erase to end of line.

### DICE Macroinstructions

DICE macros operate the same way as the hexadecimally written DICE codes, but they are easier to use because they eliminate the tedious hexadecimal coding.

DICE macros are easily recognized because they include the prefix ZO#, a mnemonic suffix, and two decimal numbers. When these macros are assembled, they expand into the 4-byte format of the hexadecimally coded DICE sequences.

Table 4-2 lists DICE macros along with their corresponding function codes and their meanings.

Table 4-2. DICE Macroinstructions and Their Functions

DICE Macro	Code	Function	Interpretation
ZO#COORD	01	Set coordinates	Move cursor to row <i>m</i> and column <i>n</i> .
ZO#FORM	02	Forms control	Move cursor to row <i>m</i> and column <i>n</i> .
ZO#FORMC	03	Forms control with clear (unprotected)	Move cursor to row <i>m</i> and column <i>n</i> , and erase (unprotected) display.
ZO#POS	04	New line control	Move cursor to beginning of next line and then <i>m</i> lines down and <i>n</i> columns to the right.
ZO#POSC	05	New line control with clear	Move cursor to beginning of next line and then <i>m</i> lines down and <i>n</i> columns to the right, erasing the display between the start and end positions.
ZO#CUR	06	Current position control	Move cursor <i>m</i> lines down and <i>n</i> columns to the right.
ZO#CURC	07	Current position control with clear	Insert <i>n</i> significant spaces (DC3 characters).
ZO#BEG	08	Beginning of current line control	Move cursor to start of current line and then <i>m</i> lines down and <i>n</i> columns to the right.
ZO#TABS	09	Set tab stop	Set a tab stop at row <i>m</i> and column <i>n</i> .
ZO#FORMA	0A	Forms control with clear (protected and unprotected)	Move cursor to row <i>m</i> and column <i>n</i> , and erase display
ZO#ERSLN	0B	Erase line	Erase to end of line.

### 4.5.3. DICE Summary

This subsection summarizes the DICE functions and the three classes of terminals ICAM works with: CRT terminals, character-oriented printer terminals, and page printing terminals. For the details of DICE, you should turn to the ICAM interface user guide for the interface you are using.

#### CRT Terminals

- Input - DICE gives locations of text on the screen.
- Output
  - Move cursor to row  $m$  and column  $n$ .
  - Move cursor  $m$  rows down and  $n$  columns to the right.
  - Clear screen.
  - Insert spaces.
  - Set tabs.

#### Character-Oriented Printer Terminals

- Input - Form feeds, carriage returns, and line feeds in the message.
- Output
  - Carriage return and line feeds.
  - Advance  $m$  lines and  $n$  columns to the right.
  - Advance to line  $m$ , column  $n$ .

#### Page Printing Terminals

- Input - Indicates end of input record.
- Output
  - Advance  $m$  lines.
  - Advance 1 line.
  - Advance to top of form.

## 4.6. Format Edit

In 4.5, we looked at DICE to simplify message formatting. The optional format edit feature simplifies message formatting even further. With it, you specify a network definition:

- *Terminal line width* - When this value is reached, the remote device handler causes the terminal to advance to the beginning of the next line.
- *Number of lines per page* - When this value is reached, the remote device handler causes the terminal to advance to the beginning of the next page.
- *One or two editing codes* - When the remote device handler encounters either one, it causes the terminal to advance to the beginning of the next line.

Instead of specifying the line widths and page lengths, you can use the following defaults:

- DCT 475, 500, and 1000 - 132 characters per line, 54 lines per page
- Teletypewriters - 72 characters per line, and no limit on number of lines per page
- UNISCOPE and UTS 400 terminals - Either 64 or 80 characters per line and no limit on number of lines per page

In the following example, we have input assembler constants to a format edit network whose specification was a default of 80 characters to a UTS 400 screen.

Input:

```
TEXT DC C' WHEN WE TALK ABOUT ICAM LINE AND TERMINAL SUPPORT WE X00800000
      ARE REALLY TALKING ABOUT WHAT THE REMOTE DEVICE HANDLER' 00810000

DC C'S DO. THEY ARE THE LINK BETWEEN THE SOFTWARE AND THE HX00820000
  ARDWARE. THESE ROUTINES: -OPEN AND CLOSE LINES AND HANDX00830000
  LE ALL INPUT AND OUTPUT: -TRANSLATE THE CODE OF YOUR M' 00840000

DC C'ESSAGES FROM EBCDIC USED IN THE COMPUTER TO WHATEVER CX00850000
  ODE YOUR TERMINAL USES: -HELP YOUR PROGRAMS PROVIDE COX00860000
  NTROL INFORMATION FOR FORMATTING YOUR MESSAGES THROUGH ' 00870000

DC C'DEVICE CONTROL EXPRESSIONS(DICE) AND FORMAT EDIT -ANX00880000
  D PROVIDE FOR ERROR RECOVERY DURING INPUT AND OUTPUT. TX00890000
  HE RELATIONSHIP OF YOUR PROGRAM WITH THE REMOTE DEVICE ' 00900000

DC C'HANDLERS...' 00910000
```



**Output:**

```
WHEN WE TALK ABOUT ICAM LINE AND TERMINAL SUPPORT WE ARE REALLY TALKING
ABOUT WHAT THE REMOTE DEVICE HANDLERS DO. THEY ARE THE LINK BETWEEN THE
SOFTWARE AND THE HARDWARE. THESE ROUTINES: -OPEN AND CLOSE LINES AND
HANDLE ALL INPUT AND OUTPUT;; -TRANSLATE THE CODE OF YOUR MESSAGES
FROM EBCDIC USED IN THE COMPUTER TO WHATEVER CODE YOUR TERMINAL USES;;
-HELP YOUR PROGRAMS PROVIDE CONTROL INFORMATION FOR FORMATTING YOUR
MESSAGES THROUGH DEVICE CONTROL EXPRESSIONS[DICE] AND FORMAT EDIT;;
-AND PROVIDE FOR ERROR RECOVERY DURING INPUT AND OUTPUT. THE
RELATIONSHIP OF YOUR PROGRAM WITH THE REMOTE DEVICE HANDLERS...
```

For the details of using format edit, see the current version of the *ICAM Operations Guide* (UP-9745).

## 4.7. Translate Tables

The remote device handlers translate your messages from EBCDIC, used by the host computer, to whatever code is used by your terminals. ICAM supplies standard translation tables for the terminals that it supports. There may be times, however, when you want to supply your own translation table or modify one of the ICAM standard translation tables. For example, if you attached a special printer to a terminal, it has special codes to control printing. You can supply a translation table for the remote device handler to convert certain EBCDIC characters to the print codes. For example, if your program uses the percentage sign (%) to mean skip to the top of the next page, the remote device handler converts it to the appropriate print code.

## 4.8. Status and Error Codes

The remote device handlers report certain error and status codes to your program. Because the information given your program depends on the interface used, the following subsections are summaries. For details on getting and using these codes, go to the appropriate ICAM interface manual.

### 4.8.1. Input Error Notification

ICAM does not inform standard interface programs of any input errors. When they occur, the remote device handler takes care of them. If an error results in marking down the last terminal on the line, your program gets a logical line-down notification. (See the discussion of line connection errors in 4.2.) You can use a message processing routine (see Section 6, especially 6.3) to inform your program of many input errors. ICAM informs direct data interface programs of any input error that occurs.

Input errors break down into three broad classifications:

- *Hardware errors* - The remote device handlers do not attempt recovery. ICAM breaks the circuit and notifies your program and system operator.
- *Transmission errors* - These occur if a terminal does not respond to a poll or a parity error occurs. In both cases, the remote device handler retries the poll that led to the error. If the error still occurs after several retries, the remote device handler marks the terminal down. (See "Polling with Buffered Interactive Terminals" earlier in this section for a description of error recovery with polling. The process is similar with most buffered batch terminals.)
- *Invalid framing characters* - The remote device handlers successfully received a message but the framing characters are invalid. One possibility is that the terminal address is not one declared at network definition. If the terminal is buffered, the remote device handler asks the terminal to retransmit the message; if the terminal is unbuffered, the remote device handler throws the message away and no error recovery is attempted. Another possibility is that the header is unreadable. Again, the remote device handler attempts recovery with buffered terminals but not with unbuffered terminals. If recovery isn't successful after several retries, ICAM marks the terminal down.

### 4.8.2. Output Error Notification

As with input messages, ICAM gives the direct data interface program the most information on the status of output messages. Standard interface programs, however, can learn the status of an output message through the output delivery notification request feature. But ICAM notifies the standard interface program only whether or not the output is successfully delivered. You can use a message processing routine (see Section 6, especially 6.3) to inform your standard interface program of the details of output errors.

The remote device handlers inform the direct data interface program of the following output error conditions:

- The message was successfully delivered.
- The message was not successfully delivered. - For standard interface programs, this status is given only after the retries are exhausted. For direct data interface programs, this status is given after each retry because the programs must request subsequent retries. Also, the direct data interface programs get this status broken down into the following categories for UNISCOPE, UTS 400, and UTS 4000 terminals:
  - Single line communications adapter reported an error.
  - No acknowledgment came from the terminal.
  - Error occurred in the tape cassette system.

For DCT 1000 terminals only, the status can be:

- Prolonged busy signal

For teletypewriters, DCT 475, DCT 500, and UTS 10 terminals, you can get:

- Error during output to primary device, or else a break was received during output

For batch terminals, the unsuccessful output status is not qualified.

- If output was not successful during a write to an auxiliary device, this is reported to your program.
- The line was logically marked down.

### 4.8.3. Terminal Statistics

At network definition, you can request ICAM to keep statistics on each terminal. These statistics give you:

- Total number of messages received
- Total number of times the remote device handler requested input retransmissions
- Total number of messages transmitted to the terminals
- Total number of times the terminals requested the remote device handler to retransmit output
- Total number of polls sent out by the remote device handler
- Total number of times the terminals replied to polls with the no-traffic response

## 4.9. Output Delivery Notification Request

ICAM provides an optional feature called output delivery notification request (ODNR) that notifies your program when a message it sent was delivered. This feature is available for standard interface programs and IMS.

Your program uses this feature by setting a field and supplying a message identifier in the output DTFCP before issuing the output request. When the message is delivered, your program receives control at a location it specified. At this time, the identifier and status (message delivered or error status) is returned in general registers. Details on how to use output delivery notification request notice is provided in the *ICAM Standard MCP (STDMCP) Interface Programming Guide* (UP-8550).



# Section 5

## Buffers and Queues

### 5.1. General

The network buffers and queues are the heart of the standard and transaction control interfaces. They isolate the programs of both interfaces from the physical problems of communications. Direct data interface programs, by contrast, work directly with the remote device handlers and direct input, output, and error recovery. Figure 5-1 compares a standard and a direct data interface system.

In the following discussions, buffers and queues may be determined to reside in main storage or on disk according to the application requirements. If on disk, they may be defined as disk-buffered files or disk-queued files. For example, a disk-buffered file is required for input when ICAM is used with IMS. A disk-queued file is used for input and output with the standard interface and may be used for output with IMS. The advantages and disadvantages of each are discussed in more detail in 5.4.

Even though the major part of this section deals with network buffers and queues, it also covers two related subjects: line/VLINE buffers (5.2) and activity request packets (5.3). You should read these subsections regardless of the interface you use.

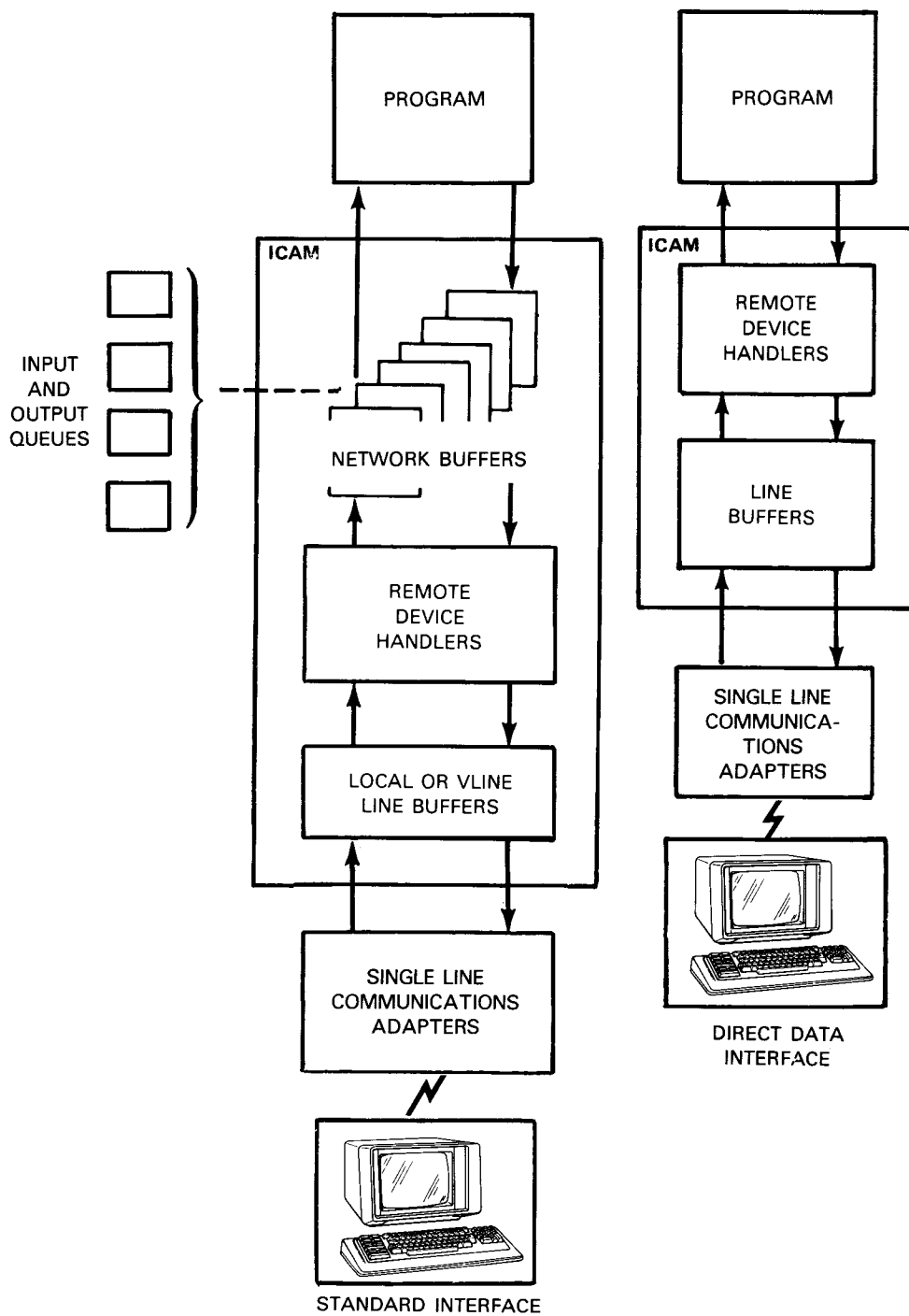


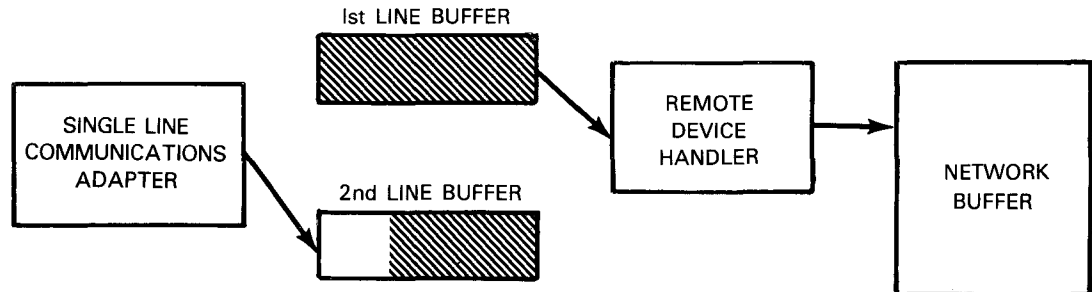
Figure 5-1. Comparison of Standard and Direct Data Interfaces

## 5.2. Line Buffers and VLINE Line Buffers

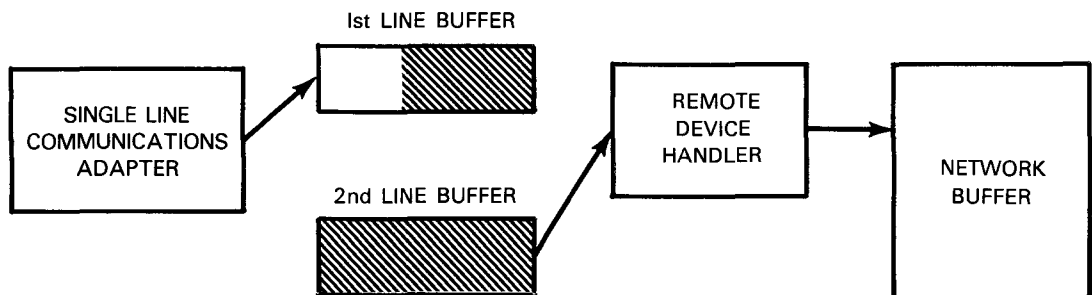
ICAM creates special buffer areas to temporarily store message data as it is transferred from a single line communications adapter to main storage on input and while it is waiting to be transferred to a single line communications adapter on output. These areas are called line buffers. If a line buffer works in conjunction with a VLINE, the line buffer is called a VLINE line buffer.

### 5.2.1. Line Buffers

For most terminals, ICAM creates two equal-sized line buffers for each line in your network to use as input and output staging areas. During input, the single line communications adapter fills the first line buffer byte by byte with the incoming message. If the message doesn't fit into the first line buffer, the communications adapter begins filling the second line buffer. As it does so, the remote device handler processes the contents of the first line buffer and puts it in a network buffer:

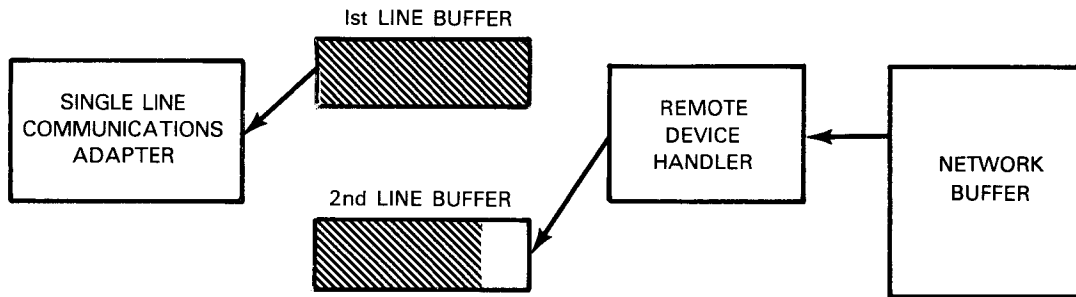


If more of the message remains after filling the second line buffer, the communications adapter switches back to the first line buffer and overwrites its contents. Meanwhile, the remote device handler begins to process the contents of the second line buffer:



Alternating between line buffers continues until the entire message is received.

During output, the process reverses and the remote device handler fills the line buffers with the outgoing message:



If a line buffer fills faster than the remote device handler can process its contents, part of a message is lost. The longer a line buffer is, the more time a remote device handler has to process its contents. ICAM supplies default lengths for line buffers based on how the line is used. It bases the line buffer length for lines supporting batch terminals on message size (it knows the message size when you specify the kind of terminal you have), and you cannot change the line buffer length.

ICAM supplies a default line buffer length for interactive terminals based on transmission rate. It uses transmission rate because, with interactive terminals, message size varies a lot. (A message may only be a few characters on part of a display screen or it may be the entire screen including format control characters.)

You specify line buffer length on the LINE macro when you define your ICAM network. Refer to the *ICAM Operations Guide* (UP-9745) for details.

You should specify a line buffer size for UNISCOPE, UTS 400, UTS 4000, and teletypewriter terminals (UTS 10) because the default sizes specified for these interactive terminals are only minimum sizes that will always work. They use a minimum amount of storage, but they are not optimum sizes from a performance standpoint. Larger sizes up to one half of message size may be used; however, we recommend that you do not specify line buffer sizes greater than 128 words (512 bytes). This is because lengthy ICAM processing can interfere with the timely dispatching of other tasks in the system. If you have a mixed ICAM system of teletypewriter, UNISCOPE, UTS 400, or UTS 4000 terminals, we recommend you use the same line buffer size for all lines.

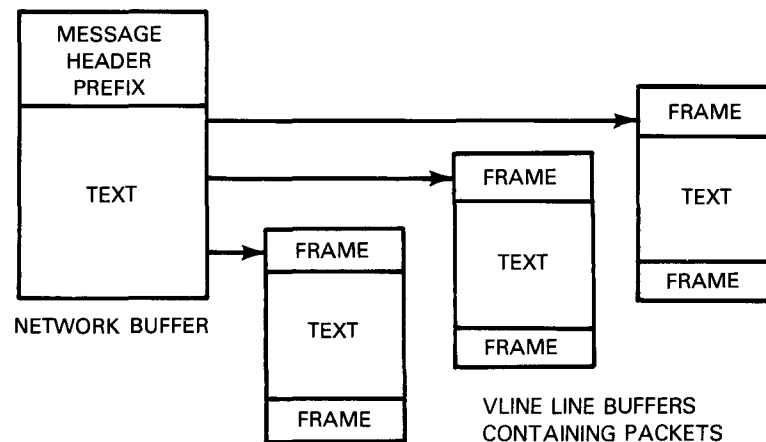
Local workstations are the only interactive terminals that do not follow normal line buffer use. They have a single line buffer that must be large enough to hold complete messages including format information. Decrease the size only if the largest message including format information is less than the default length; otherwise, increase the length as appropriate. This may be necessary if screen format services are being used by IMS and the text contains many FCC sequences.



## 5.2.2. VLINE Line Buffers

VLINE line buffers are input and output staging areas in main storage that ICAM creates to temporarily store messages as they are sent and received over a virtual line (VLINE). VLINE is the name for a dedicated circuit that connects your computer to another computer or to a public data network. The protocols used on VLINEs dictate that messages must be sent in segments or packets, and that each packet is a separate message framed according to the protocol used.

Each VLINE line buffer holds one packet, and they are not used in pairs and toggled the way line buffers are for most lines. They are used (much like network buffers) from a pool of VLINE line buffers established at network generation time. ICAM acquires them from the VLINE line buffer pool as needed and releases them back to the pool when they are no longer needed. The following figure shows how one network buffer might relate to several VLINE buffers to make up one message.

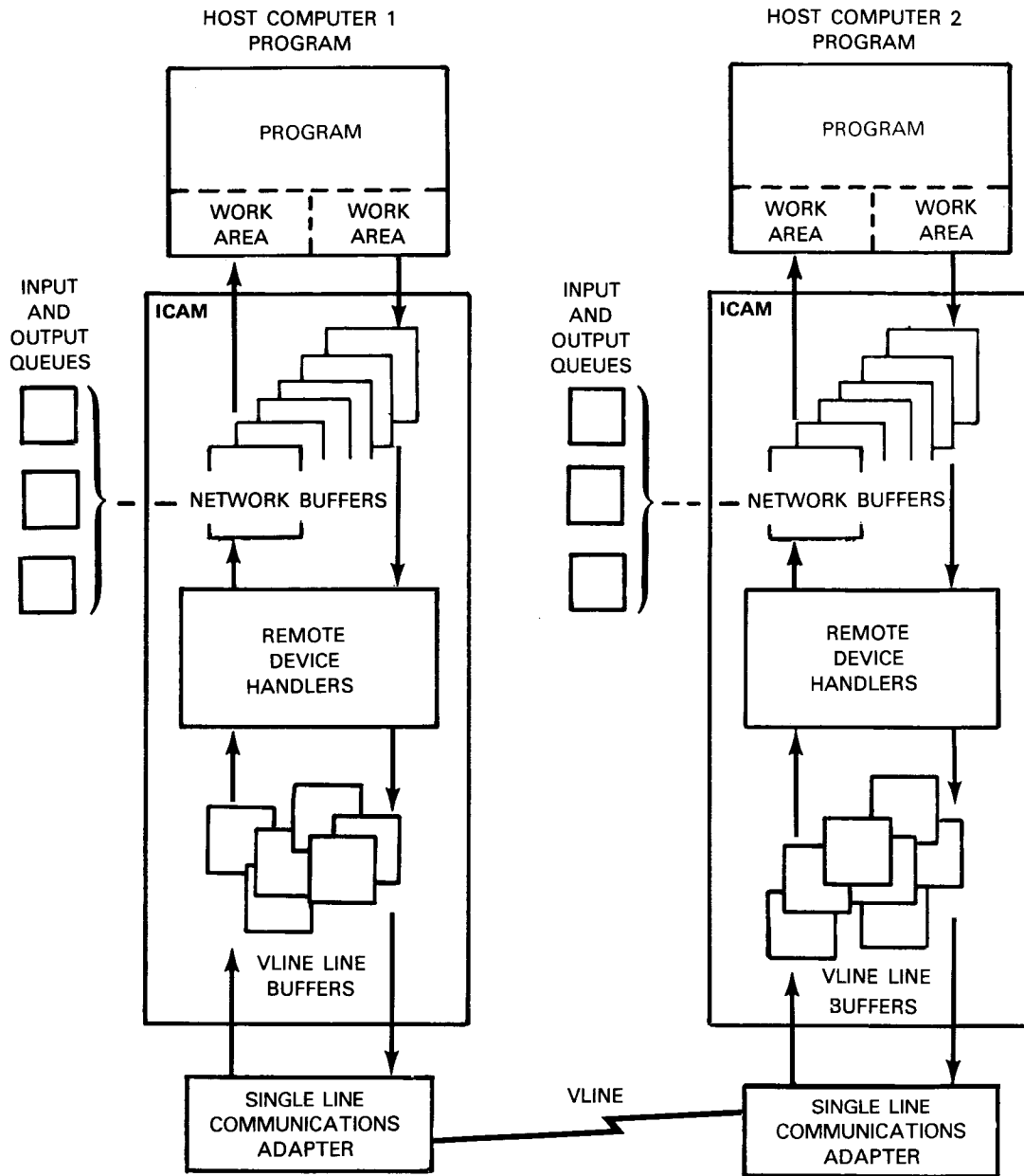


At network generation, you declare the number, size, and threshold value of the VLINE line buffers. While there is no formula for calculating these values, the following are some considerations:

- *Number* - Specifies the number of VLINE line buffers. The greater the traffic over the VLINE, the greater the number of VLINE line buffers you should declare.
- *Size* - You declare the amount of text, in words, each packet contains. ICAM automatically sizes the VLINE line buffers large enough to contain the amount of text plus message frames. If the VLINE connects to a public data network, the size must be the text size used by the network. If the VLINE connects directly to another host computer, the text size must match in the two hosts.
- *Threshold value* - The threshold value sets the number of VLINE line buffers held in reserve. When all but this number of VLINE line buffers are in use, ICAM slows the traffic rate until more VLINE line buffers become free. You can't lose a message because all the VLINE line buffers are in use.

## Buffers and Queues

The software and hardware elements associated with VLINEs are shown in the following illustration.



NOTE:

Additional global elements are introduced in subsequent subsections.

### 5.3. Activity Request Packets

Activity request packets are work areas used by the ICAM routines. You have no control over their use, nor does their use affect the arrangement of your network. We mention them only because, at network definition, you must declare the number needed. The *ICAM Operations Guide* (UP-9745), contains the formula used to determine the number.

If you request it to, ICAM keeps statistics on activity request packet usage. The statistics are almost identical to those kept on network buffer usage (5.4). They tell you the following:

- Number of times activity request packet requests were rejected - ICAM rejects requests when all activity request packets are in use and an ICAM routine needs one. In most cases, ICAM terminates when an activity request packet request must be denied. If the statistics ever show a reject request, redefine your network with more activity request packets.
- Total number of times any number of activity request packets were available - The statistics give the number of times none were available, the number of times one was available, the number of times two were available, and so forth, up to the number of activity request packets in the system.

You get the activity request packet statistics by dumping the system or by using the journal utility. Both methods give the same statistics, but they're easier to obtain with journaling. See the *ICAM Utilities Programming Guide* (UP-9748) and the *ICAM Operations Guide* (UP-9745) for formats and procedures of journaling.

Figure 5-2 shows the format of the statistics area in a system dump.

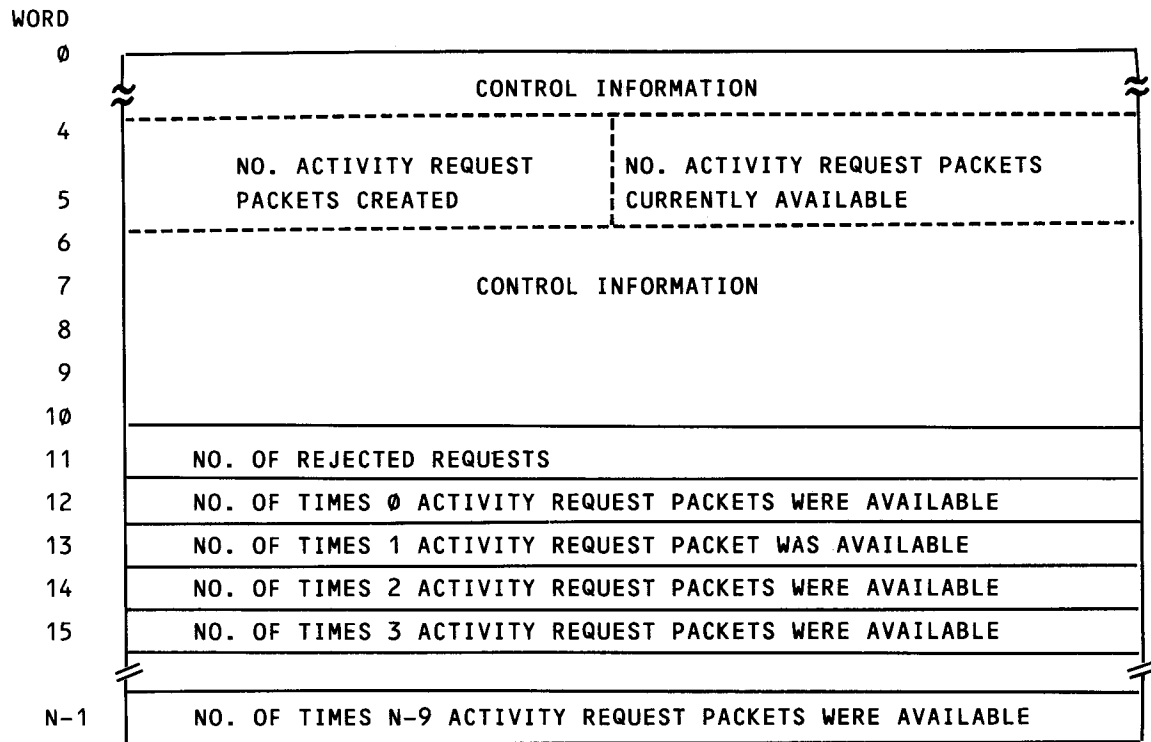
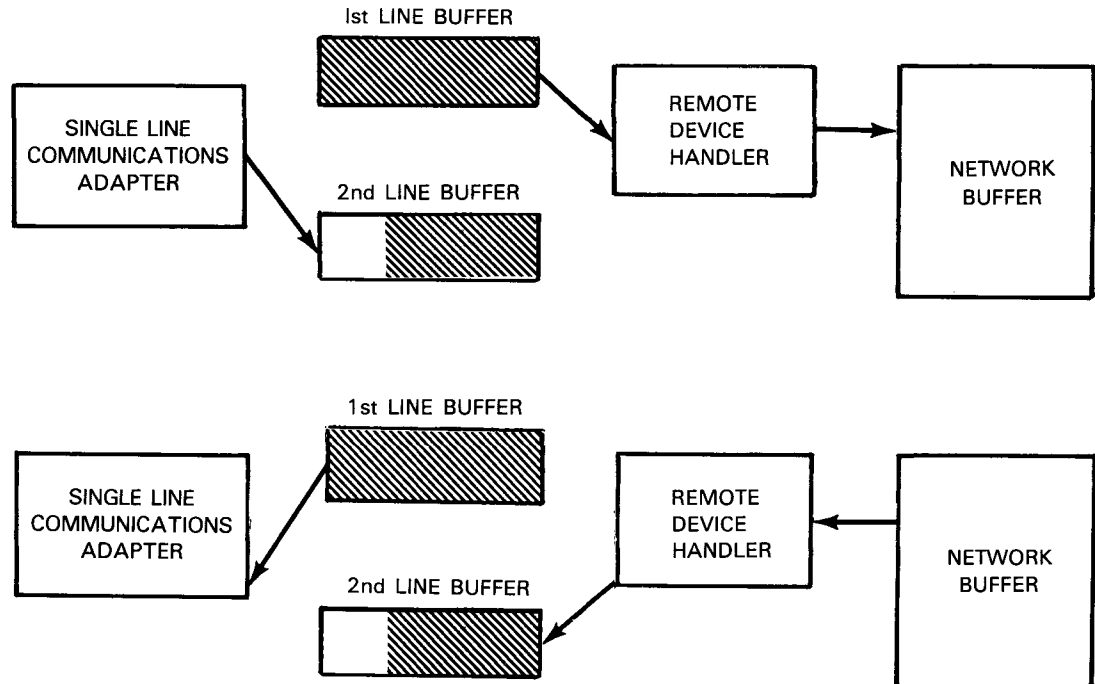


Figure 5-2. Activity Request Packet Statistics in a System Dump

## 5.4. Network Buffers

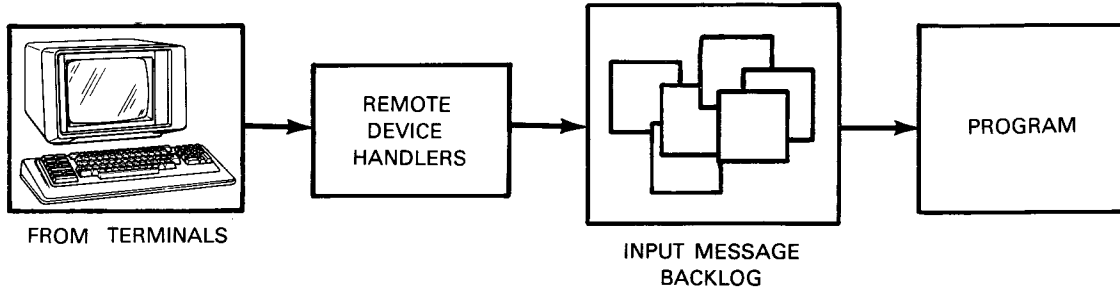
The network buffers are areas in main storage used for two purposes. They hold messages as ICAM processes them, and they store messages that cannot be immediately sent to their destination (main storage queuing). The processing is minimal. Remote device handlers place input messages in the network buffers as they process them, and take output messages from the network buffers to prepare them for transmission.



Once the messages are in the network buffers, message processing routines, if any, process the messages, then they are queued. As Section 6 describes in detail, the message processing routines handle transmission errors, examine message headers (beginning of messages), and route messages to different destinations.

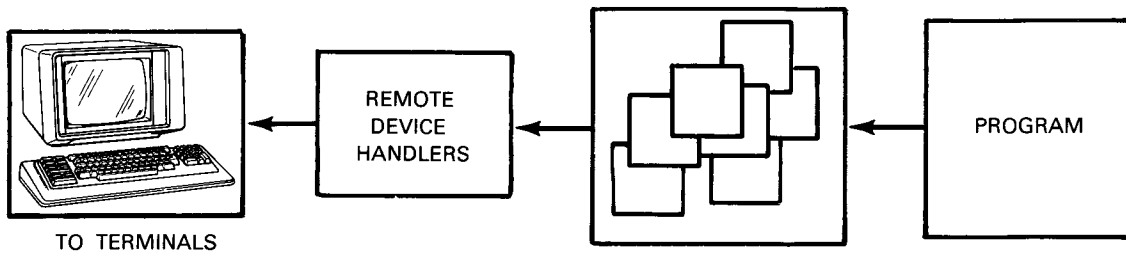
Normally, messages reside in the network buffers for short periods of time, measured in fractions of a second, before going to their destination. ICAM must store messages in the network buffers, however, under three circumstances:

1. *ICAM delivers input messages to programs only when the programs request input.* On an average, programs must request input as fast or faster than ICAM receives the input from the terminals. Otherwise, a continuously growing backlog of messages is created, filling the network buffers. But programs may temporarily request input slower than ICAM receives input messages, creating a small backup.



ICAM stores these input messages in the network buffers until the programs request them.

2. *ICAM sends output messages to the terminals as soon as it can.* Physical constraints on the transmission speed of messages can create a backlog. For example, a 1000-byte message takes about 30 seconds to send over a line capable of transmitting 300 bits per second. Your programs, operating much faster, can create many output messages in this time. Normally, they should not create output messages faster than ICAM can send the messages to the terminals. Otherwise, a continuously growing backlog of messages is created, filling the network buffers. But programs may temporarily create output messages faster than ICAM sends them, creating a small backlog.

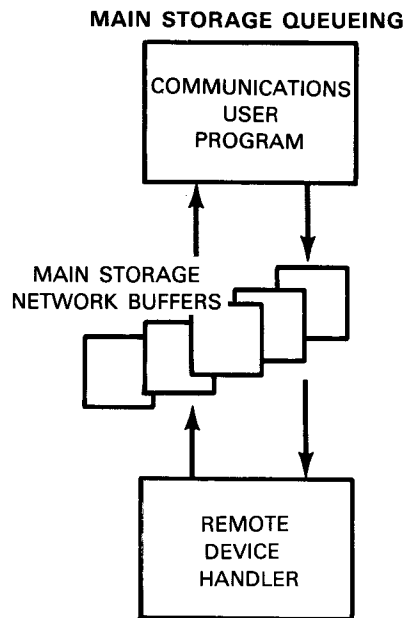


ICAM stores these output messages in the network buffers until it can send them to the terminals.

3. *If a program is inoperative or a terminal is down, ICAM stores any messages for it in the network buffers.* Unlike the previous cases, the backlog created can last for seconds or minutes and can continue to grow during that time.

When you define the network, you create a pool of network buffers by declaring its characteristics: the type of buffering (main storage or disk), the size of the network buffers, and the number of network buffers. So many factors affect the network buffer pool, we can't give hard and fast rules for declaring the characteristics. Instead, we recommend you use the guidelines in this subsection to estimate the characteristics. Then, define your network, regularly check the network buffer statistics, and change the characteristics as needed.

Deciding between main storage and disk queuing is basically a matter of examining the trade-offs:

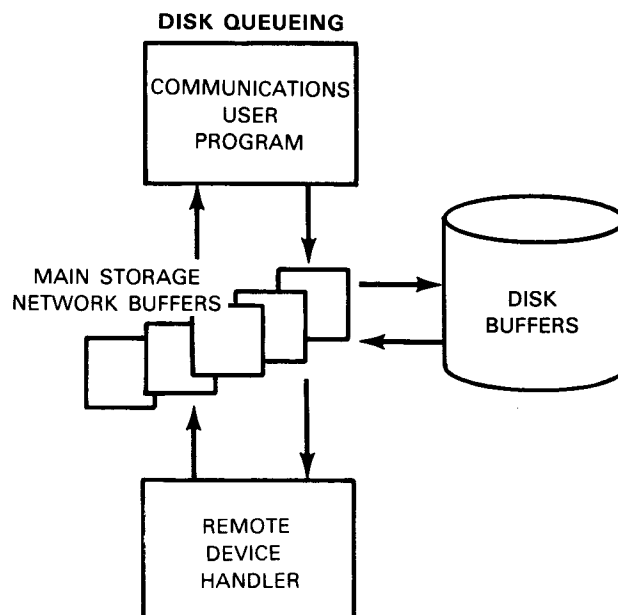


Advantage

ICAM processing 10-15 percent faster than with disk buffering

Disadvantages

- Many bytes of main storage committed to the network buffers. A modest network buffer pool of twenty 1020-byte buffers requires 20.4K bytes of main storage.
- If the system crashes, all messages in network buffers are lost.
- Not good for message storage because many extra network buffers needed, requiring more main storage



Advantages

- Few main storage buffers needed, meaning little main storage committed to the network buffers
- If the system crashes, all messages in disk buffers are saved.
- Good for message storage because disk file containing network buffers can hold as many network buffers as needed

Disadvantage

ICAM processing 10-15 percent slower than with main storage queuing

## Buffers and Queues

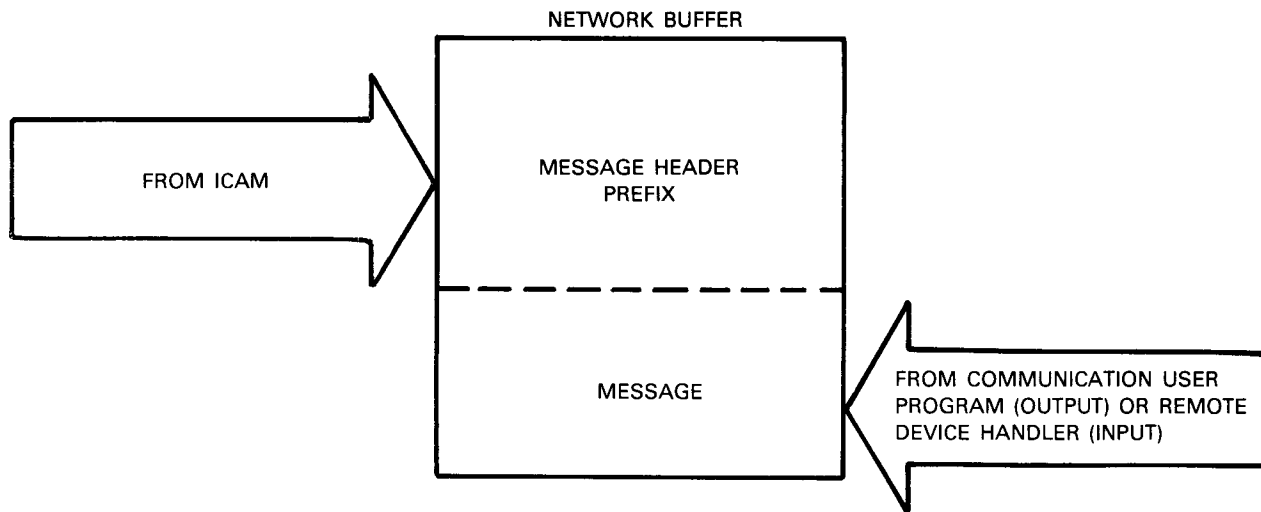
---

If you have large messages, we recommend disk queuing. When messages of 1000 bytes or more are stored in main storage buffers, large amounts of main storage are needlessly tied up. This problem doesn't occur with disk queuing.

You can mix main storage and disk queuing. Messages associated with one queue can go to main storage buffers, and messages associated with another queue can go to disk buffers.

By using disk queuing, you can store messages for later transmission, perhaps when line rates are cheaper. Your programs send messages to queues associated with disk buffers, where they're stored until the lines are activated. ICAM then reads the messages from disk and transmits them.

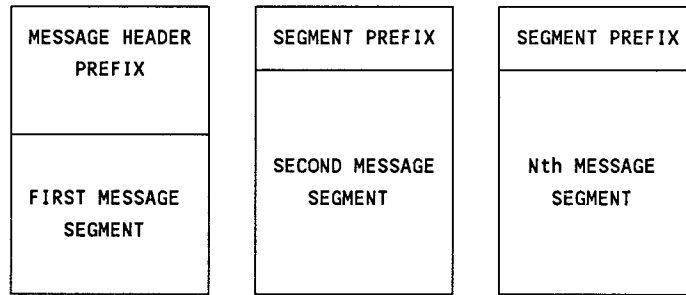
When declaring the network buffer size, you must make it at least large enough to hold a message header prefix supplied by ICAM and part of a message. The network buffer format looks like this:



The message header prefix contains information used by ICAM, like the message origin and destination.

Often, network buffers do not hold complete messages. When a message is too large for a single network buffer, ICAM segments it and places it in two or more network buffers. ICAM prefixes each network buffer containing a message segment other than the first with a smaller segment prefix as shown here:





The actual sizes of the message header prefix and segment prefix vary according to the type of interface you use, whether you use disk or main storage queuing, and if you require journaling. The size of the message header can range from 88 to 120 bytes depending on your system, while the segment prefix can range from 12 to 16 bytes. You specify the size you need in the BUFFERS macro, as discussed in the *ICAM Operations Guide* (UP-9745).

The size of the message area in your network buffers depends on several factors:

- *Disk queuing* - With disk queuing, we recommend that each network buffer be large enough to hold any size message without segmentation. This way, ICAM reads or writes a message from the disk file in a single input or output operation. Otherwise, ICAM reads or writes each segment of a message in a separate input or output operation. Each input or output operation increases the ICAM processing time on that message by 10 to 15 percent.

If your largest messages are 2000-byte messages from interactive terminals, you don't use journaling, and assuming a 100-byte message header prefix size, the network buffer size is:

100 bytes (message header prefix)  
+2000 bytes (message size)

2100 bytes or 525 words

**Note:** *Network buffer size is declared in words not bytes. If the number of bytes is not an even multiple of 4, round it up to the nearest multiple of 4, divide by 4, and declare that many words.*

- *Main storage queuing* - With main storage queuing, we recommend that each network buffer be large enough to hold all but your largest messages. This doesn't affect ICAM processing time, but it holds down the total number of network buffers, reducing the amount of main storage used.

If 30 percent of your messages are 80 bytes long, 30 percent are 133 bytes long, 35 percent are 200 bytes long, and 5 percent are 1000 bytes long, you should create network buffers capable of holding 200 bytes of message because most of your messages are that size or smaller. With journaling and a message header prefix of 112 bytes, network buffer size is 312 bytes or 78 words.

## Buffers and Queues

---

With buffers of this size, it takes four network buffers to hold a 1000-byte message. The first buffer holds 200 bytes; the second, 288 bytes; the third, 288 bytes; and the fourth, 224 bytes. For example:

<u>Buffer 1</u>	<u>Buffer 2</u>	<u>Buffer 3</u>	<u>Buffer 4</u>
112 bytes (message header prefix)	16 bytes (prefix)	16 bytes (prefix)	16 bytes (prefix)
+200 bytes (message size)	+288 bytes (message)	+288 bytes (message)	224 bytes (remaining portion of message)
<hr/> 312 bytes	<hr/> 304	<hr/> 304	<hr/> 240

- *Batch terminals* - With batch terminals, each network buffer must be large enough to hold the largest record used by your batch terminals.

If your batch terminals use 80-byte input records from a card reader and 133-byte output records to a printer, you base your network buffer size on the 133-byte output records. With journaling, assuming a message header prefix size of 112 bytes, the network buffer size is:

112 bytes (message header prefix)  
+133 bytes (record size)

245 bytes or 60 words

Interactive terminals don't impose any additional network buffer sizing considerations.

Deciding on the number of main storage network buffers needed is more difficult than deciding on their size because it's usually harder to estimate message flow than average message size. The first step is deciding whether to use main storage or disk queuing. You create a pool of main storage buffers either way. But in main storage queuing, the main storage buffers act as both holding and processing areas, while in disk queuing, the main storage buffers act only as processing areas. We'll show how to determine the number of main storage buffers needed with both types of network queuing, starting with main storage queuing.

Main storage queuing requires enough main storage buffers to hold all the messages ICAM has at a given time. Because backups usually occur on output and not input, calculate the number of output messages your programs create in a time interval (for example, a minute), subtract the maximum number of messages sent to the terminals in that interval, and add an insurance factor to give yourself some cushion.

Determining the number of messages your programs create is an estimate probably unique to your system. Part of it depends on the type of applications. Batch applications, for example, typically create more messages than interactive systems (although interactive messages are usually larger and may require more network buffers). Another part depends on the number of users. More users usually mean more messages. And another part depends on the type of lines and terminals.

Your programs shouldn't continuously create more messages than can be sent because a large backlog of messages is inefficient. From these considerations, you need two numbers: the average and maximum number of messages. We'll show how to use them later.

To determine message number, consider:

- Your applications
  - Are they batch or interactive?
  - Do they create large or small messages?
  - Do they create many or few messages?
  - How many different applications are there and how do these considerations vary with each of them?
- The users
  - Are there many or few of them?
  - Do they use the communications system in the same way?
  - Do they use the communications system at the same time?
  - Do they use the communications system for long or short periods?
- The equipment
  - How fast do your lines transmit messages?
  - How fast do your terminals receive messages?
  - How much do these vary between different lines and terminals?
  - How many terminals on different lines?

Determining the maximum number of messages each single line communications adapter sends is reasonably simple.

On a given line, it's the smaller of two numbers: the line speed or the terminal speed. If a line transmits at 2400 bits per second and the terminal receives at 4500 bits per second, use the 2400 bits per second. Remember, for multiterminal lines, one terminal at a time receives messages. If you have six terminals on a line capable of receiving 300 bits per second and the line transmits 300 bits per second, only 300 bits per second can be transmitted over that line. Also, most lines and terminals receive and send messages. If your terminals send as many messages as they receive, the number of output messages sent over a line is half the theoretical maximum.

## Buffers and Queues

---

To determine the number of output messages sent, consider:

- How fast do your lines transmit messages?
- How fast do your terminals receive messages?
- Do you have more than one terminal per line?
- What's the mixture of output to input messages?
- How long does it take to send a message?
- How much time does polling require? How do these considerations vary from line to line?

Determining the insurance factor means judging the number of network buffers needed to hold messages for long periods (anywhere from seconds to hours). Part of this is the difference between the maximum number of messages created and the maximum number of messages sent to the terminal in a given interval. For example, if your programs generate a maximum of one hundred 80-byte output messages a minute and send them over three lines rated at 300 bits per second each, your calculation looks like this:

$$\begin{array}{r} 100 \text{ messages per minute} \\ \times 80 \text{ bytes each} \\ \hline 8,000 \text{ bytes per minute} \\ \times 8 \text{ bits per byte} \\ \hline 64,000 \text{ bits per minute} \end{array}$$
$$\begin{array}{r} 300 \text{ bits per second} \\ \times 60 \text{ seconds per minute} \\ \hline 18,000 \text{ bits per minute} \\ \times 3 \text{ the number of lines} \\ \hline 54,000 \text{ bits per minute on all lines} \end{array}$$
$$\begin{array}{r} 64,000 \text{ bits to send} \\ -54,000 \text{ bits that can be sent} \\ \hline 10,000 \text{ bits that must be buffered} \\ \div 8 \text{ bits per word} \end{array}$$
$$\begin{array}{r} 1,250 \text{ bytes to be buffered} \\ \div 80 \text{ bytes to the message} \\ \hline 16 \text{ messages to be buffered} \end{array}$$

The first calculation gives the number of bits to be transmitted.

The second calculation gives the total number of bits the communications adapter can send per minute. (The 300 per second assumes the lines transmit as many input as output messages.)

The third calculation gives the difference in messages, between the number of messages that can be sent. You must create enough buffers to hold the difference.

You need 16 network buffers to handle a 1-minute backlog, 32 network buffers to handle a 2-minute backlog, and so forth. You cannot continue at this rate for long. To clear a 1-minute backlog, only 69 messages can be created in the next minute. Thus, 85 messages a minute is the most that can be created without creating a backlog.

At this point, you're ready to use the average and maximum number of messages you calculated before. If the average output load is 25 messages per minute for 3 lines, you need 8 network buffers to handle the average load (3 to hold messages as the communications adapter outputs them, 3 to hold input messages, and 2 just in case). You also need enough to hold the backlog as the maximum number of messages is reached. If you never create messages at the maximum rate for more than 2 minutes, you need an additional 32 network buffers for a total of 40. For 80-byte messages, you devote at least 6400 bytes of main storage to network buffers. (Remember, the network buffers also hold message header prefixes.)

The insurance factor calculations do not take into account messages held in the network buffers for down terminals and programs. These messages may reside in the network buffers for long periods of time, increasing the number of network buffers needed. To avoid this problem, use disk queuing, the alternate destination capability (described near the end of this section), the intercept queue capability (described near the end of this section), or a message processing routine (described in Section 6).

To determine an insurance factor, consider:

- What's the difference between the maximum and average number of output messages created?
- How fast are messages sent to the terminals?
- Are messages held in main storage buffers for down terminals and programs?

Disk queuing requires enough main storage buffers to hold only the messages ICAM processes at a given time. Messages reside in main storage buffers just long enough to be processed, written to disk, read from disk, or transmitted. The last is the problem because it takes somewhere between a fraction of a second and several seconds while the others are much faster. You must create at least one main storage buffer. In general, however, the system works fastest if you create one main storage buffer for each line plus a few additional main storage buffers. For example, if you have three lines, create five or six main storage buffers. You don't declare the number of network buffers in the disk file. ICAM creates as many network buffers in the file as it needs.

Whether you use main storage or disk queuing, create extra network buffers when in doubt. When you look at the buffer statistics described later, you'll see how many are not used and then you can reduce their number.

On the other hand, creating too few network buffers degrades ICAM processing with disk queuing and causes messages to be lost in main storage queuing. The following shows what happens if the network buffers fill with main storage queuing and ICAM receives a message from a program or terminal.

**IF:** Every network buffer contains a message or message segment

**AND**  **THEN**

your program tries sending a message

ICAM rejects the message and notifies your program. Your program should resend the message.

a message is received from an uncontrolled terminal

ICAM ignores the message. No retry is made. Neither your program nor your terminal is notified that the message is lost.

a message is received from a controlled terminal

ICAM rejects the message but asks the terminal to resend it in case a network buffer becomes free. After five unsuccessful retries, the message is lost. Neither your program nor your terminal is notified that the message is lost.

An optional network buffer threshold capability lets you create at network definition a network buffer reserve. It serves two purposes:

1. When all network buffers except the reserve fill, ICAM stops soliciting input from the terminals and empties network buffers by sending their messages to the terminals and programs. After a substantial portion of the network buffers empty, ICAM resumes soliciting input. Without the threshold capability, ICAM does not stop soliciting input when the network buffers fill. The network buffers empty when input slows, but messages may be lost.
2. Even though ICAM stops soliciting input when the network buffers fill, controlled terminals, uncontrolled terminals, and programs that were sent output can give ICAM new messages to hold. The network buffer reserve gives you a margin, although the reserve may fill faster than network buffers are emptied.

With the network buffer statistics, which you request ICAM to keep at network definition, you monitor the use of your main storage buffers. They tell you:

- *Number of times network buffer requests were deferred* - ICAM defers buffer requests when all network buffers are full and one of its own routines requests a buffer. A deferred buffer request does not result in the loss of any messages.
- *Number of times network buffer requests were rejected* - ICAM rejects buffer requests when all network buffers are full and either a terminal or a program tries to send a message.
- *Total number of times any given number of network buffers were available* - The statistics give the number of times none were available; the number of times one was available; the number of times two were available; and so forth, up to the number of main storage buffers available.

You get the buffer statistics by either dumping the system or using the journal utility. Both methods give the same statistics, but they're easier to see with journaling. In this section, we'll show the statistics as they appear in a system dump; see the current versions of the *ICAM Utilities Programming Guide* (UP-9748) and the *ICAM Operations Guide* (UP-9745) for journaling formats and procedures.

The format of the statistics in a system dump (Figure 5-3) is seven words of control information followed by several counters. The first gives the number of deferred requests; the next, rejected requests; and then one word for each network buffer, showing how often that many network buffers were available. Figure 5-4 is an example of a buffer statistics area. Message flow in this system was particularly heavy and frequently overloaded the buffer pool. Buffer requests were rejected 562 times ( $232_{16}$ ). A good portion of the rejected requests probably involved lost messages. No buffers were available 616 times ( $268_{16}$ ). The threshold value for this system is five network buffers. By adding up the number of times five or fewer buffers were available, we see that the threshold was exceeded 1823 times ( $72F_{16}$ ). Each time it was, system performance dropped substantially.

The user of this system should redefine the network and increase the number of network buffers.

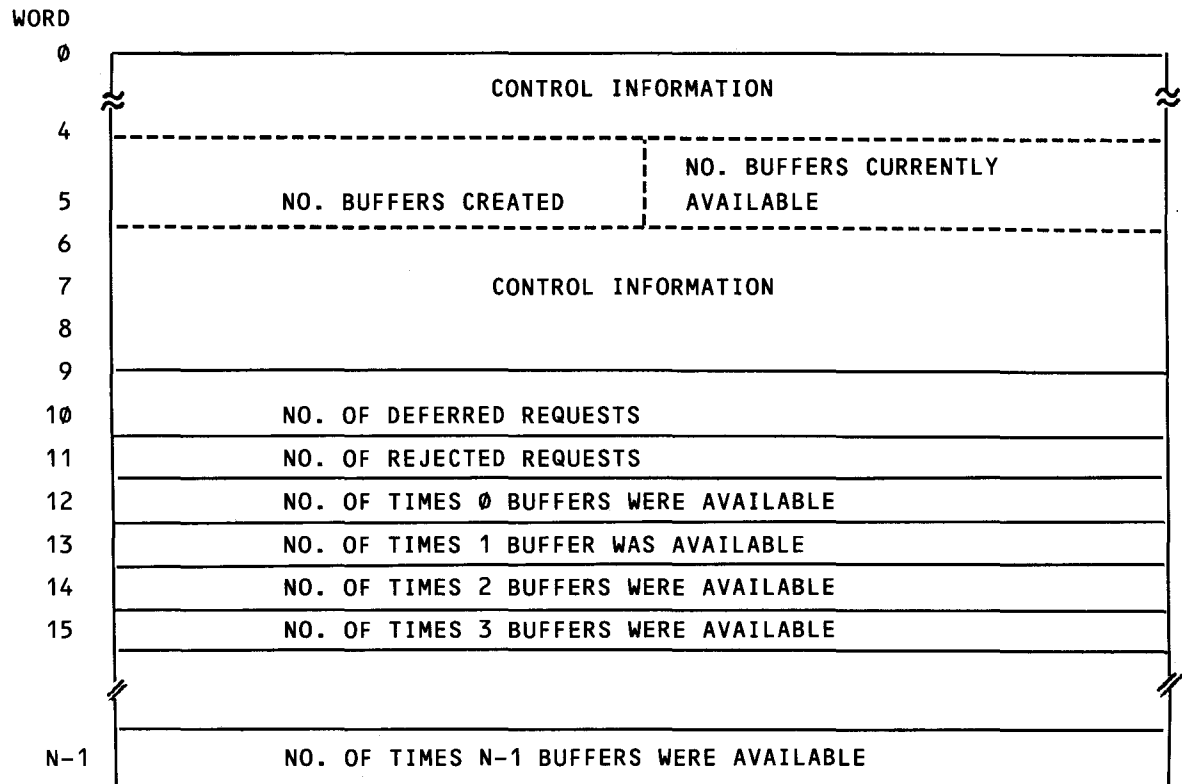
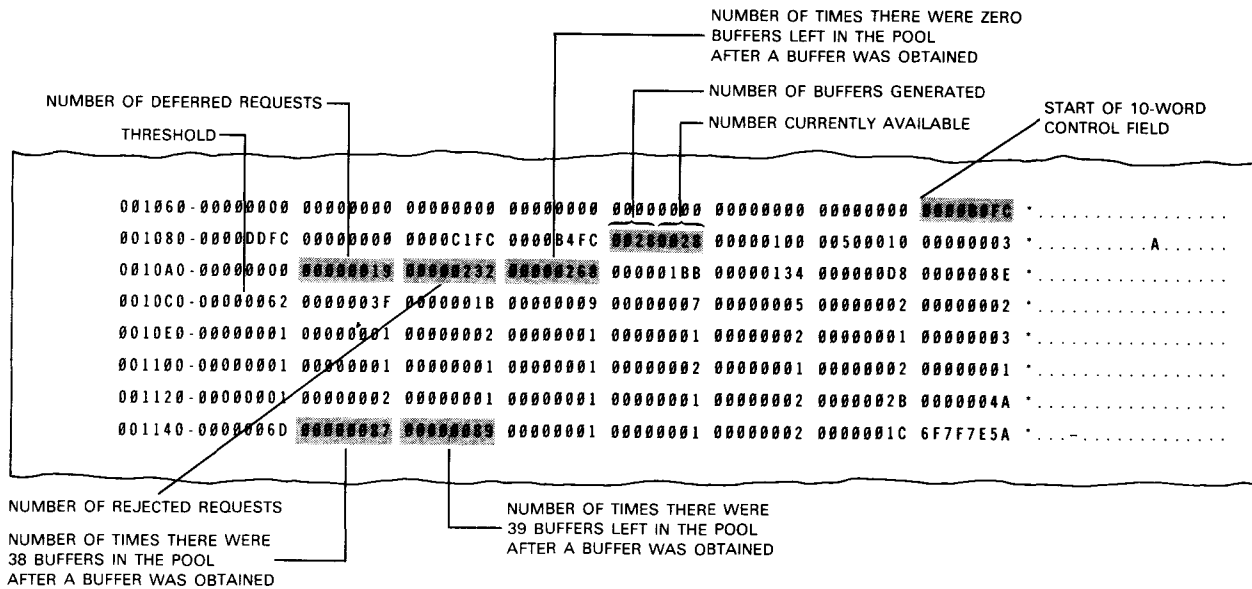
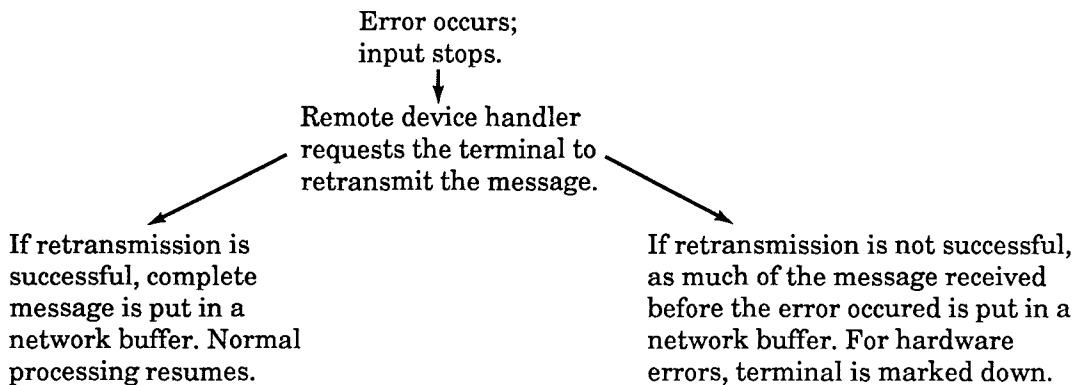


Figure 5-3. Buffers Statistics in a System Dump



**Figure 5-4. Statistics Area in a System Dump for Network Buffers**

The final network buffers topic is what happens to a message in a network buffer when an error occurs during input or output. Input errors are either hardware failures or line buffer failures. In the latter case, the line buffer is too small and parts of the message are lost. (See 5.2.) When either error occurs, the following happens:

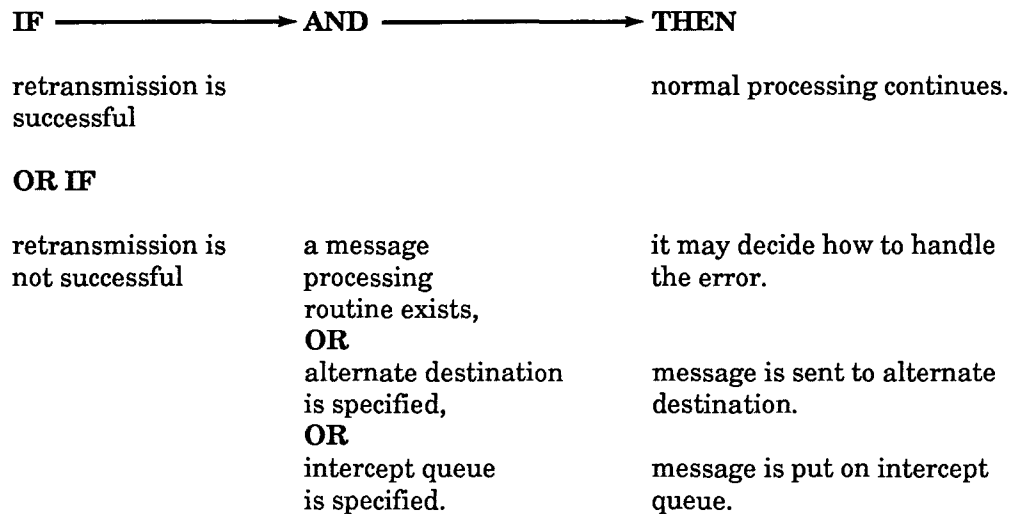


If retransmission isn't successful, ICAM discards the message fragment in the network buffer, unless you have a message processing routine. It can then decide whether to cancel the message, send it to a program or terminal (if a successful EOM was received), and/or send error messages to a program or terminal.



During output, hardware failures and line buffer failures can occur, plus two new errors: message truncation and terminal down. Message truncation happens when the message is too long for the terminal to handle and must be truncated. This is the only output error for which the message, although truncated, goes to the terminal. Terminal down is not strictly speaking an error, but it prevents the message from going to the terminal and is treated like an error.

The sequence of events for output errors is different from input errors. When an error occurs, the remote device handler tries to retransmit the message (except for truncated messages). For example:



The recovery options are hierarchical. A message processing routine takes precedence over alternate destinations, which, in turn, take precedence over intercept queues.

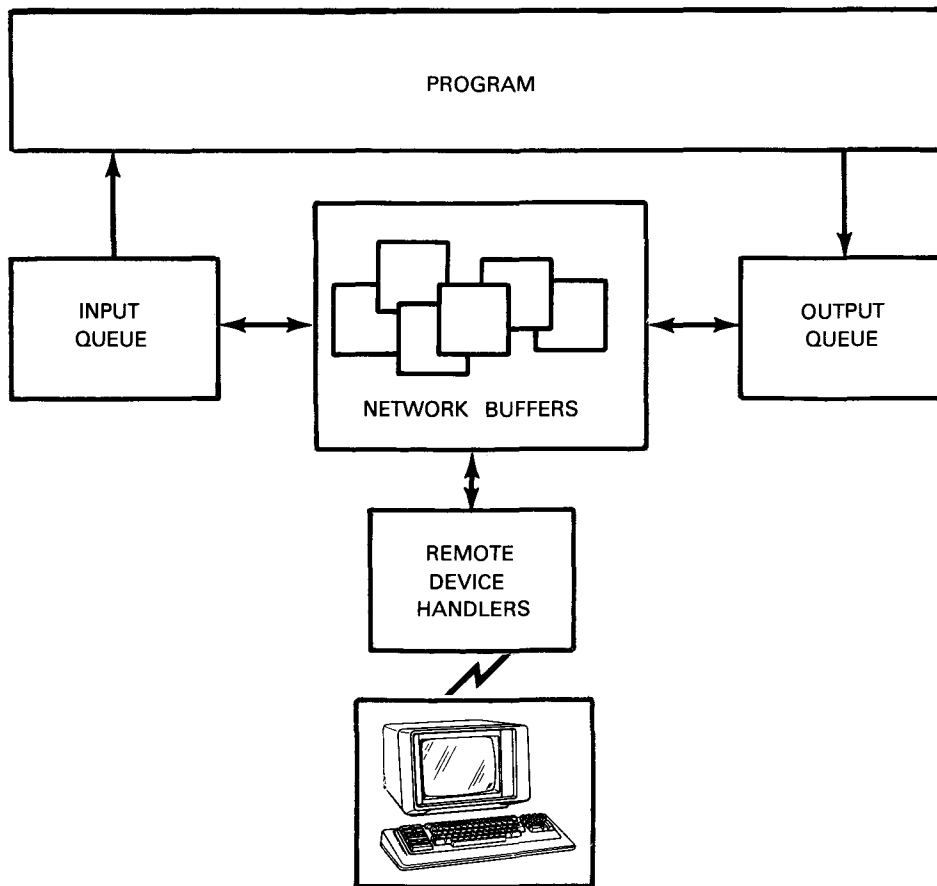
The functions of each are:

- Message processing routine, if present, receives control after every output. If an error occurred, an error message is sent to a program, terminal, or intercept queue.
- An alternate destination is another terminal, a locap file, or a process file that the message goes to when the primary terminal is unavailable.
- An intercept queue is a queue of messages intended for a terminal that's marked down. The messages remain on the intercept queue until a communications program releases them or until the terminal is marked up. The messages then go to the terminal.

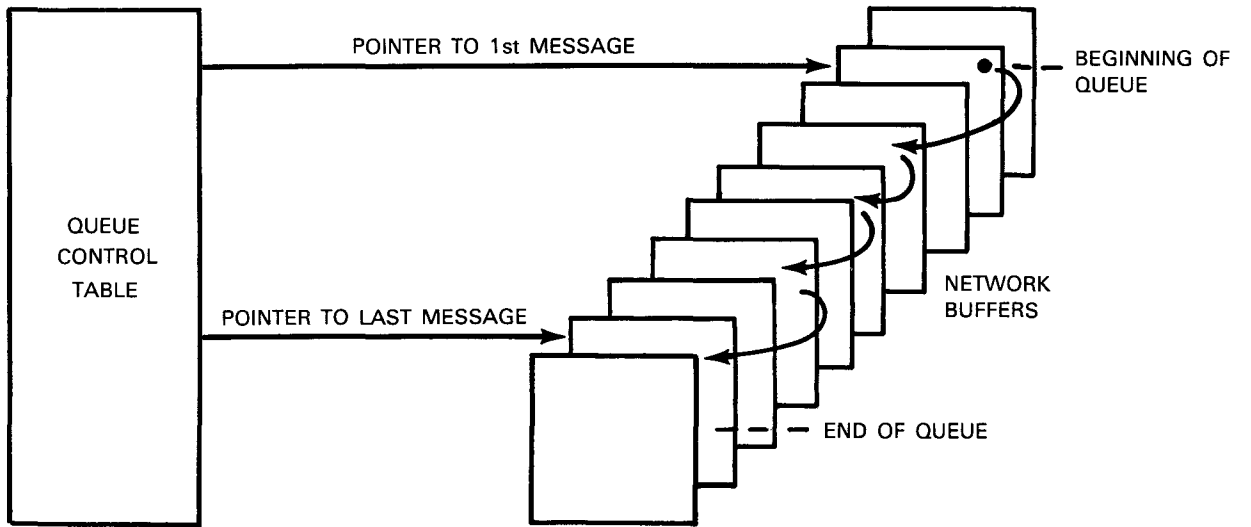
## 5.5. Queues

A queue is a set of messages having a common destination. In many ways, queues are the major feature of the standard interface because they allow you to group your messages into distinct sets. Consequently, your programs - or each routine in your program - can specialize by processing just one message type.

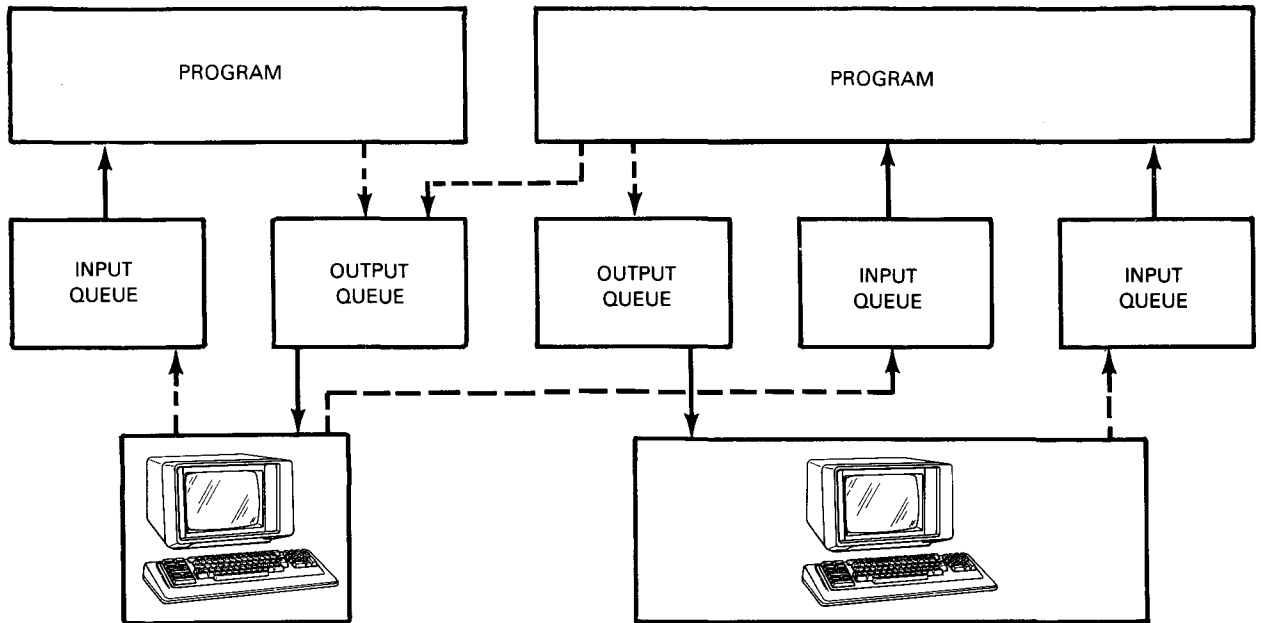
Ultimately, queues determine the destination of a message. When a message is placed on an input queue, it goes to a program; when a message is placed on an output queue, it goes to a remote device handler and from there to a terminal.



Physically, a queue is a set of messages linked together in the network buffers. A queue control table points to the location of the first and last messages in a queue. Fields in the message header of the network buffer prefixes link the intervening messages.



ICAM places new messages at the end of queues. It takes messages from the beginning of queues. Thus, queues operate on a first-in, first-out basis.



When you define your network, you need to define two systems of queues:

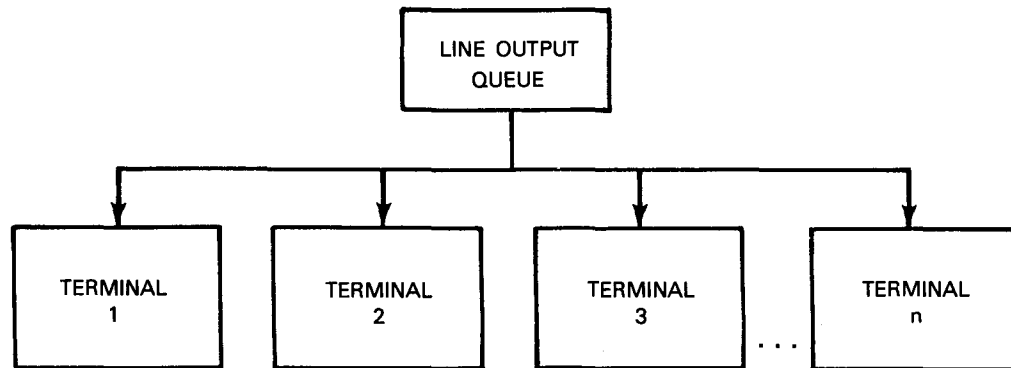
1. *Queues that support your terminals (output queues)* - When your program sends a message, it issues an output request - a PUTCP, for instance - and releases control to ICAM. ICAM performs any further processing needed and places the message on an output queue that relates to a destination. There are two kinds of output queues:

- *Line queues* - A line queue is an output queue that services some or all of the terminals on one communications line. You specify a line queue in the LINE macro when you define your network.

*Note:* Do not specify a line queue if you are using a global network.

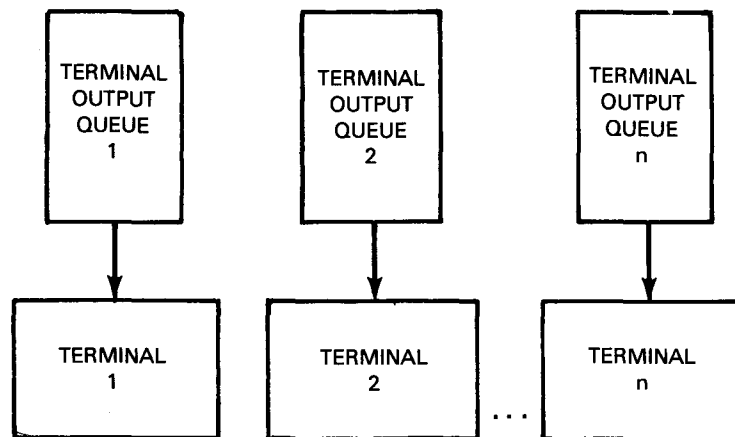
- *Terminal queues* - A terminal queue is an individual output queue established for each terminal on a line. We highly recommend terminal queues for all systems. You specify terminal queues in the TERM macro when you define your network.

The following illustration shows the difference between line and terminal queues for output, and how each is used to service terminals:



NOTE:

Do not use line output queues with a global network.



It is possible to mix line output queues and terminal output queues on the same communications line. You do this by specifying a line queue in the `LINE` macro and specifying an output queue in the `TERM` macro for each of those terminals you want to have its own queue.

2. *Queues that support your programs (input queues)* - Like terminals, each program receiving messages is associated with an input queue. But there are a number of differences. ICAM has three types of input queues:

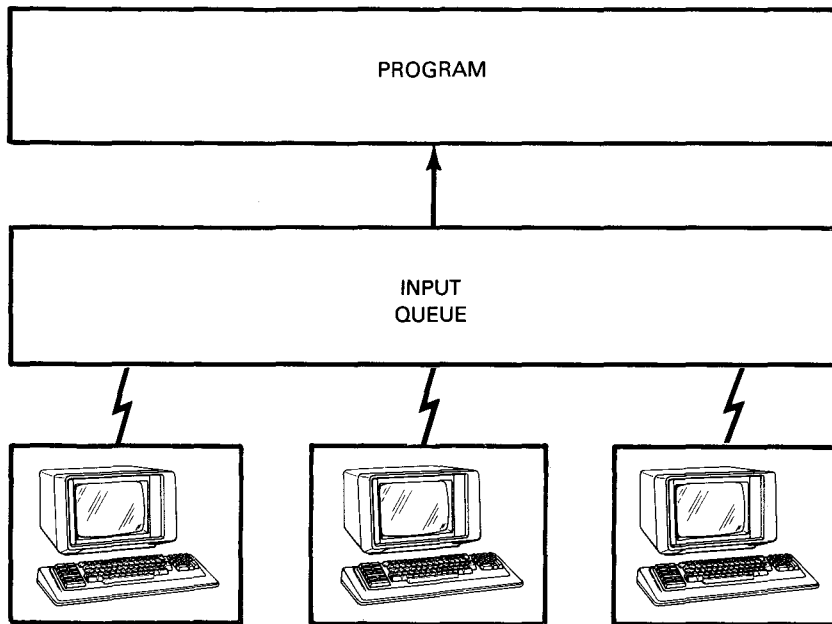
- Input terminal queues (described in detail in 5.5.2)
- Process files (described in detail in 5.5.3)
- Locap files (available only with global networks)

## Buffers and Queues

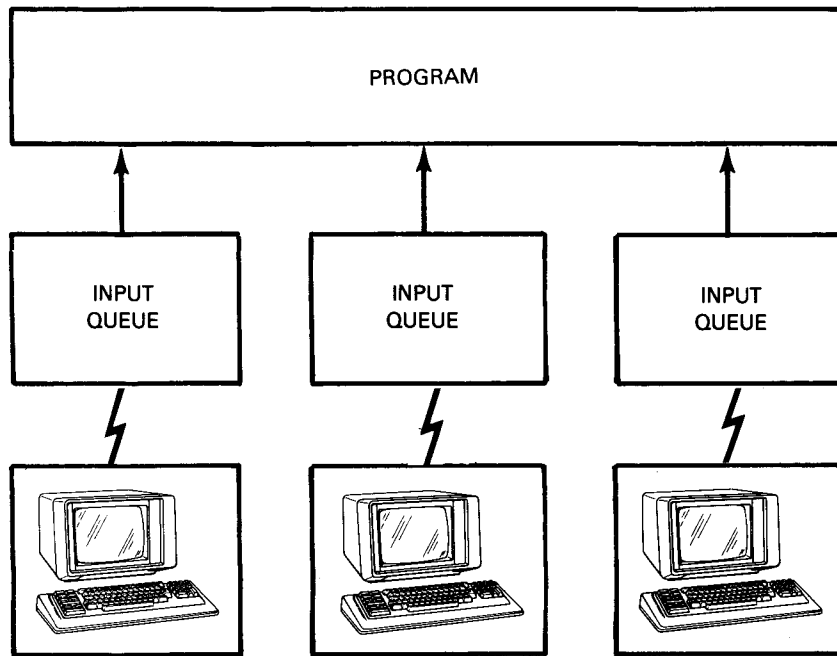
---

You can also associate a single program with multiple input queues. You can arrange these input queues in a number of ways, which we'll look at in the following paragraphs.

Let's start with the simplest input queue arrangement, one for a single application communications system. All messages contain similar types of data and need similar processing. Examples would be an inventory control system or a sales order entry system. Whatever the application, one program does the processing. You could use a single input queue or one for each terminal:

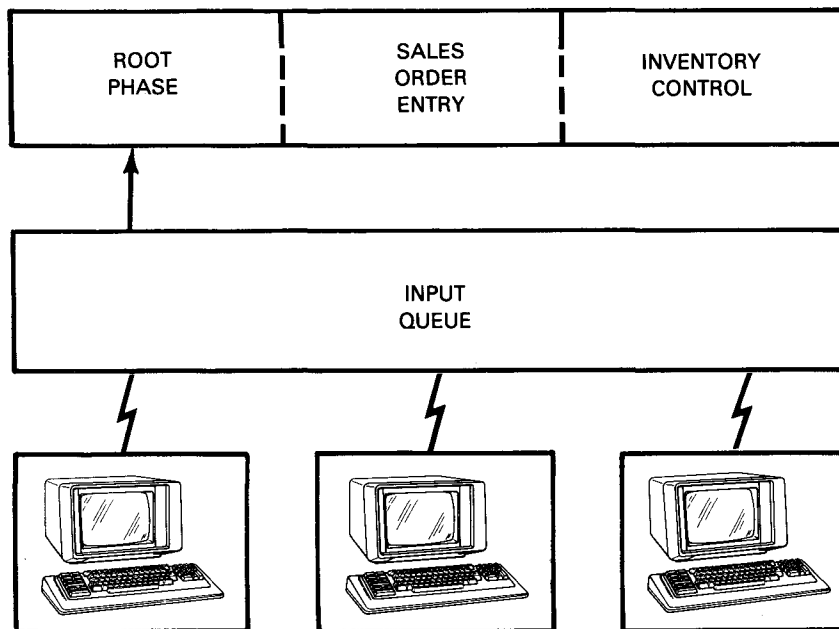


This arrangement is available only with process files and locap files.



This arrangement is available with all types of input queues.

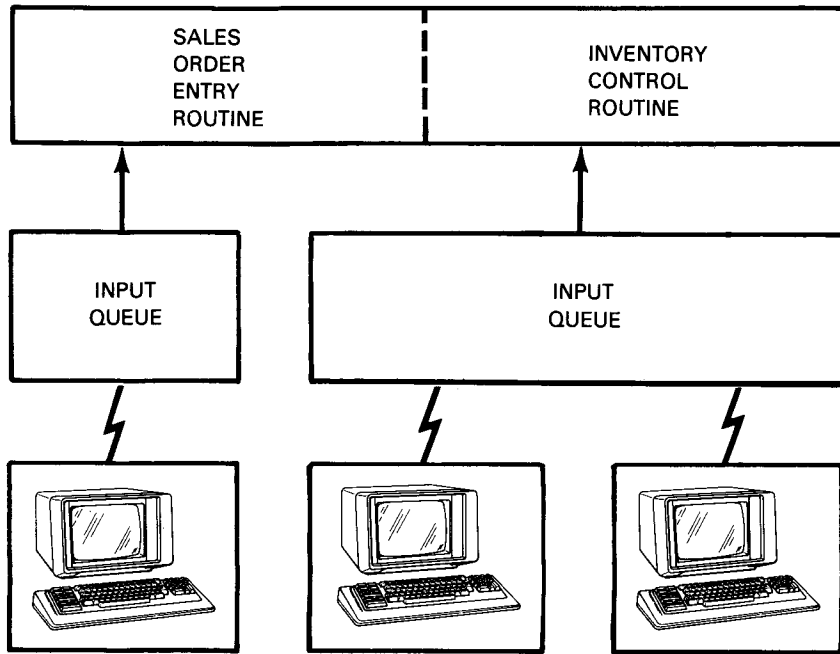
Now, let's add a second application. One program can handle both applications if it has separate routines for each application. One way of arranging the program is to have a root phase that reads the messages, sorts them, and passes them to the correct routine:



This arrangement is available only with process files and locap files. If each terminal has its own input queue, input terminal queues can be used.

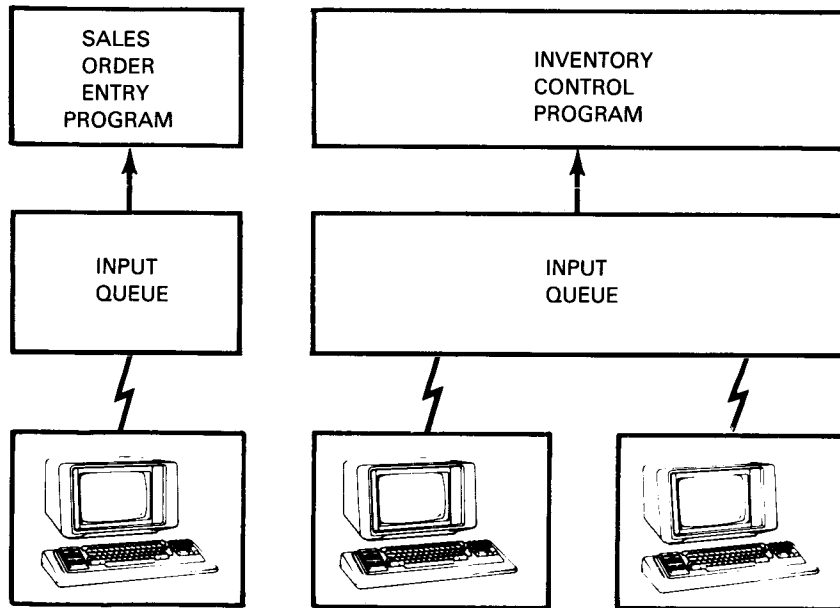
In this, as in the following examples, each terminal can have its own input queue.

Or each routine can read messages from its own input queue:



Shared queue must be a process file or locap file. Individual queue can be any type of input queue.

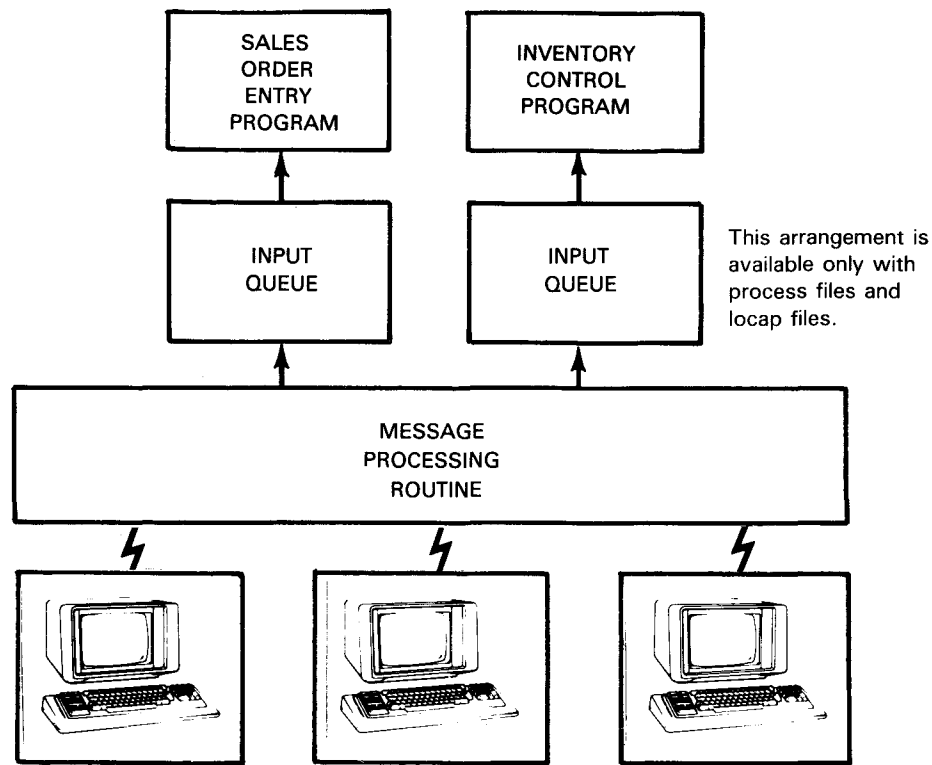
Or you can have two separate programs with their own input queues:



Shared queue must be a process file or locap file. Individual queue can be any type of input queue.

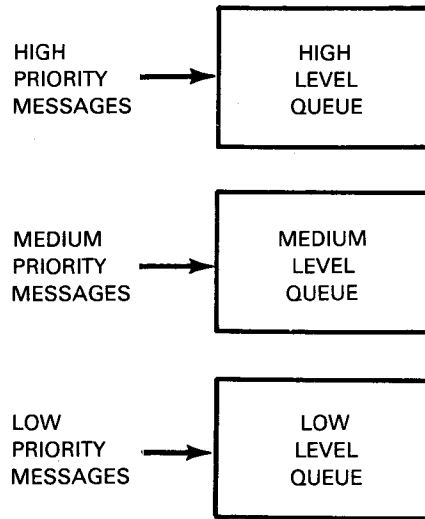


So far, the examples show messages from each terminal going to a single input queue. Many times, this is what happens: the terminals using inventory control are in the warehouses, and the terminals using sales order entry are in the sales office. But the sales personnel may need to know what the inventory levels are, and the stock clerks may need to know what is ordered. A message processing routine (Section 6) can sort the messages from terminals and route them to the proper input queue:



We'll look at other ways of using queues when we look at the different types of queues later.

With most queue types, you have the option of giving messages a high, medium, or low priority. Messages with higher priorities go to the destination before messages with lower priorities. To give your messages different priorities, you create queues with multiple levels:



Multilevel queues act as normal queues. Everything said of queues in this section holds for multilevel queues except the order in which messages leave multilevel queues. With a single-level queue, messages leave in the order in which they arrived. With a multilevel queue, messages leave according to their priority. Messages are taken from the high level queue first. If it is empty, messages are taken from the medium level queue. If both the medium and high level queues are empty, messages are taken from the low level queue. Within a single level of a multilevel queue set, messages normally are taken on a first-in, first-out basis.

Only a program or a message processing routine places messages in multilevel queues on any level other than the lowest level created. Messages go to different levels, as shown here:

<b>If a message comes from a:</b>	<b>Then the queue level is:</b>
program	level specified by the program. Default is the lowest level queue created.
message processing routine	level specified by message processing routine. Default is the lowest level queue created.
terminal (without message processing routine)	lowest level queue created.
distribution list	lowest level queue created.

As implied in the preceding paragraph, you do not have to create all three levels of a multilevel queue set. Because most of the types of queues have multilevel queuing, we give you the option of creating one, two, or three levels of queues. With a single level queue, all messages (regardless of their priority) enter and leave the queue on a first-in, first-out basis. With two queue levels, low priority messages go to the lower level queue and high priority messages go to the higher level queue. Medium priority messages go to either level queue, depending on the levels created. This table shows where messages go, with different level queues created:

<b>If you create these queue levels:</b>	<b>And you give your messages a priority, they go to the following queue levels:</b>		
	<u>Low Priority</u>	<u>Medium Priority</u>	<u>High Priority</u>
Low	Low	Low	Low
Low,medium	Low	Medium	Medium
Medium	Medium	Medium	Medium
Medium, high	Medium	Medium	High
High	High	High	High
Low, medium, high	Low	Medium	High

If you don't give your message a priority, it goes into the lowest queue level created.

Figure 5-5 lists the possible characteristics shared by messages in a queue. Priority and network queuing have already been discussed. Message classification is a more complex concept than the other because it depends on your use of the queue and not on any of its inherent characteristics.

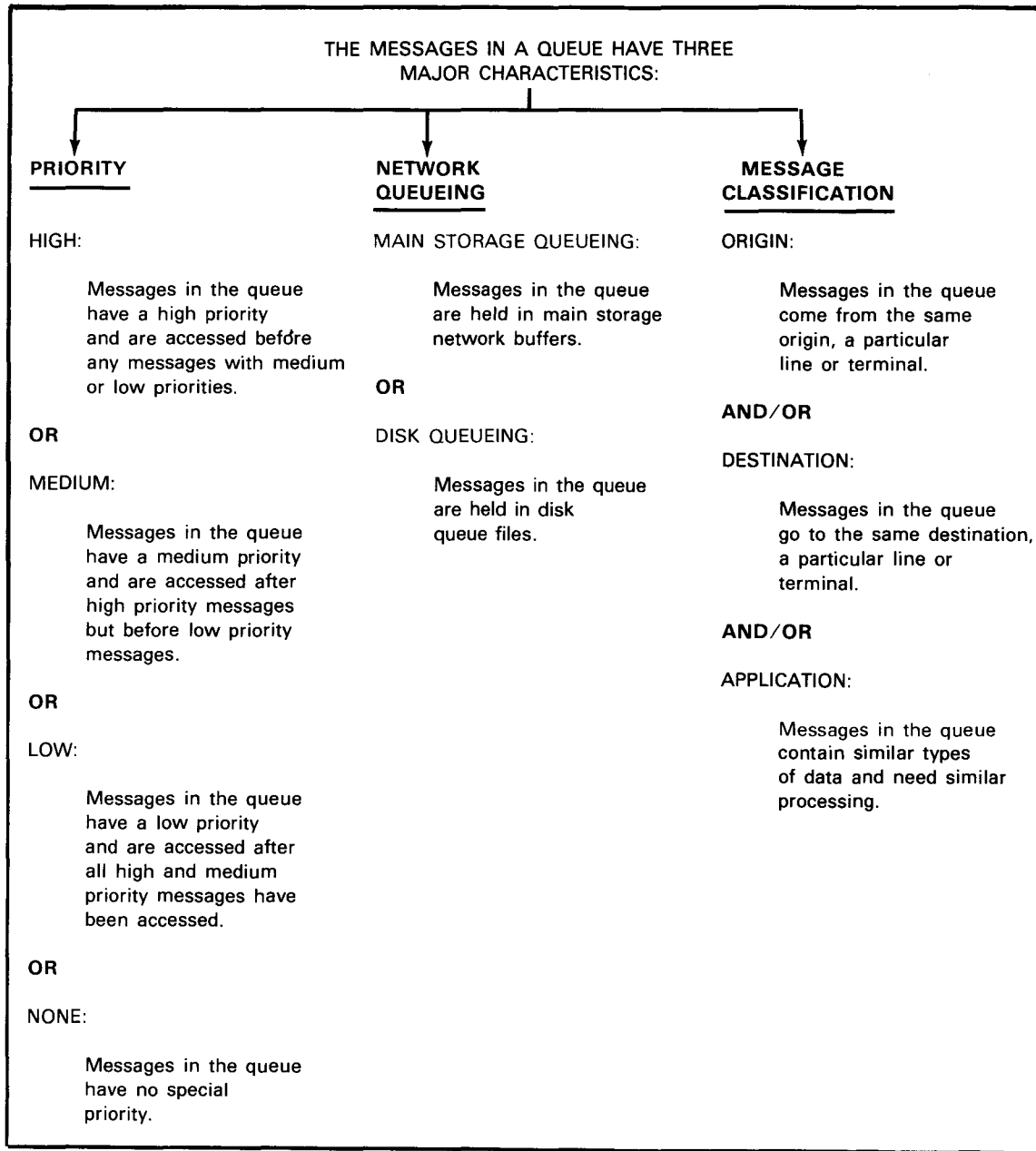
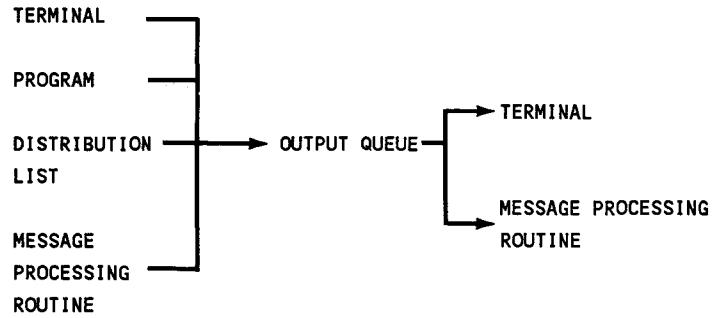


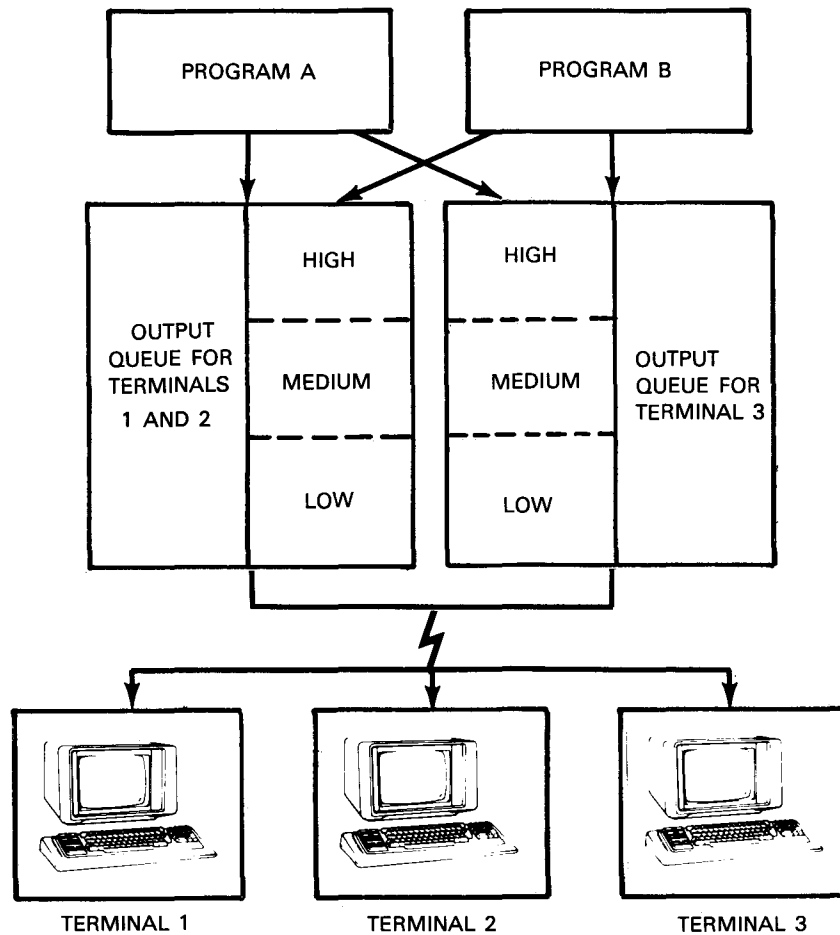
Figure 5-5. Message Characteristics

### 5.5.1. Output Queues

ICAM supports multilevel or single-level output queues. They may reside in main storage or on disk. Messages may come from one or many origins and go to one or more terminals on a line.



ICAM has just one type of output queue. As described in the general queue discussion (5.5), each terminal receiving messages must be associated with an output queue, although terminals on the same line can share an output queue.



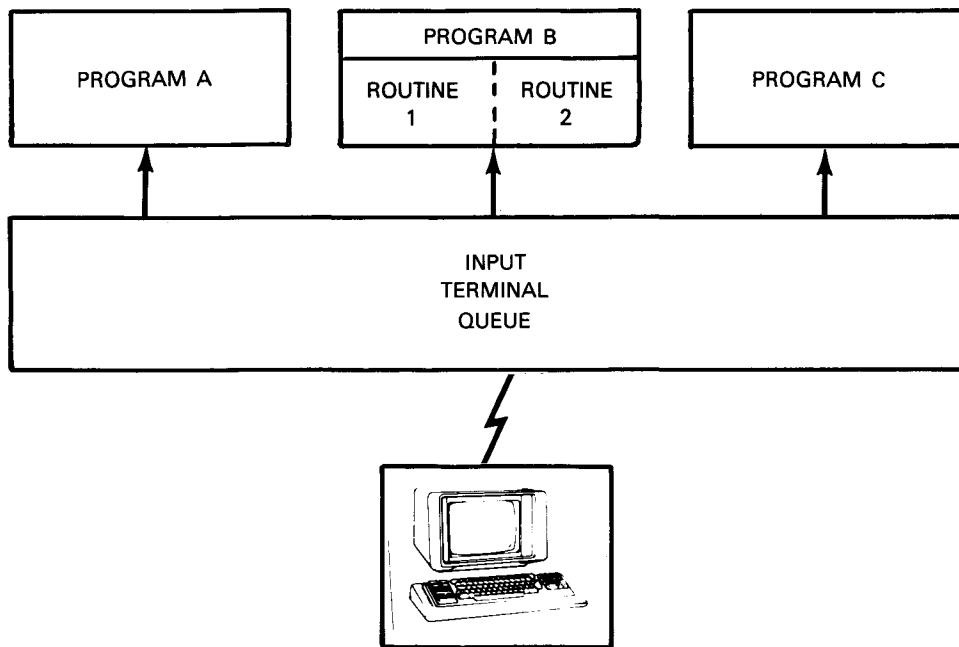
### 5.5.2. Input Terminal Queues

Input terminal queues are defined on the INPUT operand of the TERM macro. They are single-level queues and may reside in main storage or on disk. Messages come from a single terminal.

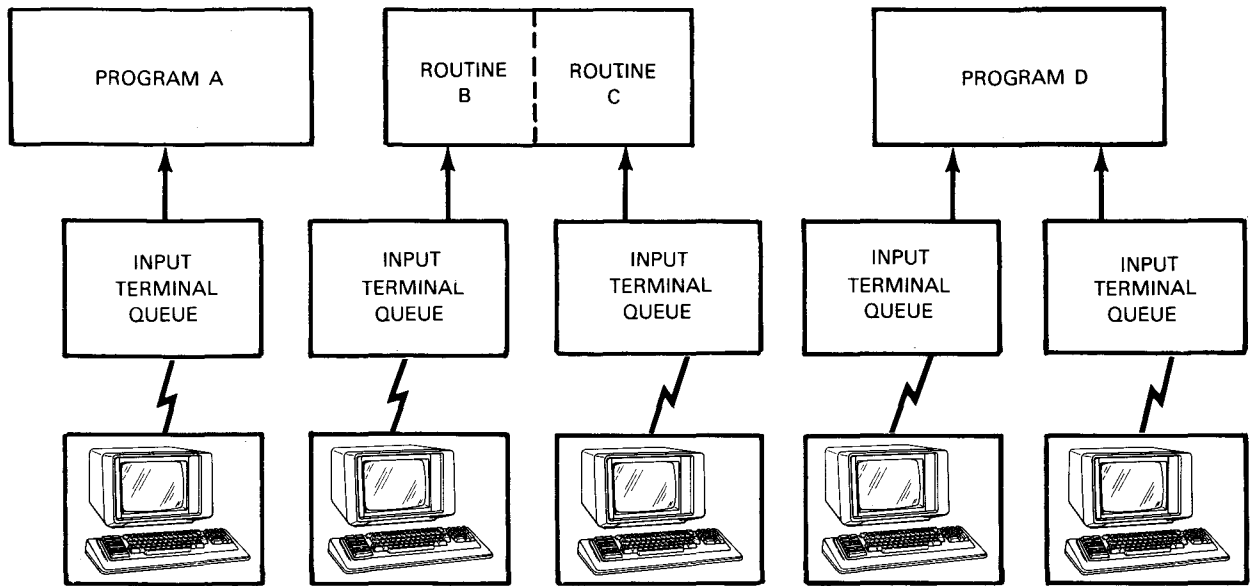


Input terminal queues are the simplest type of queue. Each input terminal queue queues messages from and relates to a single terminal. Because they are single level queues, all messages have the same priority. Messages leave a queue in the order they enter it.

Although messages in a queue come from a single terminal, depending upon which programs issue an input request, they could go to any number of destinations. Thus, it is important that you coordinate your programs carefully if several are coded to access the same input terminal queue.



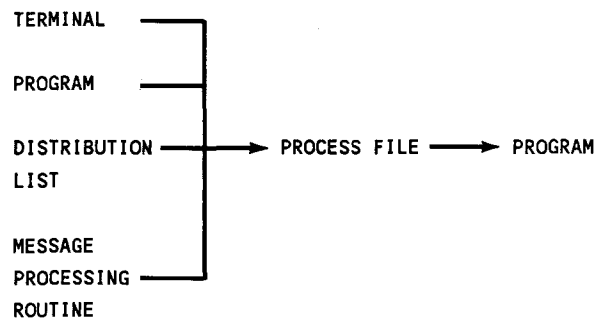
Since the first program to access a message on a queue removes it from the queue, we do not recommend more than one destination. It would be impossible to predict which program or routine will get a particular message. Instead, we suggest that all messages in an input terminal queue go to a single destination, as shown here:



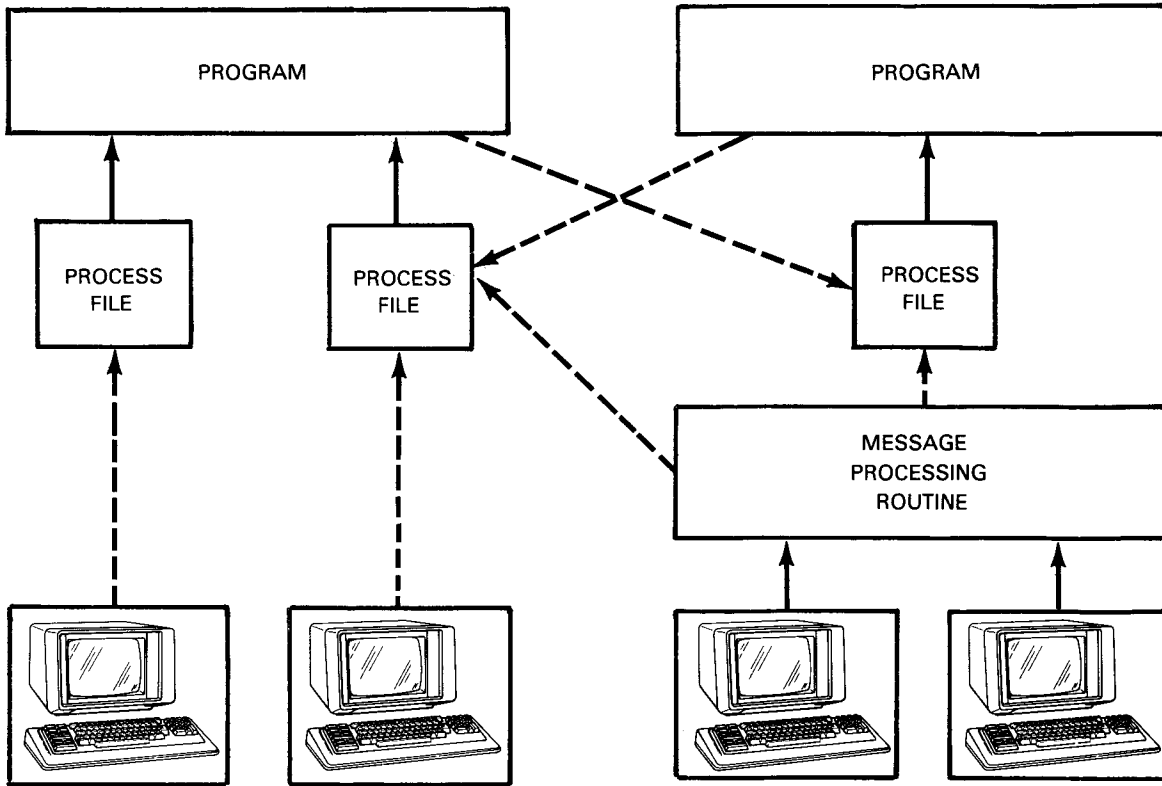
If the messages in an input terminal queue go to the same destination, chances are they are part of the same application.

### 5.5.3. Process Files

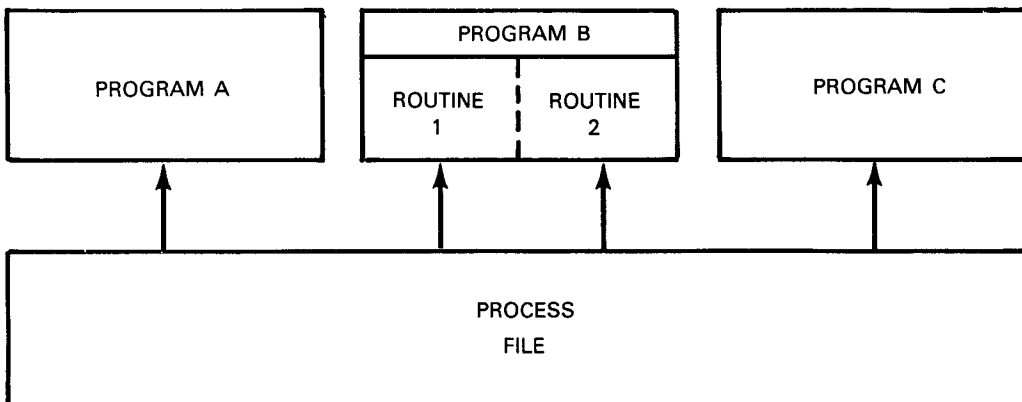
Process files support multilevel or single-level queuing. They may be located in main storage or on disk. Messages come from one or many origins, and the messages may be accessed from one or many locations.



A process file is a set of up to three queues that your program or ICAM can use to store messages until they are accessed by your program. They are probably the most complicated of the queues. Messages can come from many places, as shown here:



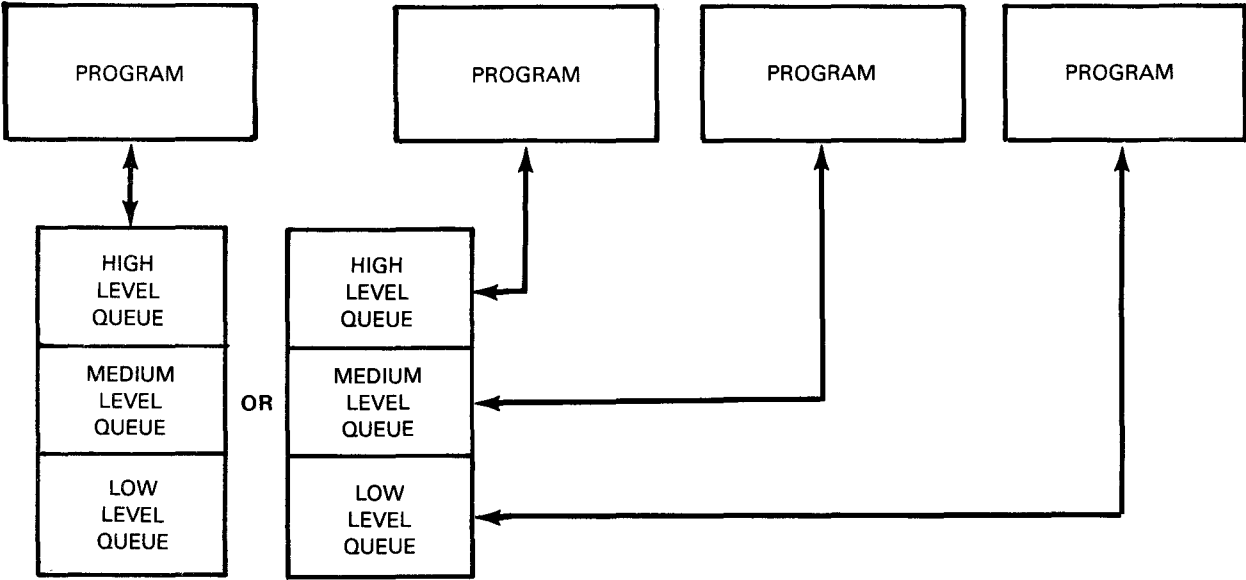
And messages in a process file can go to any number of destinations:



As with the input terminal queues, you must carefully coordinate your programs if you let more than one program access the same process file queue because it's impossible to predict which program or routine will get a particular message.

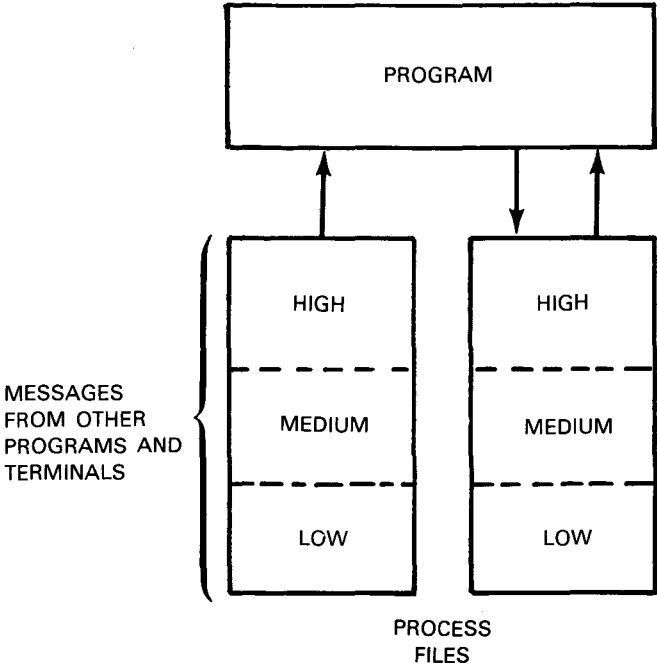
Like all queues, except the input terminal queues, you may create multiqueue process files. Unlike any other queue types, however, you use multilevel process files in one of two ways: as a single queue with two or three priority levels or as two or three separate single-level queues.





You access the levels of a process file separately when each level contains messages that are somehow different. They may come from different origins, go to different destinations, or need different processing.

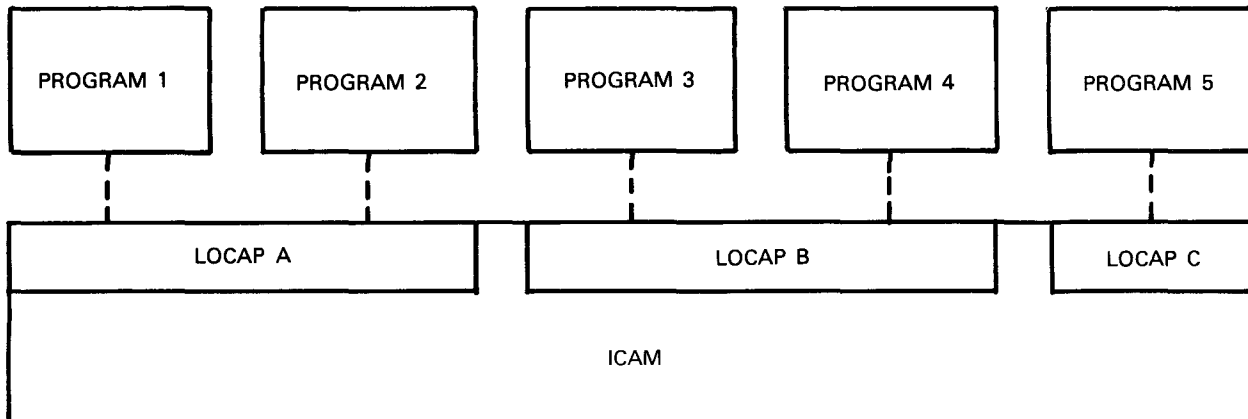
A program can write a message to a process file and later read it from the process file:



Used this way, a process file is like a memo pad. Let's say your program generates price quotes for your salesmen. Occasionally, it has system-related messages to send (for example, SYSTEM WILL SHUT DOWN AT 3:00 PM FOR MAINTENANCE) that you don't want to appear in the middle of a price quote. Your program can put these messages in a process file and, at the end of each price quote, read the messages in the process file and send them to the terminals. When used as a memo file, you should use disk queuing with the process file so main storage buffers are not tied up.

### 5.5.4. Locap Files

Programs use locap files (from local applications) to attach to a global network. At ICAM generation, you create one or more locap files and assign passwords. Any program can attach to any locap file by issuing the correct password. The result is that programs can dynamically attach or detach themselves from ICAM as needed. A locap file can support only one program at a time.



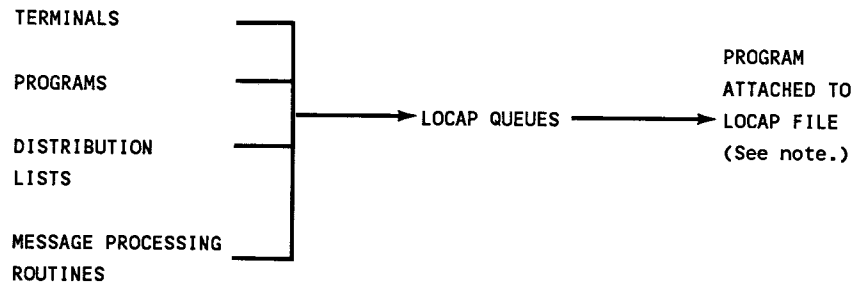
Let's say you have several communications programs. Those that need ICAM constantly can have their own locap files. Others can share locap files.

Besides being a means for a program to attach to ICAM, a locap file can also have a set of priority input queues similar to those used with a process file.

Your program may read a message from a specific queue level, or it may do a general read, in which case a message is read from the top of the highest priority queue containing messages.

Any program attached to a locap file can read messages from the locap file queues.

Messages to locap file queues come from the same sources as messages for process files:

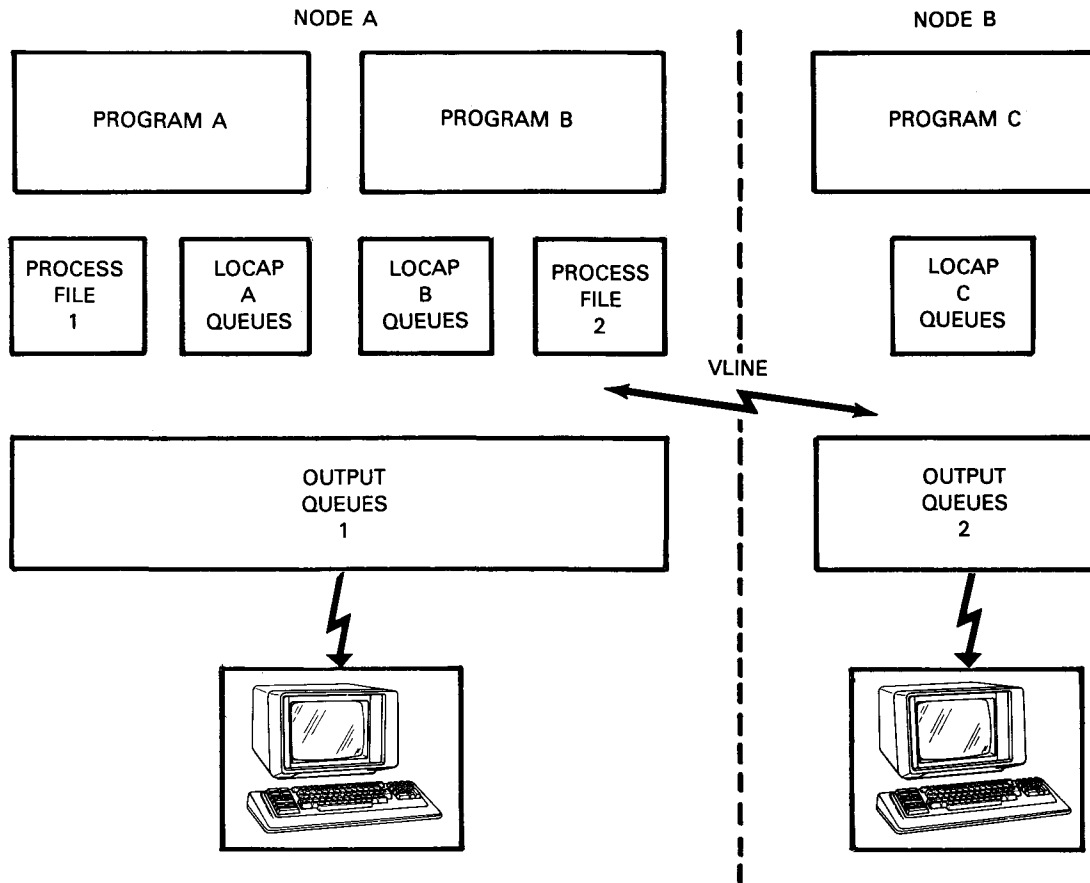


Note:

*Your program cannot place messages on the queues of the locap file to which it is connected.*

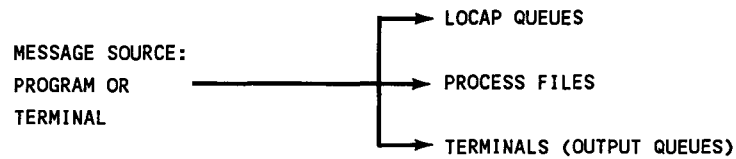
## Queue Arrangements

The most important fact of global queue arrangement is that messages can be placed on queues in any node but can be read from queues only in the local node. Let's say you have the following queue arrangement in a multinode environment.



Any of the three programs can write messages to any queue in the system except the queues of the locap file to which they're attached. Program C cannot read messages from either process file in node A. Similarly, output queues for a terminal must be in the node in which the terminal is attached. Think of it this way: you can push a message from one node to another by writing it to a queue in another node, but you can't pull it from one node to another by reading it from a queue in another node.

During network definition, you may define static sessions that specify which programs and terminals can send messages to which queues. Whether the message comes from a program or terminal, the allowable destinations are the same:



### 5.5.5. Using Queues

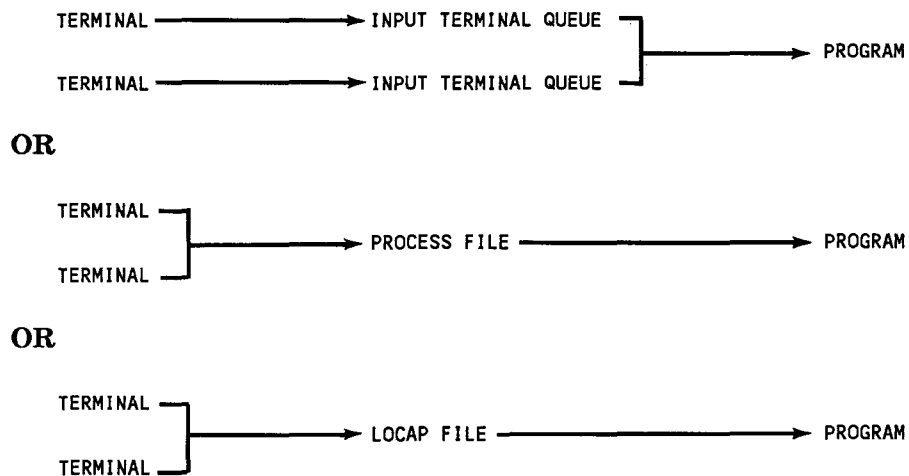
The primary point to keep in mind as you design your queue system is that it must allow messages to be sent between programs and terminals that you want to communicate with each other. After that, several other points influence the details of your queue system:

- Do messages have different priorities?
- Do you want messages from different origins to have different queues?
- Do you want line or terminal queuing?

Let's start by looking at the queue arrangements for messages coming from terminals. The big factor here is that you must decide at network definition where messages from a particular terminal are to go. If messages of a terminal are to always go to a particular program or terminal, specify that destination at network definition.

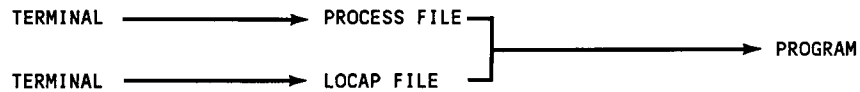
#### Obtaining Messages from Terminals

To access messages sent from a terminal and placed on queue by ICAM, your program specifies the name of an input terminal queue, process file, or locap file. Note that each terminal must have its own input terminal queue while any number of terminals can share process file or locap file queues:



## Buffers and Queues

You can specify that the input from any terminal is to be placed on any process file or locap file:



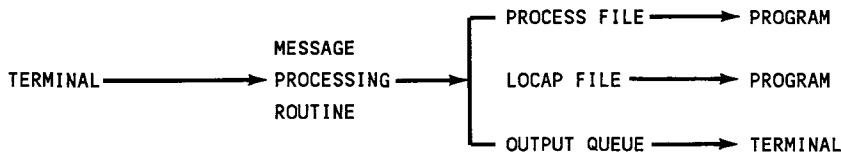
To send messages of one terminal to another terminal, specify the other terminal name as the destination. ICAM puts the messages in the correct output queue:



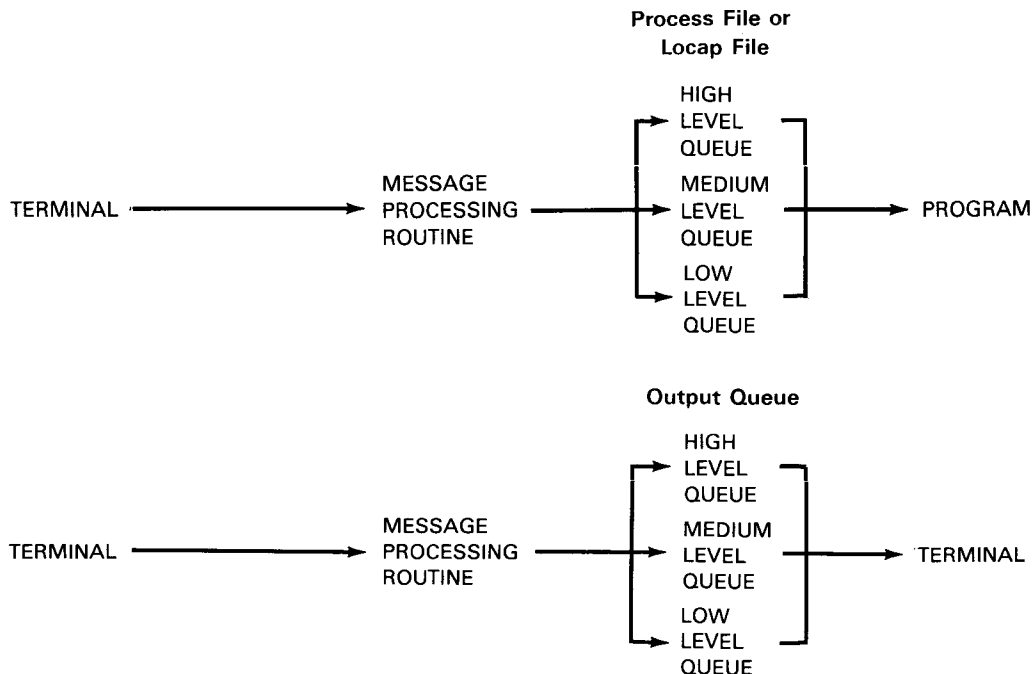
If messages of a terminal are to go to several destinations, you must:

1. specify a message processing routine as the destination (Section 6); and
2. put the message destination (input terminal queue, process file, locap file, or terminal) or its type in the header (beginning) of each message from the terminal.

When the message processing routine receives the messages, it routes them to the proper destination, based on the information in the message header. (See 6.4 for details of message processing routine message routing.)



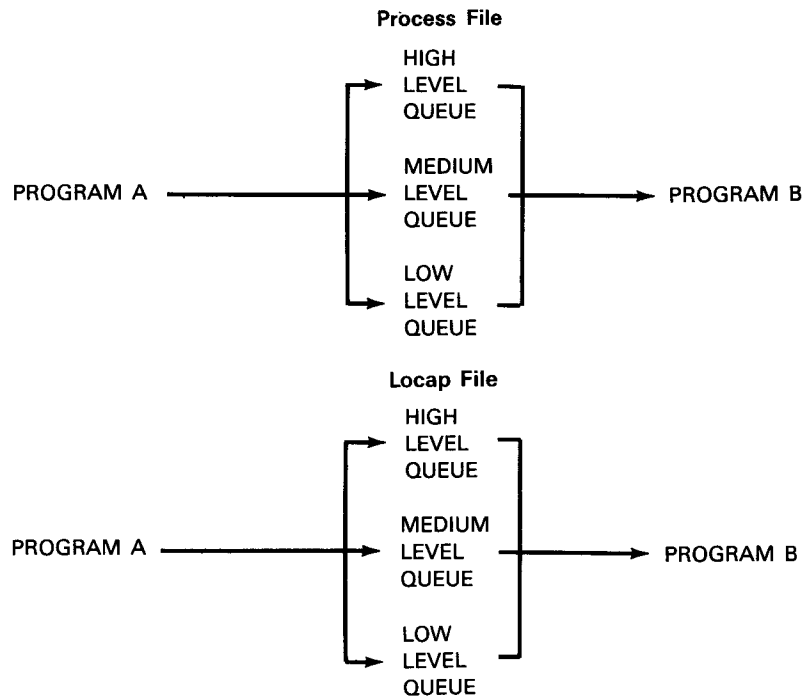
Also, use a message processing routine to send messages to different priority queues within a process file, locap file, or output queue:



## Sending Messages from Programs

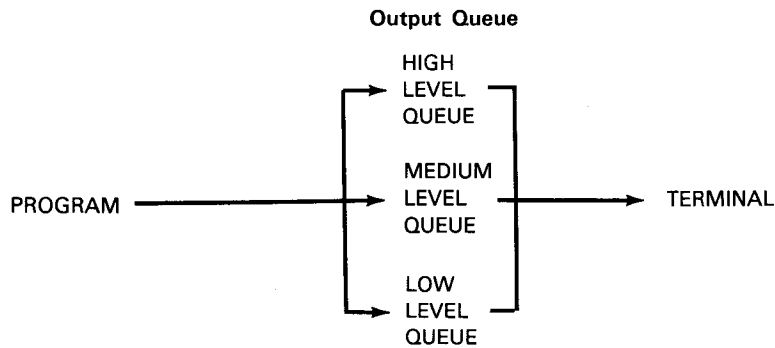
Sending a message from a program is easier to visualize than receiving a message from a terminal because a program specifies for each output message its own destination and priority. You do not need a message processing routine to route messages to different destinations and put them in different priority levels in a process file, locap file, or output queue.

To send a message to another program, a program specifies as the message destination the name of a process file or locap file used by the other program. (A program cannot place a message in an input terminal queue.) The program can also give the message a priority; if it doesn't, the message goes in the lowest queue level created for the process file or locap file.

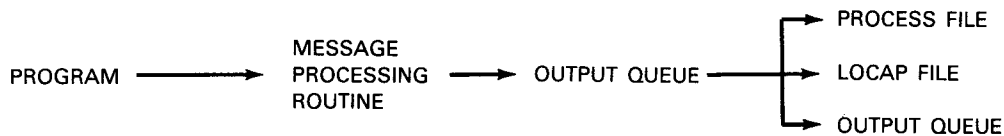


## Buffers and Queues

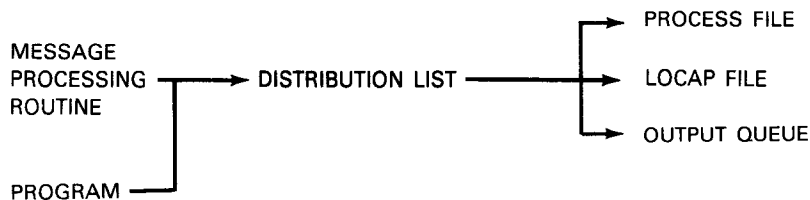
To send a message to a terminal, a program specifies the terminal name as the destination. It can also give the message a priority; if it doesn't, the message goes in the lowest queue level created for the terminal output queue. ICAM places the message in the correct output queue:



If, at network definition, you associate the terminal with a message processing routine, the message is processed by the message processing routine that queues it to the appropriate output queue.



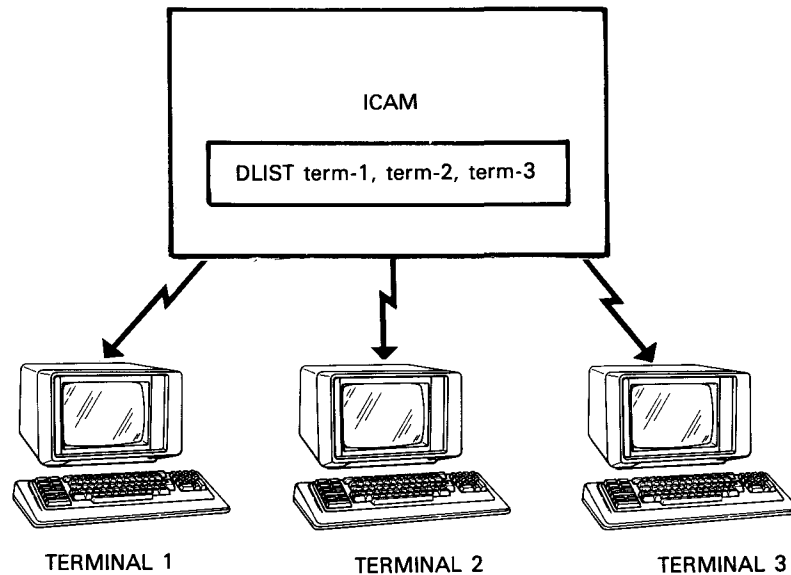
Distribution lists are aids for sending messages; they are lists of two or more destinations: process files, other distribution lists, locap files, or terminals to which a single message is to be sent. Use them to send a message to multiple destinations. For example, you might want every program or all terminals in sales offices to receive a particular message. With a distribution list, you send the message once and it goes to each destination listed. Both programs and message processing routines use distribution lists.





## Distribution Lists

When you write a communications program in BAL or COBOL, you can use a distribution list (DLIST) to send the same message to two or more destinations. By using the DLIST, you can save both processing time and coding instructions.



When you write your program in basic assembly language, you can specify your distribution list at either ICAM network definition time or directly in your communications program as follows:

- *Network definition* - When many of your communications programs use the same network, you should specify the DLIST in your network definition. Once it is defined, it is permanent for the duration of the network. When you write your communications program, you need only refer to the label, thus saving coding instructions and also processing time.
- *Communications program* - When your destinations are frequently changing or if you forgot to include some at ICAM generation time, ICAM provides you with this alternate method. Regardless of when you specify the DLIST, the result is the same - you save coding and processing time. When you write your communications program in COBOL, the distribution lists are specified at ICAM generation time. You refer to these destinations in the COBOL message control system module and in the communication section of your COBOL program. For more information about the DLIST macro, see the *ICAM Standard MCP (STDMCP) Interface Programming Guide* (UP-8550).
- *Utilities* - When you write your program in RPG II, the distribution lists are defined on the FILE DESCRIPTION and OUTPUT SPECIFICATION forms. However, they are not referred to as DLIST but as output devices. See the *RPG II Programming Guide* (UP-8067) for more information.

## 5.6. Dynamic Buffer Pool Expansion

The ICAM dynamic buffer pool expansion feature lets you use the OS/3 dynamic buffer management facility to expand ICAM buffer pools as needed. The OS/3 dynamic buffer pool management facility dynamically allocates extra main storage to system routines as they need it. For some ICAM functions this facility is required; for others it is optional. By acquiring main storage dynamically, ICAM can continue processing after previously allocated buffers are filled.

The ICAM functions that use dynamic buffer management are:

- Dynamic buffer pool expansion
- Dynamic session establishment
- ICAM trace facility (ITF)
- Public data network support

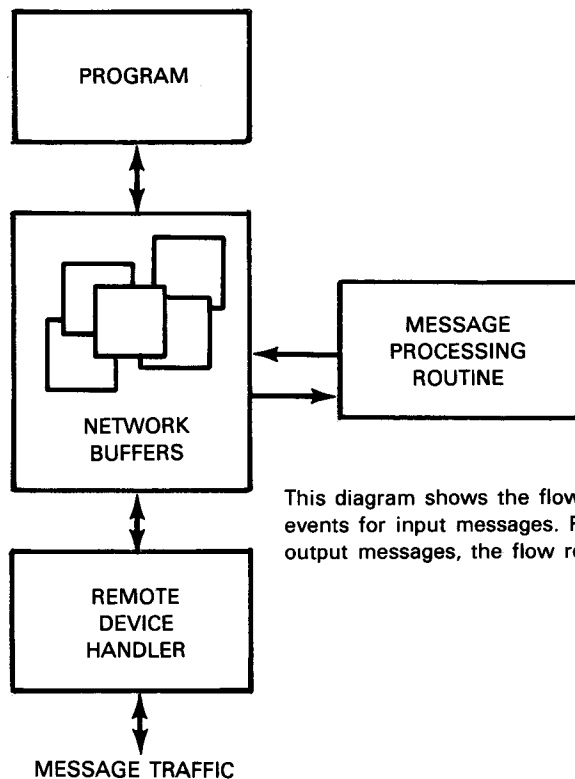
The OS/3 dynamic buffer pool expansion feature is automatically included unless you specify `BPOOLEXP=NO` in the MCP portion of the `COMMCT` section at OS/3 system generation. You can control the amount buffer pools are expanded by specifying a percentage in the `EXPFACT` operand of the `ICAM BUFFERS` macro.

# Section 6

## Message Processing Procedure Specification (MPPS)

### 6.1. General

The MPPS macroinstructions allow you to write message processing routines that perform limited processing. They process input messages after they go into the network buffers but before they go to your program. Output messages are processed after they go into the network buffers but before they go to the remote device handlers. MPPS is fully described in the *ICAM Reference Manual (UP-9749)*.



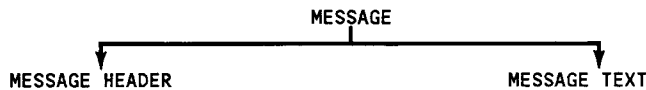
This diagram shows the flow of events for input messages. For output messages, the flow reverses.

Even though message processing routines are optional, you should use them for the following reasons:

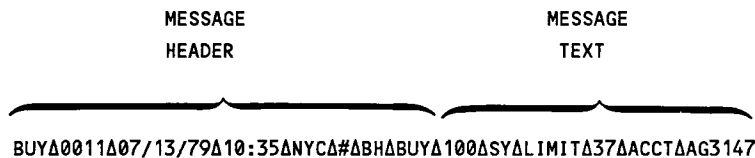
1. They allow ICAM to do some preprocessing and postprocessing of messages, thereby relieving your programs of these tasks. Time and date stamps may be put in message headers, and certain fields in the message header may be checked for validity. (See 6.2 for a complete definition of message header.)
2. They give you some control over transmission errors. Normally, ICAM handles all transmission errors without informing your programs or terminal operators that the errors occurred. You can use message processing routines to direct the disposition of messages associated with transmission errors and to inform your programs and terminal operators of the errors.
3. The message processing routines can route messages to their destinations, based on the information placed in the message header by your program or terminal operator.

## 6.2. Message Header Examination and Manipulation

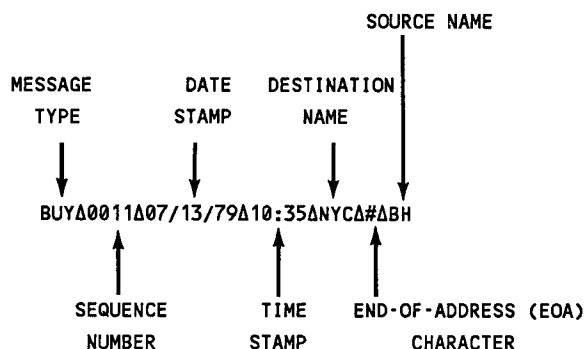
The message processing routines can examine and manipulate message headers. Expanding a message, we break it into two parts:



The message header is like the heading of a letter. It contains whatever information you want presented for all messages in a standard format: message source, message destination, message type, date, time, and so forth. The message text contains the information being transmitted and is like the body of a letter. A typical message for a stock brokerage might look like this:



The message header contains the following data:



All data in the message header is optional, as in the message header itself. Although message processing routines can add certain data to the message headers, such as the date and time, you're responsible for their format. The message header is an integral part of the message you create.

Message processing routines can look for or insert the following kinds of information:

- *Time and date stamp* - If you request it, message processing routines insert time and date stamps in the message headers of input and output messages:

```
time and date stamp    ΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔΔBUYA0011ΔNYCA#ΔBH
time and date stamp    BUYA0011Δ13:27Δ07/13/79ΔNYCA#ΔBH
```

The date stamp is either Gregorian format (usually mm/dd/yy or dd/mm/yy) or Julian format (yyddd). The date is also available to standard interface programs through the file tables or to any program through standard commands in the language in which it is written.

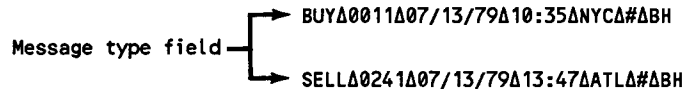
The time stamp format is hh:mm, with values ranging from 00:00 to 23:59. The time is also available to standard interface programs in the file tables or to any program through standard commands in the language in which it is written. (For example, GETIME in the basic assembly language or ACCEPT identifier FROM TIME in COBOL.)

For the time or date of each message to appear in the journal, the time and date stamp must be used.

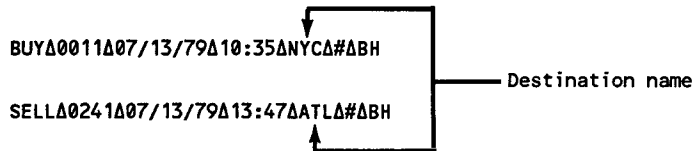
- *Message sequence* - For input, the message processing routines verify the sequence number placed in the message header by the terminal operator. If the sequence number is incorrect, a bad sequence flag is set.

For output, the message processing routines place a sequence number in the message header. The sequence numbers are incremented by terminal. In other words, the first message to a particular terminal is message 1 and the nth message to the same terminal is message n, regardless of how many messages go to other terminals.

- **Message type** - The message processing routines separate messages into two or more types to allow different processing on each type. You set up one or more fields in the message header containing the message type:



- **Routing information** - The message processing routines route input messages to a queue named in destination field:



- **Message source** - The message processing routines verify that the source name put in the message header is the correct source name. If it isn't, the invalid source flag is set.

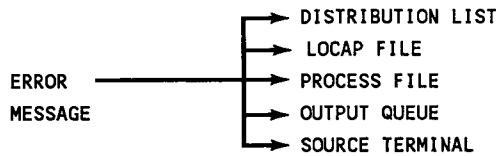
### 6.3. Error Recovery

You can use the message processing routines to recover from the following errors:

- **Transmission errors**
  - Line error
  - Terminal or auxiliary device error
- **Journal utility errors**
  - Journal staging area threshold value reached
  - Journal record staging error
- **Message header errors**
  - Invalid destination name
  - Incorrect but valid source name
  - No destination name
  - Incorrect and invalid source name

- Sequence number errors
  - Sequence number of incoming message was not placed in message header prefix
  - Sequence number of outgoing message was not placed in message header prefix
  - Incorrect sequence number in incoming message header
  - End of header
  - No room for insertion

Your options for recovering from any of these errors are the same. First, the message processing routine can send error messages to any or all of the following:

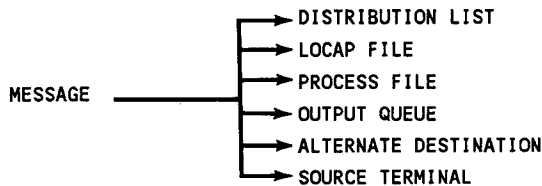


*Note:* Error messages cannot go to the system console.

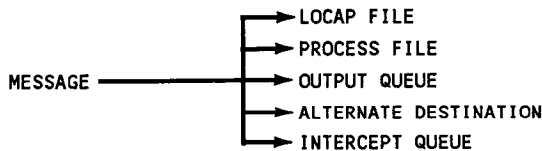
The error messages can say anything you want and go to as many destinations as needed. For example, you might send a message to a program when a terminal or auxiliary device error prevents the program's message from going to a terminal:

TERMINAL AUXILIARY DEVICE ERROR. MESSAGE FOLLOWS.

In addition to sending an error message, the message processing routine can either cancel the message or route it to a destination. If you choose to route the message, an incoming message goes to one of the following destinations:



An outgoing message goes to one of the following destinations:



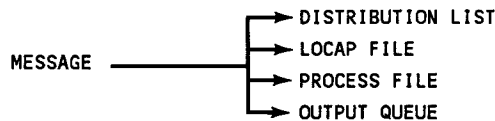
## 6.4. Message Routing

The message processing routines can use either the message type or destination field in a message header to route messages to their destination. Depending on your needs, the field used doesn't necessarily matter.

The message processing routines use the destination field for routing only with incoming messages. To use it, a terminal operator writes a message header with one or more destinations listed:

```
BUYΔ0011Δ07/13/79Δ10:35ΔNYCΔWASHΔATLΔLAΔPHILΔ#ΔBH  
                ↑↑↑↑↑  
                _____ DESTINATIONS
```

The terminal operator can list as destinations distribution lists and any type of queue except terminal input queues and intercept queues:



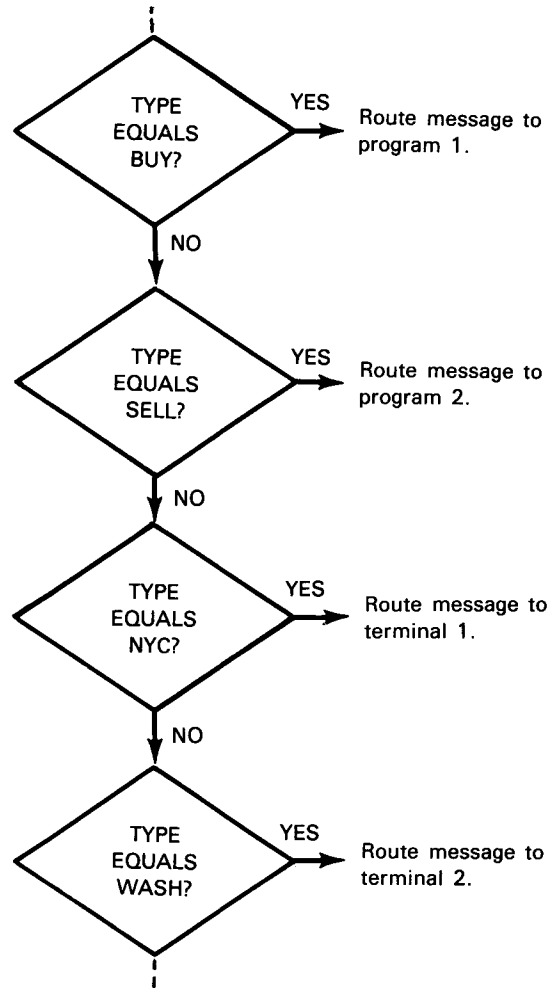
If the message type determines the destination, then you can use the destination name to route messages by their type:

```
0011Δ07/13/79Δ10:35ΔBUYΔ#ΔBH  
                ↑  
                DESTINATION
```

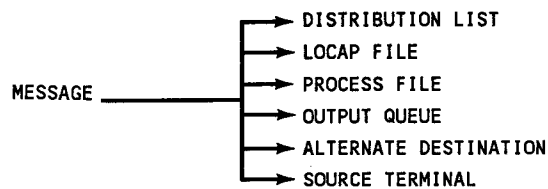
BUY must be the name of an output queue, process file, locap file, or distribution list.

Incoming messages can be routed by the message type. The end effect is the same as with routing on a destination name. Instead of naming a destination, the message type controls branching within a message processing routine:





The message processing routine can send the message to any of these destinations:



## 6.5. Miscellaneous Functions

Message processing routines can perform several additional functions:

- They can be used to journal messages.
- They can make the remote device handlers try to retransmit output messages when an error occurs during the initial transmission.
- They can limit the number of messages received from a single terminal when the terminal is polled.



# Section 7

## DCA and DDP

### 7.1. Introduction to DCA

Distributed communications architecture (DCA) provides a uniform structure for implementing communications networks. The key to providing this structure for current and future enhancements is the concept of distributed intelligence. As minicomputers and intelligent terminals become available, we can introduce intelligence into the communications network. This intelligence is often in the form of a front-end processor. Front-end processors relieve the host processor of some of the communications load. Distributed communications architecture defines the concepts for a total system approach to data communications.

Specifically, distributed communications architecture:

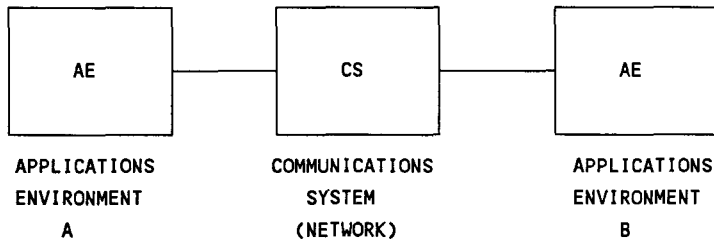
- Provides the logical structure of paired layers of processing components
- Provides a set of protocols and interfaces for exchanging commands and data
- Establishes a common access method for the logical interface to the communications system
- Provides a common network control capability that is part of the network, but separate from the host processor

In OS/3, the physical protocol is a bit serial line discipline called universal data link control (UDLC). UDLC is a Unisys standard protocol. This physical protocol is supported by a special single line communications adapter (F2798-00). The SLCA is a down-line-loaded microprocessor-based communications adapter.

The common access method to UDLC lines in OS/3 System 80 systems is an enhanced version of ICAM. OS 1100 uses Communications Management System (CMS) for 1100 Series systems.

## 7.2. DCA Concepts

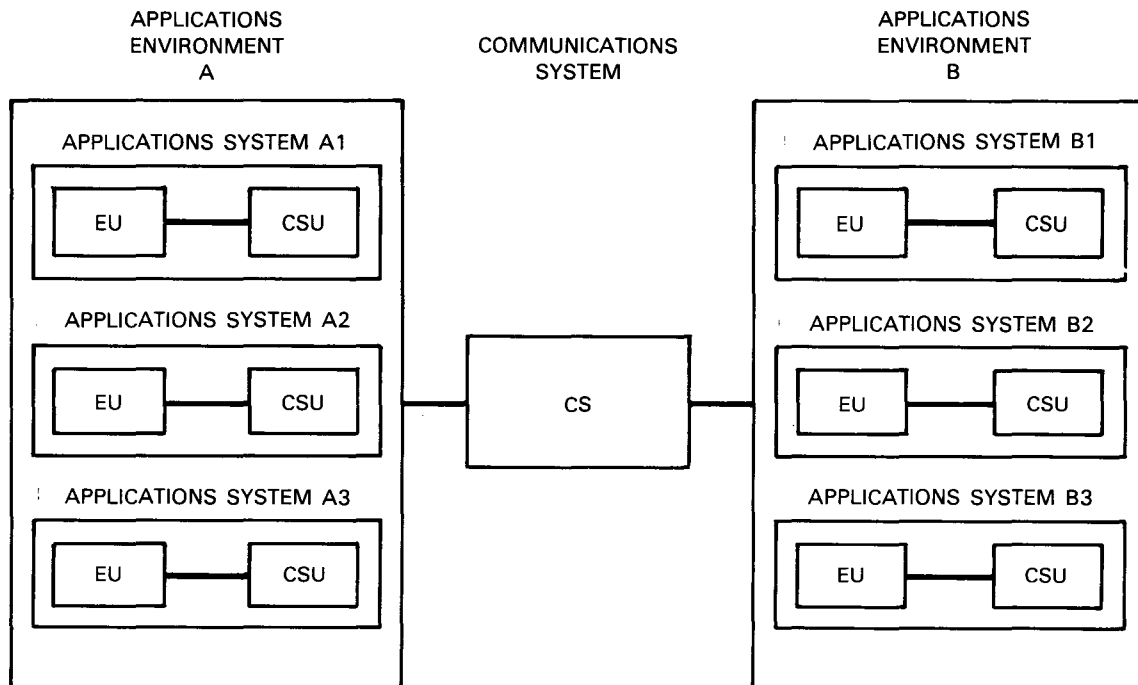
In a total systems environment, distributed communications architecture separates the communications system from the applications environment. This makes the host processor independent from the communications network and permits connection of different kinds of processors and operating systems. This relationship is shown as follows:



An applications environment may contain:

- End users (EUs) - End users are user programs and terminals.
- Communications system users (CSUs) - Communications system users are programs written for the user, such as IMS or interactive services.

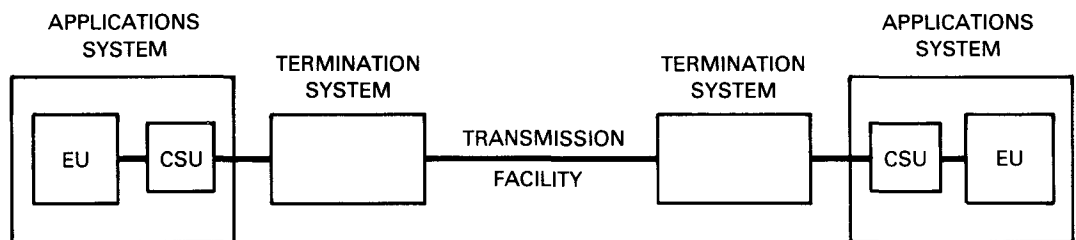
These are shown in the following illustration.



A communications system usually consists of a termination system (TS) and a transport network (TN).

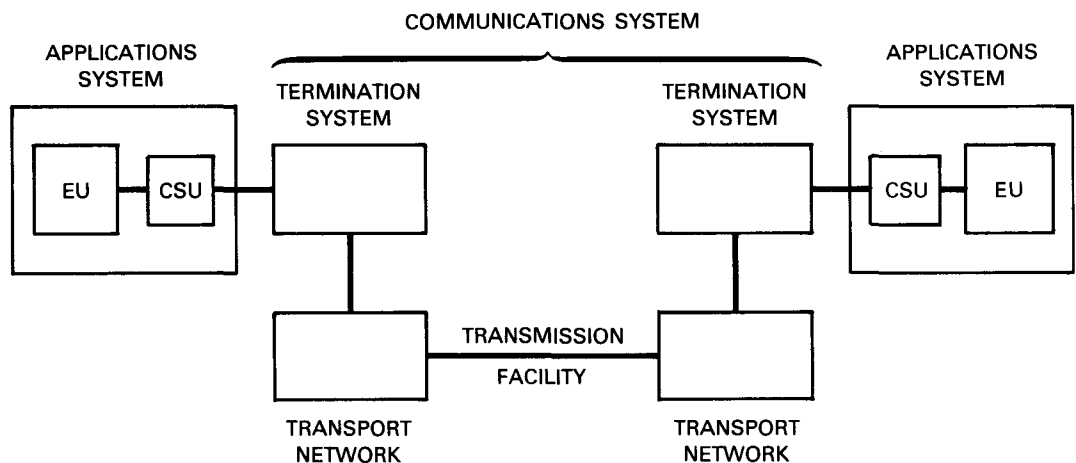
A termination system acts like a bridge between an applications system and a communications system. When a transport network is used, it provides the logical ports into the transport network through which the communications system user establishes logical connections called sessions.

Not all communications systems need a transport network. Such is the case in a homogeneous communications system where two processors use OS/3; for example, a System 80 processor linked to another System 80 processor. The following illustration shows a termination system that doesn't require a transport network.



A transport network transfers units of data between paired termination systems. It regulates the flow of data, selects the circuits, and segments messages into transmission buffers.

The following illustration shows a simplified communications system with a transport network.



Transport networks are required when different kinds of host processors using different operating systems are connected together in a heterogeneous environment. Figure 7-1 shows a communications system environment with a transport network. All of the layers of control are shown. The figure also shows a session path that might be established between end users. No attempt is made to explain the layers of control within the transport network or termination system, because they are completely transparent to the end user. Detailed information regarding DCA is contained in the *Distributed Communications Architecture System Description* (UP-8469).

In distributed communications architecture, hardware and software are configured so that the communications system achieves independence from the applications system. The termination system provided by ICAM permits this independence.

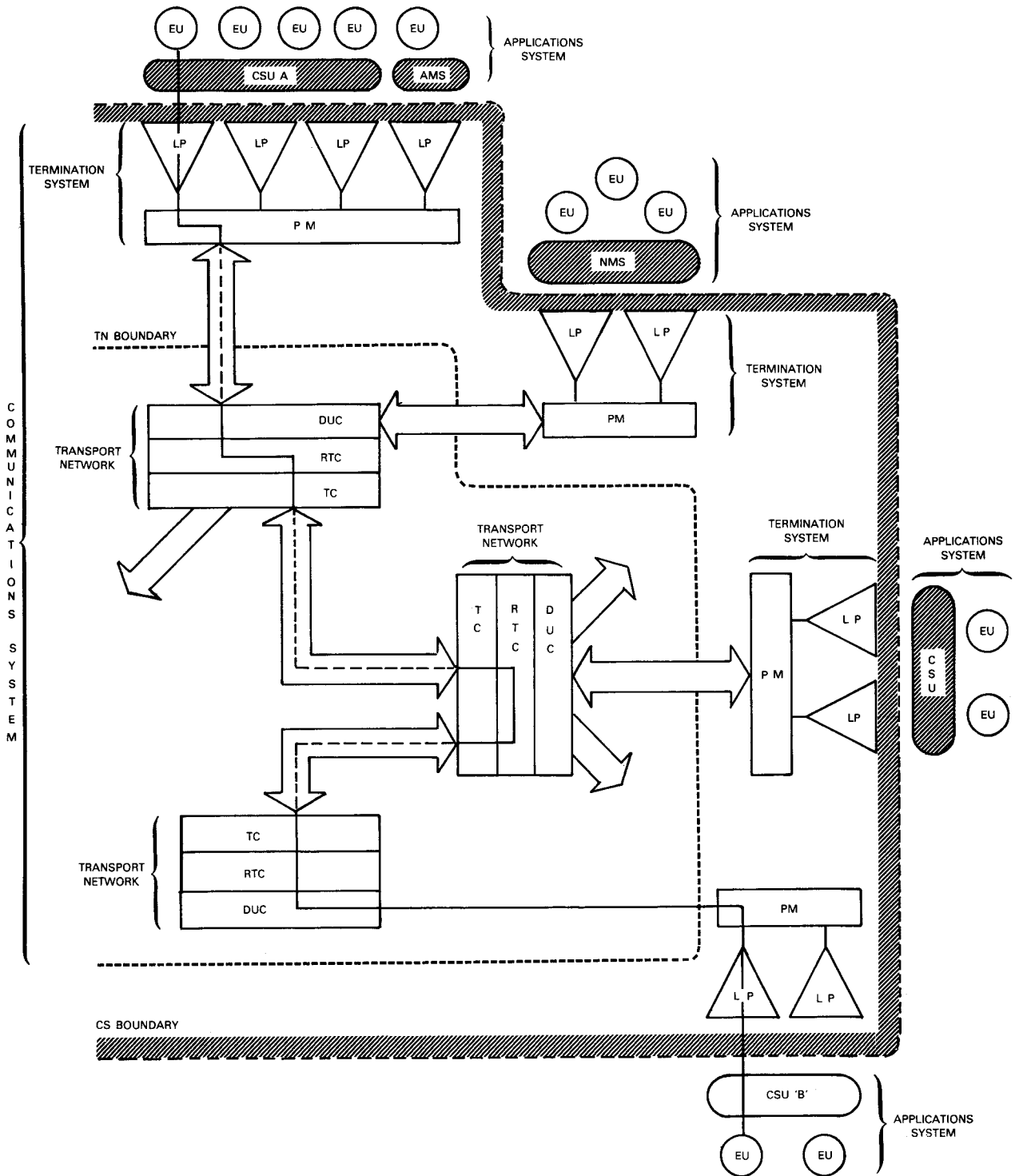


Figure 7-1. Example of Communications System and Session Path

### 7.3. Single-Node and Multinode Global Networks

Global networks can be used in a single computer environment and in an environment where multiple computers are present. In a multicomputer environment, ICAM provides DCA support.

- *Single-node global network* - A single-node global network functions in a single computer. You can use a global network to pass data from one local program to another local program, and between any local program and the terminals and process files defined in the global network.

In addition, you can:

- Connect workstations directly to interactive services (a system program that handles workstation input and output on a data management level)
- Define any ICAM supported terminal to make it function like a workstation with interactive services

When you define a global network that uses interactive services, you must specify that the demand mode interface (DMI) is used. See the *ICAM Operations Guide* (UP-9745) for details on how to do this.

Figure 7-2 illustrates a typical single-node global network.

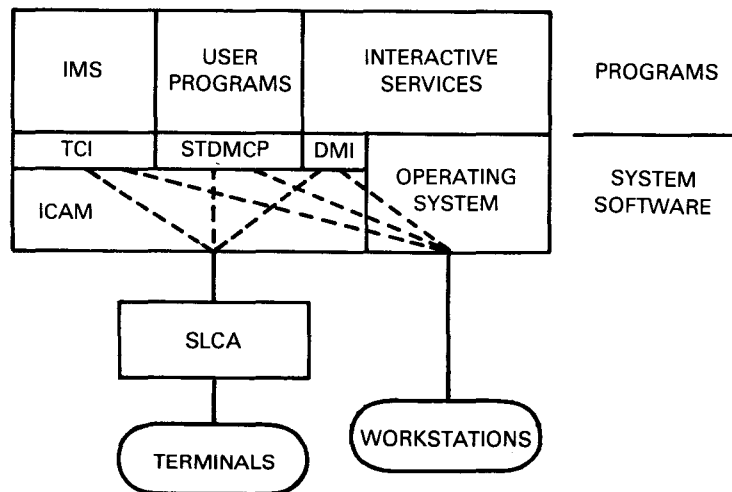


Figure 7-2. Typical Host Processor Using a Single-Node Global Network



- *Multinode global network* - A multinode global network supports more than one host processor.

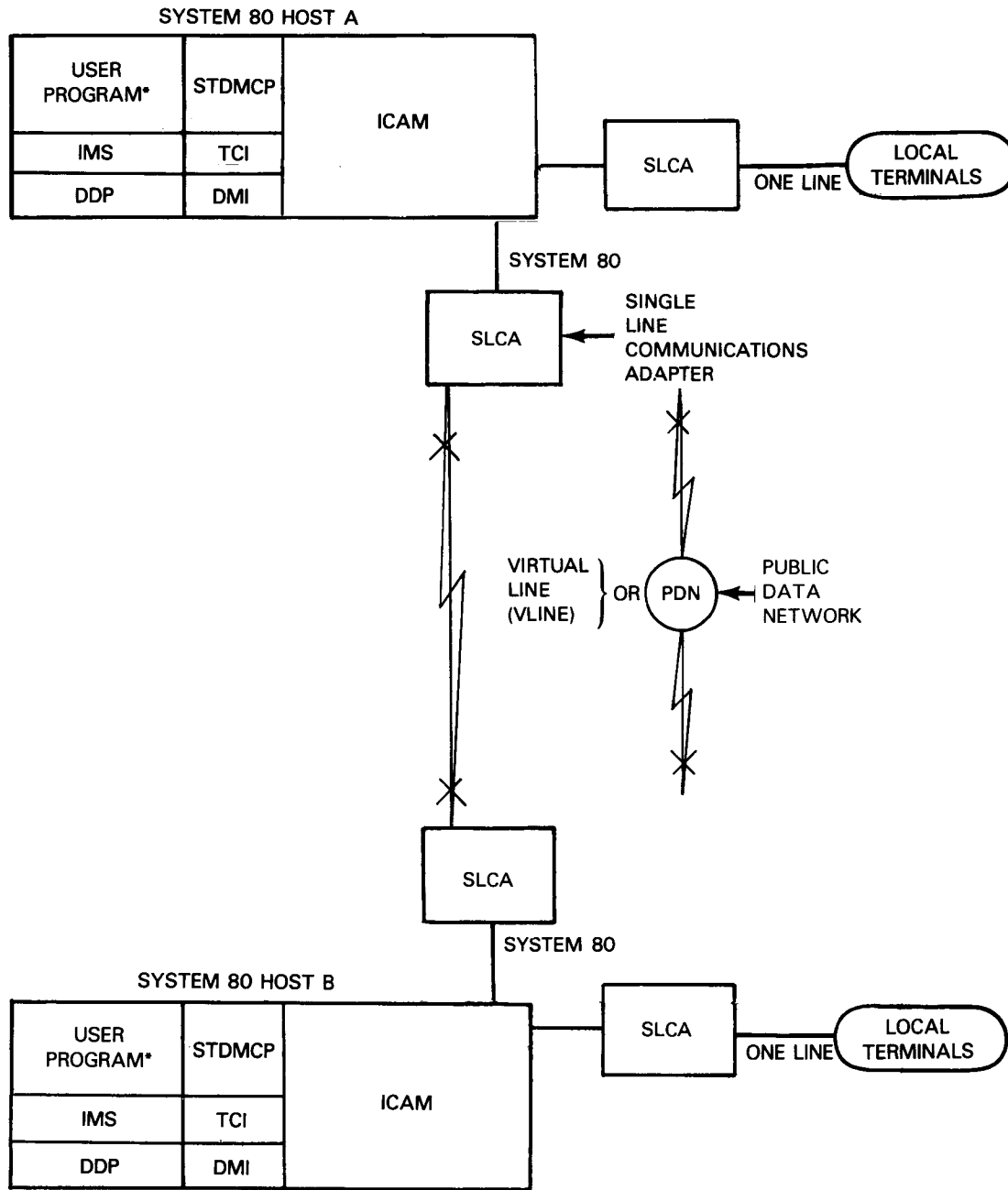
A DCA multinode network permits local-user-program-to-remote-user-program sessions as well as local-terminal and local-user-program-to-remote-terminal sessions. Terminals can also communicate with local or remote interactive services, with local or remote IMS, and can use the distributed data processing (DDP) command language to transfer files and records from one computer node to another. DCA networks also support public data networks.

To create a DCA multinode environment, define a DCA global network in each computer node. In each global DCA network definition, define the local and the remote resources that you will use. Your programs must be written in assembly language only. The COBOL communications control system (CMCS) is not supported in a DCA network because it requires static sessions.

VLINEs that connect the computer nodes use asynchronous balanced mode (ABM) universal data link control (UDLC) protocol.

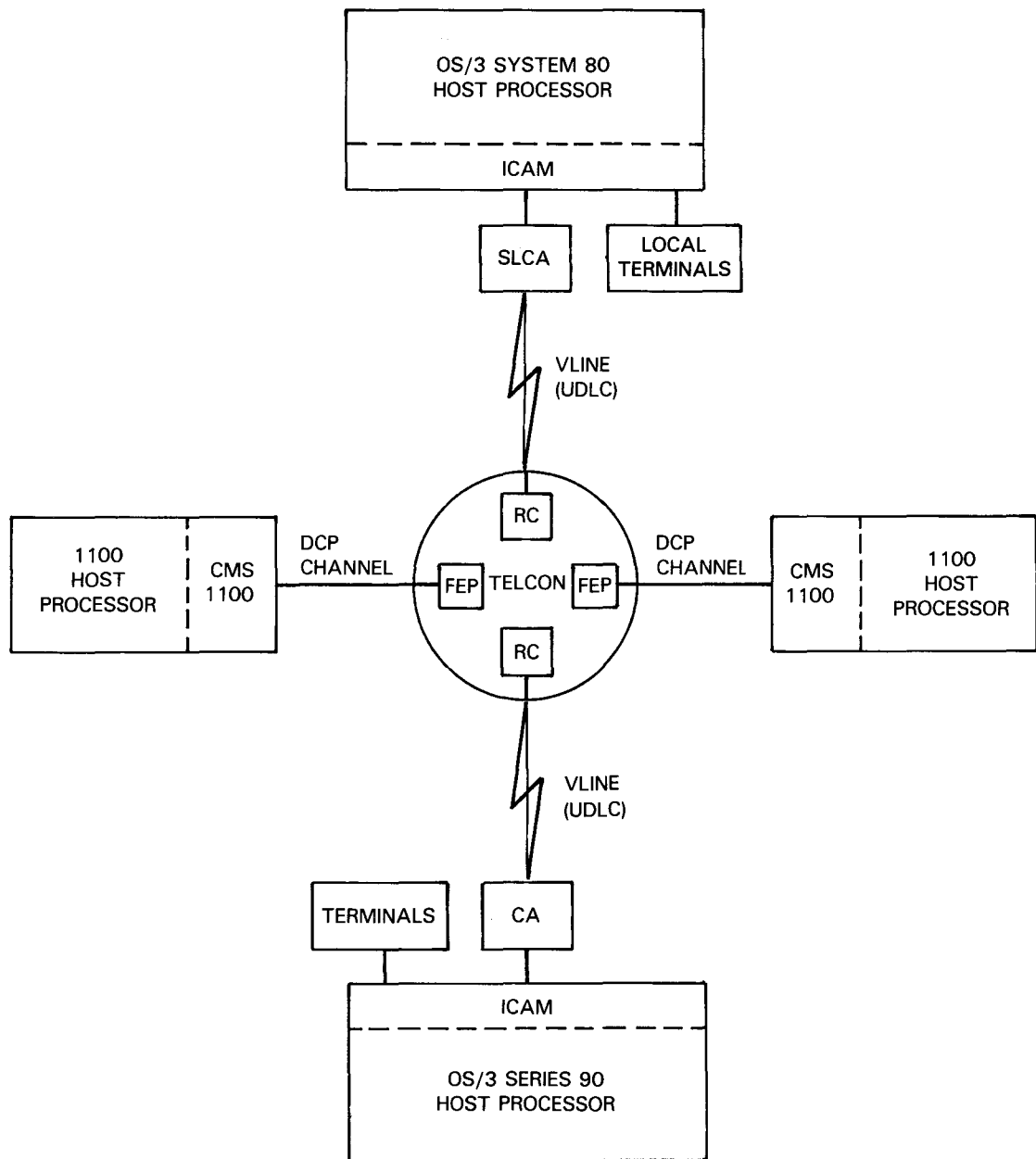
Figure 7-3 illustrates a multinode global network.

As shown in Figure 7-4, a DCA network may include a distributed communications processor (DCP) as an intermediate or remote node. If a distributed communications processor is included in the network configuration, Telcon software for the DCP must be generated on a Series 1100 system. (Refer to the *Series 1100 Telcon Installation Guide* (UP-9956) and the *Series 1100 Telcon Primary Mode Operator Reference* (UP-9256).) The Telcon software can then be transferred to the OS/3 system using utilities for subsequent cross-channel or downline loading. (Refer to the *ICAM Operations Guide* (UP-9745) and the *ICAM Utilities Programming Guide* (UP-9748).)



\* User-written programs use the ICAM standard interface (STDMCP). IMS uses the transaction control interface (TCI). Distributed data processing (DDP) requires the demand mode interface (DMI and DCA).

Figure 7-3. Multinode Global Network



**Legend**

- CA            Communications adapter
- FEP         Front-end processor
- RC          Remote concentrator
- SLCA        Single line communications adapter

**Figure 7-4. Multinode DCA Global Network Using Telcon**

## 7.4. ICAM Support for Distributed Data Processing

ICAM supports four distributed data processing (DDP) facilities:

- DDP file access
- DDP transfer facility
- IMS-DDP transaction facility
- OS/3 to UNIX file transfer facility

The DDP file access enables your application programs to access and process files residing on remote systems. It also enables your application programs to communicate with application programs in remote systems.

DDP file access provides two functions:

- Remote file processing between systems using OS/3
- Program-to-program communications

Remote file processing enables your application programs to access disk data files located on other OS/3 systems in the DDP network. This is done by adding a host-identifier parameter to the device assignment (// DVC) job control statement. When the local applications program is executed, the host-identifier parameter tells OS/3 where the file is located in the DDP network. OS/3 to UNIX file transfer does not support remote file processing.

Remotely accessed files can be cataloged in multiple host with the file residing in only one of them. This permits access from several computers and is advantageous in applications, such as an inventory control, where a master file is constantly being updated, processed, and maintained, and it is not desirable to maintain separate copies at each site.

Program-to-program communications enables you to write basic assembly language (BAL) programs to communicate with similar programs in remote systems. Communications between these programs is called a conversation, and conversing programs can transfer data from one host to another host, elaborate on the data, and then send it back to the original host if necessary. OS/3 to UNIX file transfer does not support program-to-program communications.

A program that initiates a conversation is called the primary program and the initiated program in the remote host is called the surrogate program. Primary and surrogate programs can reverse roles as many times as desired, and up to 255 surrogate programs can be initiated by a single primary program.

Conversations carried on between programs may be either simple or complex. A simple conversation is one that takes place between two programs in one direction only, and the primary/surrogate relationship never changes. For example, a primary program in system A activates a surrogate program in system B and provides it with data to be processed. The surrogate program performs the required processing, e.g., updating files, but it does not return the processed data to the primary program in system A because it is the surrogate program.

A complex conversation allows communicating programs to reverse the primary/surrogate relationship and also allows more than two programs to be involved in the conversation. For example, a primary program in system A sends data for processing to system B. System A then assumes surrogate status. After processing the data, system B assumes primary status and sends the results back to the program in system A.

Both simple and complex conversations require the same basic functions: open communications to the remote program, transfer data, and terminate the conversation when it is no longer required. Complex conversations require additional functions to allow the surrogate program to respond to the primary program and to accomplish role reversal when necessary. Unisys provides special DDP macroinstructions to implement all of these functions.

The DDP transfer facility enables you to distribute your jobs and files among multiple, separately-located computer systems.

Historically, remote batch and job entry systems were limited to a master computer at a central site providing job facilities for slave computers physically located at a central site or connected to it by communications lines. Although this mode of operation is supported, the DDP transfer facility extends this interaction in both directions, with the communicating hosts recognizing each other as peers.

The DDP transfer facility allows you to view each host in your DDP network as an available resource for scheduling and executing work. Using simple commands, you can distribute jobs and transfer files within the DDP network without concern for the intricacies of hardware systems, software systems, and communications protocols. OS/3 to UNIX file transfer does not support job distribution.

So long as sufficient resources exist at a site, the DDP transfer facility allows you to:

- Create a file on any host in the DDP network
- Copy a file from any host in the DDP network to another host
- Remove a file from a host
- Send a job control stream to a host
- Run a job control stream on a host
- Terminate a job already submitted

- Request a standard function of a host's operating system
- Receive output from an executed job or send the output to another host
- Interrogate the status of a job, file, command, host, or user in the DDP system
- Send a message to an operator or user in another host

OS/3 to UNIX file transfer is a pairing of DDP transfer facility and IS/5000. It lets you:

- Create a file on any host in the DDP-IS/5000 network
- Copy a file from any host in the DDP-IS/5000 network to another host
- Remove a file from a host

The IMS DDP transaction facility allows you to easily define and perform remote transaction processing. Specifically, it supports the transfer of transactions from one IMS to another IMS, using the Unisys Distributed Communications Architecture.

This facility supports the following features:

- *Transaction routing* - Simple or dialog transactions can be routed on a DDP network. A simple transaction is a single-request and single-response exchange; a dialog transaction is composed of a series of request-response exchanges.

Routing mechanisms underlying the transfer of transactions from one transaction processing system to another within IMSTF are of the following types:

- *Directory routing* - Directory routing of a transaction is accomplished through a transaction ID directory. The transaction ID directory contains a list of valid transaction codes and associated information. This information determines if a transaction is to be processed locally or remotely. If the latter, a message of appropriate format is created and sent to the destination. Otherwise, processing proceeds as usual in the local environment. There is no need to change IMS action programs to use directory routing.
- *Operator routing* - A terminal operator can cause a transaction to be routed to a remote system by adding a previously defined special character to the beginning of the input message. After validation of the special character, the rest of processing is similar to directory routing. This makes it possible to perform transactions with the same transaction code on both local and remote systems. For example, UNIQUE (UNiform InQuiry Update Element) transactions can be processed at a remote system by adding the appropriate special character to the OPEN command (e.g., \*OPEN), or at a local system by omitting the special character.

- *Program routing* - Program routing allows action programs written in COBOL or basic assembly language (BAL) to determine where a transaction is to be processed. The action program identifies the remote system in its output message header, generates a message for delivery to the remote system, and issues an ACTIVATE function call to initiate the transaction. The action program processing the transaction on the remote system returns a message to the originating action program or its successor.
- *Remote transaction processing* - Transactions routed to a remote IMS system can be processed by UNIQUE or action programs written in COBOL, RPG II or BAL. Remote transactions are processed in the same way as local transactions. Most IMS features are available for remote transactions, including the use of formatted screen displays through screen format services.

Distributed data processing functions only in a DCA global environment, and when you define your network, you must specify a locap file for DDP. DDP also makes use of the ICAM DMI interface.





# Section 8

## Administrative Functions

### 8.1. Overview

In addition to providing you with the means for fulfilling your data communications requirements, ICAM also provides some optional administrative functions. These functions include collecting statistical data regarding the lines and terminals in your network and creating checkpoint records needed to restart ICAM in case of a software or hardware failure. The journal utility program is supported only in the standard and the transaction control interfaces. If you are using either the direct data or communications physical interfaces, you must write your own journal utility.

### 8.2. Journal Utility - Report Segment

The journal utility reporting facility can produce a summary report, a statistical report, or a report containing summary and statistical information selected optionally for a given set of conditions. The utility uses records created during the communications session by the ICAM administrative function (journaling). The utility itself does not require any interfaces or an ICAM environment. The summary report shows the line and terminal usage within ICAM. The statistical report shows the network buffer utilization data and the ARP network buffer utilization data.

Since the journal utility records are generated by ICAM, the journaling feature must have been specified at network definition time. The *ICAM Operations Guide* (UP-9745) shows the journaling parameters required at ICAM generation time, while the *ICAM Utilities Programming Guide* (UP-9748), describes the journaling utility and the reports it produces.

### 8.3. Journal Utility - Restart Segment

As we mentioned earlier, the journal utility can reconstruct the messages in case of a system hardware or software failure so that ICAM can be restarted. Reconstruction only involves complete messages; any incomplete messages must be reprocessed after the recovery operation.

## 8.4. Online Diagnostic Facilities

The online diagnostic and maintenance programs are supplied by Unisys to be used by maintenance engineers to locate the system hardware failure in peripheral subsystems supported by OS/3. The operator, however, may use these same programs to verify that the communications equipment is working. For a description of these programs, see the *System 80 Operator Maintenance Guide* (UP-8915).

## 8.5. System Activity Monitor

The system activity monitor, commonly known as SAM, is a system utility run as a symbiont that records and checks the activity of your system.

SAM is intended for the system administrator and installation manager to help detect bottlenecks, optimize production job mixes, and identify and change variables influencing system performance.

After SAM produces its report, the information can then be applied to certain system parameters achieving system optimization. Table 8-1 describes some of the information obtainable with the system activity monitor utility.

SAM is described in the *System Activity Monitor Programming Guide* (UP-9983).

Table 8-1. System Activity Monitor Available Data

Parameter	Description
CIMR	Indicates number of times per second communications by port were received by ICAM.
COMR	Indicates number of times per second communications by port were transmitted by ICAM
CINT	Indicates number of interrupts serviced from respective port
CSEN	Indicates number of sense commands issued by ICAM requesting further status information regarding interrupt marked as bad status completion
CERR	Indicates number of error conditions (time-outs, negative acknowledgments (DLE-NAK), or reply requests (DLE-ENO)) by port
CNOT	Indicates number of no-traffic responses received by port
CPOL	Indicates the number of polling interrupts per second
CBYT	Indicates the number of bytes per second transmitted over the communications line

The item readings under the communications class measure the activity of your communications system. They are:

- Input message rate
- Output message rate
- Number of interrupts
- Number of sense commands
- Number of error conditions
- Number of no-traffic responses
- Rate of poll interrupts
- Rate of bytes transmitted

### **8.5.1. Input Message Rate (CIMR)**

This value tells you the number of times per second that communications by port were received by ICAM during the specified interval. You can use this value to measure input message traffic in your communications system. For example, suppose that your communications system is configured with six terminals on each port on your system. A high input message rate may mean that there is too much message traffic for one port to service efficiently. By modifying your communications system (using more ports and with only two terminals per port), you can reduce the amount of message traffic over any one port and thus increase communication efficiency. Conversely, with a 6-terminal-per-port configuration, a low input message rate will signify a lack of message traffic. In this case, you may want to increase the number of terminals per port to more efficiently use each port.

### **8.5.2. Output Message Rate (COMR)**

This value tells you the number of times per second that communications by port were transmitted from ICAM during the specified interval. A high output message rate may mean that the number of terminals per port in your system is too high, resulting in a high rate of message traffic. By installing more ports and redistributing the number of terminals per port, you can reduce message traffic and increase communication efficiency.

### 8.5.3. Number of Interrupts (CINT)

This value tells you the number of interrupts that were serviced from the respective port during the specified interval. You can use this value to see which users are running jobs requiring substantial interrupt processing. For example, if the users on port 1 of a system were running a large number of I/O dependent jobs over the specified interval, a high number of interrupts would be serviced from port 1. If I/O dependent jobs from other ports were being run at the same time, a large amount of WAIT time would result since the jobs in the system would be I/O bound. One way to remedy this problem would be to tell the users on port 1 to run their I/O dependent jobs at another time, and run other jobs in their place. As a result, interrupt processing can be reduced and system efficiency increased.

### 8.5.4. Number of Sense Commands (CSEN)

This value tells you the number of sense commands that were issued by ICAM requesting further status information regarding an interrupt marked as a bad status completion. Sense commands are issued in response to error conditions and request status information from the system communication hardware. Therefore, a high number of sense commands indicates that there are a number of error conditions resulting from malfunctioning communications hardware.

### 8.5.5. Number of Error Commands (CERR)

This value tells you the number of error conditions (time-outs, negative acknowledgments, or reply requests) by port that occur during the specified interval. These error conditions indicate that there are transmission problems resulting from hardware malfunctions associated with the respective port. This value is provided only for ports handling UNISCOPE devices.

### 8.5.6. Number of No-Traffic Responses (CNOT)

This value tells you the number of no-traffic responses received by port during the specified interval. You can use this value to determine whether the terminals on your system are being used efficiently. For example, a high number of no-traffic responses from a particular port may mean that certain terminals on the port are not being used during much of the specified interval. In this case, you may want to reschedule extra work for this time interval to ensure a more efficient use of your system.

### 8.5.7. Rate of Poll Interrupts (CPOL)

This value tells you the rate per second of polling interrupts. Each terminal is polled to determine operational readiness and status, and to avoid contentions. For an idle period with a 1-second polling rate, the CPOL and the CNOT values equal a rate of 1.

### 8.5.8. Rate of Bytes Transmitted (CBYT)

This value tells you the number of bytes per second transmitted over the communications line. This value, in addition to the message text, includes the message byte count and all control and command bytes associated with line protocols. With the CBYT value, you can determine line utilization, where line utilization equals the ratio of total bytes transferred to the baud rate of the line.

## 8.6. ICAM Trace Facility

The ICAM trace facility provides a simplified way to monitor the operation of ICAM in a real life environment. It is a symbiont designed as a problem-solving tool to be loaded and used only as needed. Therefore, the diagnostic capability requires no main storage when ICAM is functioning normally and its monitoring capabilities are not needed. The trace facility has no ICAM network definition (CCA) requirements and the only OS/3 system generation requirement is that the system must include dynamic buffering.

Although the trace facility is a diagnostic tool, it was not designed for users to do their own troubleshooting. It is a tool to be used by both customers and Unisys personnel to accumulate records of ICAM functions for use by Unisys personnel on site or later by software development personnel.

You load the trace facility and specify the categories and number of trace events you want recorded. The trace facility acquires an area in main storage to save the trace events so that they can be analyzed later.

Four commands are supported:

- **ENABLE** - Specifies categories to be monitored
- **DISABLE** - Turns off specified category tracing or terminates tracing
- **STATUS** - Displays categories currently active
- **HELP** - Prints formats of all trace commands on the system console

Each category monitors critical points in a major area of ICAM. The categories supported are:

- **PHYSICAL** - Physical input/output control system (PIOCS)
- **LOGICAL** - Distributed communications architecture (DCA) structures
- **NETWORK** - Public data networks
- **QUEUER** - ICAM queueing
- **CONTROL** - ICAM activity control

You can specify one or all of the categories. However, if you have a problem with ICAM and you need to send a dump to Unisys for analysis, we recommend you specify only the categories where trouble is suspected.

Details on how to use the ICAM trace facility are provided in the *ICAM Utilities Programming Guide* (UP-9748).

### 8.7. ICAM Edit Dump

The ICAM edit dump (IED) utility is a symbiont that dumps selected groups of ICAM tables for diagnostic purposes. It supplements the OS/3 system dump (SYSDUMP).

You can use the edit dump to snapshot ICAM while it is running or any time ICAM has a program exception. When you take both an edit dump and a system dump, run the edit dump before the system dump.

The ICAM tables that you can dump are:

- General information tables
  - ICAM general information table
  - GUST general information table
  - Activity control queues
  - CCA address table
- Line link table
- ITF trace
- ICAM task control blocks
  - MCP task control block
  - Subtask task control block
- CCA control section
- ARP buffer pool
- Network buffer pool
- UDUCT (DCA data unit control table) buffer pool
- Link buffer pool
- Destination table

- End user tables
  - Communication user programs
  - Line vector table
  - Terminal control table
  - Process file
  - Distribution list
- Hexadecimal dump of CCA
- RIM (remote interface manager) queues
- Session analysis (session control entries)

To load the edit dump utility, type in:

```
IED
```

at the system console.

The console screen displays three messages in series. You must respond to each message before the next message is displayed. Edit dump executes after you respond to the last message.

Refer to the *ICAM Utilities Programming Guide* (UP-9748) for instructions on how to use the ICAM edit dump.





# Appendix A

## Coding Conventions

### A.1. Types of Macroinstructions

There are three types of macros available. They are:

- Declarative
- Imperative
- S-type

Declarative macros create nonexecutable code (tables, queues, interface areas, etc.) in your program or in ICAM. They also tell the OS/3 system generation process which ICAM modules to include in the ICAM message control program to support your communications system. All of the macros in your ICAM network definition are declarative (e.g., CCA, LINE, TERM). Only a few of the ICAM macros you use in your program are declarative (e.g., DTFCP, DLIST).

The imperative macros generate executable code sequences in a program. They request supervisor services or direct the operation of your program. Some ICAM imperative macros are: PUTCP, GETCP, and QDEPTH.

S-type macros let you separate the executable portion of a macro from the parameter list portion. This lets you save storage by creating fewer copies of the parameter portion when you call the same macros many times in your program.

### A.2. Declarative and Imperative Macroinstructions

The general format of declarative and imperative macros is:

LABEL	OPERATION	OPERAND
symbolic- name	macro mnemonic	operands

A symbolic name can appear in the label field. It can have a maximum of eight characters and must begin with an alphabetic character. If used, it must begin in column 1.

The appropriate macro mnemonic must appear in the operation field to identify the operation or service requested. At least one space must separate the operation field from the label field and the operand field.

When operands are specified in the operand field, they must be positional or keyword operands as required by the particular function. This field optionally begins in column 16. If the operation field is more than five characters, a space must be inserted before the operand begins. If the operand field is to have a continuation on the succeeding line, place any nonnull character in column 72 and continue the operands or comment starting in column 16.

You must follow the assembler rules regarding blank columns and continuation of the operand field.

Do not separate operands by using spaces; you must use commas.

### A.2.1. Positional Operands

You must write positional operands in the order specified in the operand field, and they must be separated by commas. When a positional operand is omitted, the comma must be retained to indicate the omission, except in the case of omitted trailing operands.

#### Example

The TRMREP macro has three mandatory positional operands (line-name, terminal-name, workarea-address) and one optional operand (FIELDS=CALL).

#### Format

LABEL	OPERATION	OPERAND
[symbol]	TRMREP	[line-name, terminal-name, workarea-address [, FIELDS=CALL]

Using some arbitrary symbols (tags), you could write this macro:

```
LABEL1 TRMREP LNE1,TRM1,TRWA  
LABEL2 TRMREP LNE1,TRM1,TRWA,FIELDS=(666-5708)
```

## A.2.2. Keyword Operands

A keyword operand consists of a word or a code immediately followed by an equal sign, which is, in turn, followed by a specification. Keyword operands can be written in any order in the operand field. Commas are required only to separate operands.

### Example

The DTFCP macro has one mandatory keyword operand (TYPE) and six optional keyword operands (UNIT, LEVEL, NOMAV, ERRET, DATIME, and NOTLST).

### Format

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	DTFCP	TYPE=GT [,UNIT=LINE] [ ,LEVEL= { LOW MEDIUM HIGH AVAIL } ] [,NOMAV=symbol] [,ERRET=symbol] [,DATIME=YES] [,NOTLST=symbol]

Again, using arbitrary symbols, you could write this macro:

```
DTFCP  TYPE=GT,UNIT=LINE,LEVEL=HIGH
DTFCP  TYPE=GT
DTFCP  TYPE=GT,UNIT=LINE,NOMAV=NOHIT,ERRET=ERR1
DTFCP  TYPE=GT,LEVEL=AVAIL,NOMAV=NOHIT,DATIME=YES
```

## A.3. Macroinstruction Coding Conventions

The conventions used to delineate macros are:

- Capital letters, commas, parentheses, and equal signs must be coded exactly as shown.

### Examples

```
R
ALL
(1)
SIZE=
```

- Lowercase letters and words are generic terms representing information that you must supply. Such lowercase terms may contain hyphens and acronyms (for readability).

### Examples

```
name
start-addr
number-of-bytes
param-1
ccb-name
```

- Information contained within braces implies a choice of entries - one of which must be chosen, unless the operand is surrounded by brackets or one of the choices is shaded (i.e., a default).

### Examples

```
{ PC }
{ IT }
{ AB }

{input-area}
{ (1) }
```

- Information contained within brackets represents optional entries that (depending upon program requirements) are included or omitted. Braces within brackets (with no default) signify that one of the specified entries must be chosen if that operand is to be included.

### Examples

```
[,entry-number]
[,R]
[,ERROR=symbol]
[,WAIT=YES]

[ , { ccb-name } ]
[ , { ALL } ]
[ , { (1) } ]
```

- An ellipsis (series of three periods) indicates the presence of a variable number of entries.

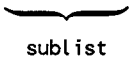
### Example

```
ccb-name-1, ..., ccb-name-n
```

- An operand may consist of a sublist of parameters called suboperands, which are separated by commas. If a suboperand is omitted, the comma must be retained, except in the case of trailing suboperands.

**Example**

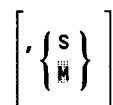
```
NET1 CCA TYPE=(GBL,,NODB),PASSWORD=ABC...
```


  
sublist

- An operand that has a list of entries may have a default that is supplied by the operating system when you do not specify the operand. When a default specification occurs in the format delineation, it is printed on a shaded background.

**Example**

```
[ , { S } ]
```



## A.4. S-Type Macroinstructions

You can save main storage and simplify your program by using S-type macros. If you need to use the same macro many times and only a few (or none) of the fields vary, you may specify the L-form of the S-type macro to create a separate reusable copy of the parameter list. You should specify the parameter list in the declarative portion of your program because it contains no executable code. You then specify as many E-form S-type macros as you need. Each E-form macro creates executable code, and you should place these in the executable portion of your program. Each E-form macro you specify points to the common parameter list. Your program modifies the parameter list as necessary prior to each call of the macro.

Not all ICAM macros have the S-type feature. However, you can recognize S-type macros because they all use the MF keyword operand.

### A.4.1. L-Form S-Type Macroinstruction

The L-form S-type macro generates a parameter list at the place the macro is encountered. When you code your program, you should place L-form S-type macros among your define constant (DC) or define storage (DS) statements because they do not receive control and contain no executable code. The format of the L-form S-type macro is:

LABEL	ΔOPERANDA	OPERAND
symbol	operation	MF=L

**Label**

symbol

Is a required entry because it becomes the label of the generated parameter list.

**Operand**

MF=L

Specifies this is an L-form S-type macro and a parameter list is required.

**A.4.2. E-Form S-Type Macroinstruction**

The E-form S-type macro creates an executable instruction that points to a previously generated parameter list. Normally it points to a parameter list created by an L-form S-type macro. However, you can generate the parameter list any way you want to. The format of the E-form S-type macro is:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	operation	MF= { (E,parameter-list) } { (1) }

**Operands**

parameter-list

Specifies the symbolic address of a parameter list associated with this macro.

(1)

Specifies general register 1 contains the address of the parameter list associated with this macro.

**A.4.3. SD-Type Macroinstruction**

The SD-type macro is an extension of the S-type macro. It provides a form of the MF operand that generates a description of the data structures associated with the macro. For example, you use it to obtain a listing of a parameter list.

In addition, you use this form to create DSECTs of the macro called that vary only with the first character of each symbolic label. The format of the SD-type macro is:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	operation	MF=(D { ,prefix-code } ) * I

## Operands

**MF=D**

Specifies a DSECT statement is to precede the actual data description.

**prefix-code**

Is a 1-character prefix that is appended to all labels generated by the macro. If you omit this operand, the prefix-code defaults to the letter I. To suppress the prefix character, you specify an asterisk (\*).





# Glossary

This glossary contains definitions of many terms peculiar to ICAM and communications programming.

## A

### **ABM (asynchronous balanced mode)**

An operational mode of universal data link control (UDLC). In this mode, either of two UDLC combined stations can initiate transmission to the other without requiring permission from the other combined station. This mode provides highly efficient communications and equal control capability to the two combined stations on a point-to-point link.

### **ACK**

The affirmative reply character response required in certain message disciplines and message switching procedures to acknowledge receipt of a message. ACK is an acceptable reply to indicate that line conditions and the status of messages are normal.

### **ACON**

An address constant

### **ARP (activity request packet)**

A series of constants and formatted storage locations for communication requests and parameters between modules and/or user programs

### **ASCII**

American Standard Code for Information Interchange

### **ASR**

Automatic send/receive unit usually related with TELETYPE equipment for automatic operation as opposed to keyboard send/receive (KSR)

### **asynchronous (nonsynchronous)**

A method of transmission or a type of equipment that uses essential timing information in each character transmitted

### B

#### **batch mode**

A means of communicating with data communications terminals in which messages are sent consecutively from or to the terminal without individual polling of each message

#### **baud**

A unit of modulation rate used in data transmission to signify the number of discrete signal events per second

#### **BCC (block check character)**

An error checking character that is typically the exclusive OR of all characters in a blocked message

#### **BCW (buffer control word)**

Word located in privileged low-order main storage that contains current buffer address and character count for input and output messages

#### **bit-oriented procedures**

The recently developed data link protocols whose data granularity is at the bit level rather than at the character or byte level, as with previous protocols such as ISO basic mode or UNISCOPE terminal protocol. Also implied by the term are the characteristics of full-duplex, mixed terminal connections and high efficiency. These procedures include, among others, HDLC, ADCCP, and SDLC.

#### **bps**

Bits per second

#### **BSC (binary synchronous communications)**

An IBM-developed protocol for synchronous transmission of binary-coded data. One of the first protocols to allow for transparent text transmission.

#### **buffer(ing)**

A temporary storage area used to collect and contain data while it is being received from, or prior to its transfer to, a communication line. Buffering the data, as it is called, is a method of compensating for the difference in speed between the transmission rate of the communications line and the transfer rate of the processor channel.

#### **buffer control**

A routine that supplies and makes the most efficient use of data storage areas allocated to each CCA under the control of ICAM

**C****CCA (communication control area)**

A software element generated dynamically in the main storage load area for a predefined network after execution of the NETREQ macroinstruction. In the ICAM system, this area is created by the network definition macroinstructions.

**CCR (channel control routine)**

A software module of the ICAM system that has as primary function the generation, execution, and completion status determination of all I/O functions to be presented to the communications hardware. The interaction of the CCR makes it unnecessary for the user to have to distinguish between the characteristics of a communications device and the characteristics of the line to which the device is connected.

**CCW (channel command word)**

Control word used by the hardware in giving commands to the channel and locating data areas

**CDM (consolidated data management)**

The name of the System 80 data management system

**channel**

A path along which data is sent or received. The central processing unit has two types of channels: selector and multiplexer

**character code conversion**

Character-by-character translation from one character code convention (e.g., EBCDIC, ASCII, or XS-3) to another

**checkpoint**

A point in a program routine or subroutine at which the status of the program or equipment is determined. The program may be rerun from this checkpoint by using a recovery procedure.

**circuit switching**

In public data network, providing a dedicated connection between two data terminal equipment (DTEs) for the period of message transmission. The connection is then cleared until the next message is ready for transmission. Protocol is in accordance with the CCITT X.21 standard.

**clocking**

Timing that requires a series of fixed intervals

**CMCS**

COBOL message control system

**CNC (communications network controller)**

A software module of ICAM, with the major function of controlling message traffic entering/leaving the system in such a way as to avoid overload connections

## Glossary

---

**communications dispatcher**

A module or routine that controls SIO function execution, packet checks, and chaining

**communications multiplexer module**

A module that contains the hardware required to service line adapter input to multiplexer channels

**communications program**

A program written by the user to process communications applications. The program uses a predefined set of macroinstructions within the applications program coding to control all the necessary I/O functions for message traffic via communications lines.

**completion mask**

A mask sent with CPIOCP that is used in masking out the completion returns from the CCR

**compression**

The process of removing blanks from text. Compression is used to shorten messages that are to be transmitted in order to increase line performance.

**connect**

A routine that is common to all handlers and is responsible for line connection. The connect routine issues turn-on and dial commands to the channel control routine.

**COP (communications output printer)**

An auxiliary printer for display terminals

**CPIOCP (Communications physical input/output control packet)**

A software routine containing the necessary information for driving the single line communications adapter through the CPIOCS.

**CPIOCS**

Communications physical input/output control system

**CS (communications system)**

The total environment over which DCA controls the logical structure as well as the interfaces and protocols. Logically, the communications system encompasses the transport network (TN) and all the connected termination systems, but not the communications system users and their end users that attach to it.

**CSU (communications system user)**

The applications-related control structure, external to DCA, that interfaces to the communications system through one or more ports. Communications system users control one or more end users, directing data and commands to and from them.

**cursor**

A symbol used to indicate the position of the next character entry on an alphanumeric display terminal

**D****data**

A representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation, or processing by human or automatic means

**datagram**

A communications program or a noncommunications program can activate a yielded communications program by means of a GAWAKE macroinstruction. At this time, the program issuing the GAWAKE macroinstruction can send a message, called a datagram, to the program being activated.

**DCA (distributed communications architecture)**

The Unisys architecture that draws together all aspects of the communications products by defining a set of logical concepts and a set of rules (protocols and interfaces) and guidelines to be used in applying the concepts in the design of hardware, software, and network products.

**DCE (data circuit terminating equipment)**

The carrier equipment in a public data network

**DCT 475**

A DCT 500 Series data communications terminal that operates in TTY mode and has no auxiliary devices

**DCT 500**

An unbuffered, asynchronous keyboard/printer data communications terminal

**DCT 1000**

A fully buffered, incremental printer terminal with optional keyboard

**DCT 2000**

A printer/reader/punch available also as a printer terminal

**DDI (direct data interface)**

An ICAM interface that provides communications capability in a minimum configuration. The user program interfaces directly with the remote device handlers via a message control table.

**decompression**

The process of inserting blanks into a text that has been compressed. The insertion of the blanks is done in such a format as to restore the text to its original content.

**dequeue**

The process of removing entries from a queue

**demand mode interface**

The system interface that supports both distributed data processing and OS/3 interactive services

**device statistics log**

A record that keeps a count of various device errors, total messages sent and received, and other device type statistics. The device statistics log is kept in a user's communications control area, maintained by device handlers, and structured by line.

**DICE (device independent control expression)**

A 4-character sequence found in the text portion of a communications message that can control the function and position of remote terminal devices

**did**

Device identification (See also *rid*, *sid*.)

**distribution list (DLIST)**

A list of destinations for messages that consists of terminal, process file, or Locap file symbolic names. The list is created by a DLIST macroinstruction whose operands are the symbolic names of the destinations.

**DTE (data terminating equipment)**

User equipment that accesses a public data network

**DTFCP (define the file xx)**

A declarative macroinstruction specifically identifying the nature of a given file for an input or output device identified by *xx*, where *CP* specifies a communications program file

**DUST (deferred user service task)**

A module that is activated by nonmessage service requests to control the activity between elements of the ICAM system and the user program initiating the request. *LNREQ* is an example of a nonmessage request.

**dynamic session**

Data transfer between paired end users defined within user programs or between paired terminals dynamically

**E****EBCDIC**

Extended Binary Coded Decimal Interchange Code

**ENQ (enquiry)**

Used as a request for a response to obtain identification or an indication of station status. Transmitted as part of an initialization sequence (line bid) in point-to-point operation and as the final character of a selection or polling sequence in multipoint operation

**EOF**

End of file

**EOM (end of message)**

Last character in a blocked message. In EBCDIC, the code is  $19_{16}$ .

**EOT (end of transmission)**

Indicates the end of a transmission, which may include one or more messages, and resets all stations on the line to control mode (unless it erroneously occurs within a transmission block). EOT is also transmitted as a negative response to a polling sequence.

**error detection and correction**

A hardware or software module capable of detecting, identifying, and correcting data errors occurring during transmission. Most often, the modules initiate a procedure to retransmit the data in an attempt to correct the erroneous data transmission.

**ETB**

End of transmission block

**ETX (end of text)**

A character that signals the end of text in a blocked message; an  $03_{16}$  in both ASCII and EBCDIC

**EU (end user)**

With distributed communications architecture (DCA) these are the sources and sinks of information across the network. Physically they are the programs, devices, and/or operators that generate and receive the data transmitted over a DCA communications system. They may not be the ultimate users of the information, but are those users closest to the communications system itself; hence their existence, if not their characteristics, is defined within the architecture. Collections of end users are under the control of communications system users (CSU) who interface to one or more logical ports.

**extended sequence**

Term used in describing the condition that exists when a device is busy (e.g., a printer on a display terminal). The terminal is busy during this time.

## F

### flag

A binary indicator set by hardware or software to convey status information for a given circumstance, value, path, etc.

## G

### GUST (global user service task)

A system task that controls all resources in a global environment

## H

### header

That part of a message containing the information for controlling and directing the text portion of the message in its various routings (destination or process queues)

## I

### IA

Interrupt answering

### ICAM (integrated communications access method)

A generalized software package and set of prescribed procedures affording the programmer multiple levels of interface to remote devices and data files/queues

### IDES (ICAM device emulation system)

Permits use of System 80 in extended tasks through emulation of remote terminals

### immediate return line

A control bit in an activity request packet that causes transfer of control immediately back to the user program without initiating an I/O sequence. The I/O is released for execution when the first packet with the bit reset is received.

### IMS

Information management system (IMS). A software product facilitating the development and installation of online, transaction-oriented, data base management applications under OS/3.

### interface

The logical unit (hardware or software) functioning as the interconnecting link between two systems or devices of different characteristics. For example, a data communications subsystem forms the common boundary between the modem and the input channel to a computer.



**interrupt**

The means by which the central processor temporarily suspends hardware execution of one task to perform another. Interrupt processing gives a computer the power to carry out simultaneous operations and still recognize priority conditions, thereby increasing the overall efficiency of the system.

**IRL**

Immediate return line

**J****journal**

A historical file of complete messages or message segments that is kept by the user's message processing procedure specifications

**K****KSR (keyboard send and receive)**

A telecommunication industry standard for a terminal device that produces a typewritten copy of the message to be sent or received. A number following the letters KSR identifies a specific model equipped with certain features or characteristics.

**L****LCT**

Line control table

**line**

A dedicated or switched telephone communications path between two modems

**line control table**

A table of entries, created by execution of the LINE macroinstruction, used for controlling and processing message traffic to or from the associated line queues

**locap (local application) file**

Specifies the queues necessary to permit program-to-program transfer, whether local or remote, and supplies the name and type of program that other applications may address for access

**logging**

The action of keeping a journal

### **logical subchannel**

A data path from one node (computer) to another node. It may be linked through one or more network processors or packet switching networks routing data through this subchannel from a source node to a destination node. Only one session can be established through one logical subchannel.

### **LRC (longitudinal redundancy character)**

Same as the block check character (BCC)

## **M**

### **machine/program check**

The hardware interrupt generated when a particular type of hardware or software error occurs

### **macro definition**

A method of generalizing a set of instructions, a program, or a routine that can be particularized for a given application by selecting a series of optional parameters in a macroinstruction that calls upon it

### **MCT**

Message control table

### **memory protect**

A method of supplying an immunity to illegal access. For instance, in ICAM, access to the communications control area is permitted only by the supervisor and the ICAM modules

### **message**

The binary-coded data or information exchanged over communication lines between two or more terminals. A message usually is composed of three parts:

1. Header - may contain any or all of the following: data source, destinations, timing, date, routing, transmission signals, and synchronization controls
2. Body - the data or information to be communicated
3. Ending - a control character to indicate end of message

### **message control table**

An activity request packet for direct data interface and higher level interfaces

### **message logging**

The process of recording message activity in a history file that can later be used by accounting and diagnostic routines. Logging usually refers to the header data of a journal file.

**message retrieval**

A means of accessing a message, other than the next message available for processing, after it is placed in a queue

**message switching**

The general classification of a switching system in which the destination addresses of messages are included as a portion of the message itself (normally the leading character or header)

**modem (data set)**

An instrument used by the common carrier to modulate communications signals, transmit them, and demodulate the signals when receiving them

**module**

A segment of hardware or software that is, in itself, a separate and complete logical entity but is normally combined or linked with other modules before it can operate functionally. As an example, an assembler turns out an object module that usually is linked with other object modules to form a load module that becomes the executable program.

**monitoring**

The action of reporting various conditions at selected decision points in device handlers. The conditions reported are kept in a monitor area that is accessible by a maintenance program. Monitoring is performed while the handlers are servicing their various devices.

**MPPS (message processing procedure specification)**

A set of macroinstructions that the user specifies in a given sequence for analyzing and controlling incoming and outgoing messages on a line or lines, provided the messages contain the same characteristics

**multidrop**

Several drops or connections (such as multiple terminals) on a communications line

**multiplex**

To interleave two or more messages on a single channel or line

**multiplexer channel**

I/O channel that can concurrently service many low-speed subchannels

**MUST (message user service task)**

An interface service, activated by GETTCP/PUTTCP requests, to control the transfer of data between ICAM elements and the user program

**mux subchannel**

A physical connection on a multiplexer channel to which a control unit may be connected

## N

### node

Defines a computer as an element in a multicomputer communications network

### NTR

An ICAM utility that enables remote batch processing to a Unisys Series 1100 data processing system. The utility permits type-in operation for standard reader, punch, printer, and device-dependent peripherals, as well as user-own-code programs, to handle device independent peripherals such as tape and disk.

## O

### OS/3

A sophisticated, multitasking operating system with extended capabilities to meet the needs of simplex or complex processing environments

### overrun

The indicator that is set when the channel does not acknowledge receipt of a data byte or send a data byte when requested during a specified time period

## P

### packet switching

In a public data network, the sharing of physical resources (links and facilities) among many users of a public data network. On demand or at subscription time, a virtual circuit is established between the host processor and the remote node for the exchange of information. The public data network controls when the data is transmitted, the size and format of data frames sent across the link, and error procedures in accordance with the CCITT X.25 international recommendation.

### password

A means of verifying the right of a user program to access a computer system or files associated with the system

### PCI (program-controlled interrupt)

One of the three interrupt conditions that cause control to be passed to the CCR. This interrupt is used to indicate that a buffer is exhausted, either empty or full, or that an additional buffer may be required to continue processing a message.

### PDN (public data network)

A network that includes new or future transmission facilities supplied by a PTT or common carrier for transferring data (as opposed to use of the telephone network for this purpose). PDNs can be packet switched, digital switched, or some combination of both. (See also X.25.)

**POLL**

The message sent across a specified line to a terminal or group of terminals to solicit input or status

**polling group**

A group of terminals that can be accessed by a single poll since they have the same rid. For example, all stations on a communications line that recognize the rid and sid of a poll message. Each station in such a group will recognize a general rid as its rid address and a general sid as its sid address. A poll containing a general rid, general sid, and general did is called a general poll. Another form of general poll is one containing a specific rid, general sid, and general did.

**port**

A logical path between a local node and the final destination in the destination node

**pre/postqueuing**

An action performed by the ICAM message user service task. It concerns the collection (prequeuing) of text segments into a complete message before giving the message to the communications network controller for destination or process file queuing. Postqueuing is the removal of text segments from a queued message.

**probe**

A message sent to a remote device that provides coordination with the device so that the device remains online

**process file**

A set of queues defined for a network that then points to the input files placed on those queues

**protected format**

A feature that provides a means of protecting selected data fields on a terminal from operator alterations. In other words, there are character fields that are protected and cannot be altered.

**protocol**

A set of rules defining the structure, content, sequencing procedures, and error detection and recovery techniques for the transmission of data. A protocol is also used to establish, maintain, and control communications between two corresponding levels in a level hierarchy. Normally implies the sending and receiving of unique command and response messages or message headers. (Contrast with interface.)

**PVC (permanent virtual circuit)**

A logical path between host and remote node. This path differs from a switched virtual circuit (SVC) in that it is statically established at subscription time.

## Q

### **queue(ing)**

The sequencing of messages in a storage medium by placing the addresses of related messages in an individual file according to a common destination

### **queue, delayed or held**

A queue permitting messages to be queued but inhibiting the queue from transmission of its data, regardless of the destination of the messages until released

### **queue, intercept**

A queue assigned to a particular terminal because the terminal is temporarily overloaded and cannot accept more traffic. The terminal would retain exclusive use of the assigned queue for the time it remained overloaded. Traffic for active terminals using the original queue can continue to have messages processed.

### **queuing, line**

Assigning messages to a specified line queue serving one or more terminals

### **queuing, message**

Staging and linking message segments in a main storage or a disk storage area associated with a designated CCA

### **queuing, priority control**

A control permitting messages to be processed in a sequence other than in the sequence of arrival in the system

### **queuing, terminal**

Assigning messages to a specific terminal

## R

### **RBP (remote batch processing)**

Type of processing where batch type jobs are submitted (and optionally receive back output) from a remote site card reader, printer, or punch. The RBP is also the name of an ICAM utility program that performs this service.

### **RDH (remote device handler)**

A program to control and direct message traffic being sent to and received from terminal, sharing a common set of characteristics

### **response time**

The time, set by hardware or software timers, in which a reply or a response is expected to an inquiry or other message transmission. Expiration of the response time usually results in the execution of a procedure for dealing with an abnormal condition.

**rid, sid, did**

A hierarchy or series of identifiers structured to identify a terminal or group of terminals in a telecommunications network

rid is an acronym for remote identifier. A transmission code assigned to a location where a terminal or number of terminals reside. This is the first level of host processor addressing (rid, sid, and then did).

sid is an acronym for station identifier, a transmission code assigned an individual terminal. This is the second level of host processor addressing (rid, sid, and then did).

did is an acronym for device identifier. A transmission code assigned to a peripheral device to identify it for the terminal or host. This is the third level of host processor addressing (rid, sid, and then did).

**route**

The path that a message, or data, follows from the point at which transmission begins until the arrival at a final destination

**RPG II**

A programming language that uses preprinted formats to generate reports. The language provides extended processing capabilities for data handling.

**S****SAT**

System access technique for disk files

**segment**

A portion of a message generally the size of a network buffer

**selector channel**

An I/O channel that handles a single high-speed subsystem

**session, session path**

The logical path through the complete network from one end user to another that includes both the port session and transport network session as well as any internal associations within the CSU environment.

**sid**

A unique station identifier presented to the processor by a remote site during the initialization of that site (See *rid, sid, did.*)

**SOE (start of entry)**

A hardware or software character that identifies the beginning of the area to be transmitted to the processor or to the auxiliary interface from a display terminal

**SOH (start of header)**

A character signaling the start of header for a blocked message

**SOM (start of message)**

First character in a blocked message; usually the same as the SOH character

**static session**

Data transfer between fixed, paired, end users defined at network definition time

**status, hardware**

An indicator or set of indicators that displays the state of a particular device or function

**status, software**

A byte or set of bytes that receives a value denoting the state of a particular device or function

**STX**

Start-of-text character

**subscriber**

In a public data network, any remote node using a public data network

**SVC**

A privileged instruction that produces a hardware interrupt, thus giving control to the supervisor, which, in turn, processes the request for service

**switched virtual circuit**

In a packet-switched public data network, it is a dynamically established logical path between a host computer and a remote node.

**switching**

As in message switching, switching is the routing or directing of messages through a central system to their final destinations.

**symbiont**

Symbionts are software modules that can be called by systems console and workstation commands or through job control statements. They are controlled by the supervisor portion of the executive system and do not occupy job slots. They can access input/output devices and files and effect multiple I/O functions. They are normally executed in response to a user request. ICAM is a symbiont.

**sync (synchronization character)**

Receipt of this character synchronizes a modem for the data following



**synchronous**

A method of transmission or a type of equipment in which sending and receiving units are synchronized by periodically exchanging sync characters prior to the actual movement of data. This method precludes insertion of timing information in each character, as required in asynchronous transmissions.

**SYSGEN**

The creation of a computer operating system as performed by a systems analyst. The output of a SYSGEN program is a working executive.

**system program**

A software program provided by Unisys

**T****TCI**

An interactive interface designed especially for IMS applications

**TCS (tape cassette subsystem)**

An auxiliary magnetic tape system for a display terminal.

**terminal**

One of many names given to data input or output equipment or facilities at one end of a communications channel

**terminal table**

A compilation of entries, resulting from the execution of TERM macroinstructions, used for controlling message traffic to and from terminals using queues.

**text**

The character transmission to or from a remote device minus any header or framing characters

**TN (transport network)**

The logical architectural entity that is responsible for the transfer of network data units between the various attached termination systems. Note that network control, while physically in the network processors, is not within separate termination systems of the transport network.

**toggling**

To switch or alternate between two or more work areas (buffers) to obtain or output a complete message. While the receiving program or routine (local) is processing the data of an already filled buffer, the sending routine (remote) is filling the alternate buffer. When both routines are finished processing, the switching (toggling) occurs. Efficiency of processing is increased since the sending routine never has to wait for the receiving buffer to be emptied.

**TP**

A designation for a terminal printer auxiliary device connected to a display terminal

**traffic summary log**

A record of message traffic through the network or on a line

**transaction**

In a real-time mode of operation, each sequence of one input message from a terminal followed by one output response from a host constitutes one transaction.

**transaction control interface**

Interactive inquiry/response interface for IMS

**transaction terminal table**

An activity request table for the transaction control interface

**translation**

The exchange of character sets of one binary code for those of another; for example, a translation from EBCDIC to ASCII. The exchange also is called code conversion.

**transmission**

The electrical transfer of a signal, message, or other form of information from one location to another

**TS (termination system)**

Term used with distributed communications architecture. A collection of associated communication system ports, including an applications management services (AMS) port and central process unit (CPU). Association can be by physical constraint (that is, all in one box) or logical (that is, belonging to the same virtual machine).

**trunk**

A logical path between a host processor and a packet-switched public data network. There is one DTE address for each trunk.

**TTY**

A designation for teletypewriters

**turnaround time**

The time required to reverse the direction of transmission on a half-duplex carrier facility. During a turnaround operation, which is controlled by the data set, the facility is not available for transmission in either direction.

**U****UDLC (universal data link control)**

Unisys bit-oriented line protocol implementation that handles HDLC, ADCCP, and SDLC protocols as subsets

**unattended answering**

The ability of a communications receiver to accept incoming traffic without human intervention; the counterpart of automatic dialing

**UNISCOPE**

A display terminal that includes keyboard and CRT

**user program**

A software program provided by the user (See communication program.)

**UTS 400**

UTS 400 Universal Terminal System. A general purpose, microprocessor-based remote display terminal

**UTS 400 TE**

UTS 400 Text Editor. A special purpose microprocessor-based remote terminal designed for the printing and publishing industry

**UTS 4000**

A family of terminals and cluster controllers. They include the UTS 20 and UTS 40 terminals and UTS 20W and UTS 40W workstations connected to UTS 4020 or UTS 4040 cluster controllers.

**V****virtual line**

The hardware connecting two nodes directly or connecting one node to a network processor or packet switching network. Each virtual line can support up to 4096 logical subchannels.

**VLINE**

See virtual line.

## X

### X.25

The CCITT recommendation for interfacing to packet-switched PDNs. This standard is in three parts:

- Level 1: X.21 (digital) X.21 bis (modem); the electrical interface to the line-terminating unit
- Level 2: Line protocol from the customer's equipment (DTE) to the exchange (DCE). This is one of the HDLC Codes of Practice.
- Level 3: Call access protocol to establish, maintain, and control virtual calls to and from the network

# Index

## A

- activate a communications program and pass a message, 2-13
- activity request packets (ARP)
  - description, 5-7
  - statistics in a dump, (figure) 5-8
- addressing terminal
  - description, 3-10
  - general identification, 4-14
  - I/O devices, 4-21
  - polling, 4-8
  - rid, sid, did, 4-10
- asynchronous balanced mode, 7-7, (figure) 7-8

## B

- basic assembly language
  - function, 2-18
  - user-written program interface, 2-15
- bit
  - start, 3-17
  - stop, 3-17
- buffers, line
  - function, 5-3
  - length defaults, 5-4
  - terminal requirements, 5-4
  - VLINE, 5-5
- buffers, network
  - description, 5-9
  - disk queueing, 5-12
  - function, 5-9
  - insurance factor, 5-16
  - I/O errors, 5-20
  - main storage queueing, 5-10
  - message storage, 5-9
  - MPPS interaction, 6-1
  - number, 5-14
  - obtaining statistics, 5-17, (figure) 5-19
  - size, 5-12
  - threshold capability, 5-18

## C

- CDM, See consolidated data management.
- circuit-switched public data network, 3-27, (figure) 3-28, (table) 3-29
- circuits
  - dedicated, 3-22
  - dedicated vs switched, 3-24
  - permanent virtual, 3-31
  - switched, 3-24
  - switched virtual, 3-31
  - virtual, 3-23, 3-31
- closed user group, 3-32
- COBOL
  - CMCS/ICAM environment, (figure) 2-27
  - message control system (CMCS), 2-14
  - programs, 2-26
  - user-written program interface, 2-15
- coding conventions, A-1
- communications adapter, single line
  - description, 3-2
  - synchronization, 3-18
  - types available, (table) 3-34
- communications lines, 3-20
- communications physical interface
  - function, 2-5
  - general, (figure) 2-4
- communications systems
  - ICAM, (figure) 1-3
  - queue arrangement, 5-22
- communications user programs
  - Unisys supplied, 2-29
  - user-written, 2-15
- concepts of distributed communications
  - architecture, 7-2
- consolidated data management, 2-21

## D

- data circuit terminating equipment, 3-26,  
(figure) 3-27
- data terminal equipment, 3-26,  
(figure) 3-28
- data/time stamp, 6-3
- DATAPAC Canadian public data network  
(table) 3-32
- DATEX-P German public data networks  
(table) 3-32
- DCT 500, DCT 524
  - description, 4-20
  - disadvantages, 4-21
  - error notification, 4-31
- DCT 1000, error notification, 4-32
- DDX-P Japanese public data network  
(table) 3-32
- dedicated circuits, 3-22
- demand mode interface, 7-6
- device dependent control characters, 4-23
- device independent control expressions  
(DICE)
  - description, 4-24
  - DICE macroinstructions, 4-27, (table)  
4-28
  - hexadecimal notation codes and functions,  
(table) 4-26
  - summary, 4-29
- device types, input/output, 4-21
- DICE, See device independent control  
expressions.
- direct data interface (DDI)
  - change line phone number, 2-20
  - contrast with standard interface, 5-1,  
(figure) 5-2
  - error notification, input, 4-31
  - error notification, output, 4-32
  - function, 2-5
  - general, (figure) 2-4
  - logical components of ICAM, (figure) 2-4,  
2-17
  - message sending and receiving, 2-23
  - polling interval, 4-12
  - program buffers, 2-23
  - relationship with RDH, 4-1
  - standard interface, comparison with  
2-15
  - user-written programs, 2-15

- distributed communications architecture  
(DCA)
  - concepts, 7-2
  - description, 7-1
  - multinode networks, 7-6
- distributed communications processor, 7-7,  
(figure) 7-9

## E

- error notification
  - input, 4-31
  - output, 4-32
- error recovery, MPPS, 6-4
- errors
  - journal utility, 6-4
  - message header, 6-4
  - sequence number, 6-5
  - transmission, 6-4

## F

- files
  - disk-buffered 5-1
  - disk-queued, 5-1
- format edit 4-30
- formatting your data device dependent  
control characters, 4-24
- device independent control expressions  
4-24
- general description, 4-23

## G

- GAWAKE
  - operand, 2-11
  - use, 2-13
- global networks
  - comparison with dedicated, 2-5
  - description, 2-7
  - single node, 2-7
  - multinode, 2-7
  - shared with dedicated networks, 2-7
  - use of GUST, 2-8
- global user service task (GUST)
  - request/release ICAM facilities, 2-8
  - use with global networks, 2-8

**H**

- hardware, communications
  - communications adapter, single line, 3-33
  - communications lines, 3-20
  - description, 3-1
  - System 80, 3-1
  - terminals, 3-3
- hexadecimal notation, for DICE, 4-26

**I**

- IBERPAC public data network, (table) 3-32
- IBM 3270 emulator, 2-40
- ICAM
  - communications system, OS/3
    - (figure) 1-3
  - definition, 1-1
  - internals and interfaces, 2-1
  - similarity to data management, 1-1
  - software program, 2-1
  - supported terminals, (table) 3-8
  - symbiont, 2-13
  - symbiont definition, 1-2
- ICAM declarative/imperative macros
  - DTFCP, 2-21
  - GETCP, 2-21
  - PUTCP, 2-21
- ICAM device emulation system (IDES) utility
  - components, 2-36, (figure) 2-37
- ICAM edit dump, 8-6
- ICAM trace facility, 8-5
- information management system (IMS),
  - 2-30, (figure) 2-32
- input/output microprocessor
  - description, 3-2, 3-3
  - System 80, 3-33
  - System 80 interface, (figure) 3-33
- interfaces
  - communications physical, 2-5
  - direct data, 2-5
  - standard message control program, 2-2
  - transaction control, 2-2

**J**

- journal utility
  - description, 8-1
  - report segment, 8-1
  - restart segment, 8-1

**L**

- languages
  - basic assembly language, 2-18
  - COBOL, 2-26
  - RPG II, 2-27
- line connections
  - broken, 4-7
  - procedure, 4-4
  - reconnecting broken lines, 4-7
- line control methods
  - controlled, 3-18
  - protocols, 3-18
  - uncontrolled, 3-18
- line queuing, 5-24
- lines
  - automatic dialing, 4-5
  - dedicated, 4-5
  - manual dialing, 4-5
  - unattended answering, 4-6
- LOCAP files
  - description, 5-38
  - end users, 2-10
  - sending messages from programs, 5-43

**M**

- macroinstructions
  - coding conventions, A-3
  - cross reference, (table) B-1,
  - format, A-2
  - keyword operands, A-3
  - positional operands, A-2
  - S-TYPE, A-5
  - SD-TYPE, A-6
  - types, A-1
- maintenance, diagnostic facilities, 8-2
- MAPPER 5 system
  - programs supplied, 2-29
  - connecting to OS/3, 2-42, (figure) 2-3

message  
  buffer stored, 5-9  
  characteristics, (figure) 5-29  
  delivery notification, 4-33  
  input, 5-9  
  output, 5-10

message formatting characters  
  BCC - block check character, 4-14  
  EOT - end of transmission character, 3-11  
  ESC - escape character, 3-11  
  ETB - end-of-transmission block character  
    3-11  
  ETX - end-of-text character, 3-9  
  SI - shift in, 3-11  
  SOH - start of header, 3-11  
  STX - start-of-text character, 3-9  
  US - unit separator, 3-12  
  VT - vertical tab, 3-11

message control table, 2-23

message header  
  MPPS effects, 6-2  
  prefix, 5-12

message processing procedure specification  
  (MPPS)  
  analyzing message headers, 6-3  
  error recovery, 6-4  
  function and use, 6-1  
  macroinstructions, 6-1  
  message sequence, 6-3  
  message source, 6-4  
  message type, 6-4  
  miscellaneous functions, 6-7  
  routing information, 6-6

modems, 3-1

multinode global networks, 7-6  
multiplexer, polling groups, 4-9

## N

network, dedicated  
  comparison to global, 2-5  
  definition, 2-6  
  limitations, 2-6  
  reconnecting broken lines, 4-7  
network, global  
  comparison to dedicated, 2-5  
  definition, 2-7  
  reconnecting broken lines, 4-7

networks, circuit-switched public data, 3-27  
networks, global vs dedicated, 2-5  
nine thousand remote (NTR) system utility  
  components, 2-38, (figure) 2-39

## O

output delivery notification request, 4-33

## P

packet-switched public data network, 3-29,  
  (table) 3-32

polling, terminal  
  algorithms, 4-13  
  buffered interactive, 4-14  
  description, 4-8  
  efficiency, 4-16  
  errors and error recovery, 4-17  
  groups, 4-8, 4-9, 4-18  
  interval, 4-11  
  speed, 4-16  
  unbuffered interactive, 4-20

program activation (GAWAKE), 2-13

protocol  
  buffered interactive terminals, 4-13  
  polling, 3-18  
  terminal support, 4-1

PSS United Kingdom, (table) 3-32

public data network  
  Canadian (DATAPAC), (table) 3-32  
  circuit-switched, 3-27, (figure) 3-28  
  Denmark, Finland, Sweden, Norway  
    (NORDIC), 3-28  
  French (TRANSPAC), (table) 3-32  
  general description, 3-26, (figure) 3-26  
  German (DATEX-L), (table) 3-29  
  German (DATEX-P), (table) 3-32  
  IBERPAC, (table) 3-32  
  Japanese (DDX-P), (table) 3-32  
  packet-switched, 3-29, (figure) 3-30  
  United Kingdom (PSS), (table) 3-32



## Q

- queuing, disk
  - advantages/disadvantages, 5-11
  - recommendations for interactive, 5-13
- queuing, main storage
  - advantages/disadvantages, 5-11
  - description, 5-11
  - recommendations, 5-12
- queues associated with a program or terminal, 5-22
  - description, 5-22
  - function, 5-1
  - input queue arrangement, 5-22
  - messages sent from programs, 5-42
  - messages sent from terminals, 5-41
  - priorities, 5-31
  - types, 5-31
  - use of line or terminal, 5-24
- queues, program
  - description, 5-25
  - process files, 5-35
- queues, terminal
  - description, 5-22
  - output, 5-33

## R

- remote batch processing (RBP) utility
  - components, 2-34, (figure) 2-35
- remote device handler (RDH)
  - function, 4-1
  - line buffers, 5-3
  - MPPS interaction, 6-1
  - polling, 4-8
  - status and error codes, 4-31
  - translate tables, 4-31
- remote job entry, 2-34
- remote terminal processor (RTP), 2-14
- remote workstations, (table) 3-1, 3-8
- routing, MPPS message, 6-6
- RPG II
  - batch processing, 2-28
  - interactive processing, 2-28
  - user-written program interface, 2-15

## S

- session
  - definition, 2-10
  - dynamic, 2-10
  - establishment (SESCON macro), 2-11
  - static, 2-10
- single-line communications adapter, 3-33, (table) 3-34
- slow polling, 4-13
- standard interface buffers and queues, 5-1
  - change line phone number, 2-20
  - contrast with DDI, 5-1, (figure) 5-2
  - contrast with direct data interface, 2-15
  - dynamic session capability, 2-11
  - global networks, 2-7
  - logical components of ICAM, (figure) 2-16
  - message sending and receiving, 2-20
  - network buffers, 4-3
  - polling inhibit feature, 4-13
  - RDH interaction, 4-2
- standard message control program (STDMCP), 2-2
- status and error codes, 4-31
- switched lines, 3-24
- switched virtual circuits, 3-31
- synchronizing transmission
  - asynchronous, 3-17
  - synchronous, 3-16
- system activity monitor
  - description, 8-2
  - input message rate (CIMR), 8-3
  - number of error commands (CERR), 8-4
  - number of interrupts (CINT), 8-4
  - number of no-traffic responses (CNOT), 8-4
  - number of sense commands (CSEN), 8-4
  - output message rate (COMR), 8-3
  - rate of bytes transmitted (CBYT), 8-5
  - rate of poll interrupts (CPOL), 8-4
- System 80 basic communications system
  - components, 3-1, (figure) 3-2
  - single-line communications adapter, 3-33, (table) 3-34

**T**

- Telcon network, 7-7, (figure) 7-9
- terminal
  - additional terminal interfaces, 3-20
  - addressing, 3-10
  - controller, 3-5
  - distributed data processing, 3-4
  - description, 3-2
  - hardware, 3-3
  - interface characteristics, 3-9
  - line control, 3-18
  - message formatting, 3-9
  - queuing, 5-22
  - statistics, function, 4-33
  - synchronizing transmission, 3-16
  - uses, 3-7
- terminal, batch, 5-14
- terminal communications direction
  - 1-way (simplex), 3-13
  - 2-way alternate (half-duplex), 3-14
  - 2-way simultaneous (full duplex), 3-14
- terminals supported, (table) 3-8
- threshold, 5-18
- transaction control interface (TCI)
  - buffers and queues, 5-1
  - function, 2-2
  - general, (figure) 2-4
  - RDH interaction, 4-1
- translate tables, 4-31
- TRANSPAC French public data network
  - (table) 3-32
- types of macroinstructions, A-1

**U**

- universal data link control (UDLC), 7-7,  
(figure) 7-4, 7-9
- universal terminal system 4000 (UTS 4000),  
(figure) 3-6
- utility programs
  - COBOL message control system (CMCS),  
2-14
  - ICAM device emulation system (IDES)  
2-14
  - nine thousand remote (NTR), 2-14
  - remote batch processing (RBP), 2-14
  - remote terminal processor (RTP), 2-14
- UNIX operating system, 2-30, 2-42

**V**

- VLINE
  - description, 3-22
  - elements, 5-6
  - protocols, 3-24
  - use, 7-6

**Y**

- yield, program control, 2-13

## USER COMMENTS

We will use your comments to improve subsequent editions.

NOTE: Please do not use this form as an order blank.

---

*(Document Title)*

---

*(Document No.)*

---

*(Revision No.)*

---

*(Update Level)*

### Comments:

**From:**

---

*(Name of User)*

---

*(Business Address)*

CUT

FOLD



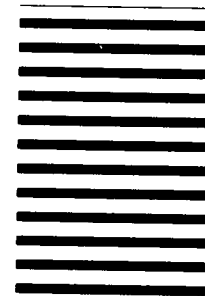
NO POSTAGE  
NECESSARY  
IF MAILED IN THE  
UNITED STATES

# BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

Unisys Corporation  
E/MSG Product Information Development  
PO Box 500 — E5-114  
Blue Bell, PA 19422-9990



FOLD



